

VC SpyGlass Lint®

Tags Reference Guide

Version S-2021.09-SP2, March 2022



Copyright Notice and Proprietary Information

© 2022 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/company/legal/trademarks-brands.html>.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

Contents

Preface.....	65

About This Book	65

Typographical Conventions	66

Using VC SpyGlass Lint Product.....	67

VC SpyGlass Lint Parameters	68

allowedOpPrecedence.....	68
allowBuffer	68
allowInverter	68
allow_indirect_types	69
allow_indirect_directions	69
allow_tie_cell_list	70
allow_latch_cell.....	70
allow_module_list.....	70
allowedScanChainLength	71
ALLOWED_DRIVER_LIMIT	71
attach_violation_tag	71
apply_on_lib_cell_only	72
avoid_nba_always_latch	72
avoid_seq_logic.....	73
avoid_synth_disable	73
bigmux_max.....	74
check_assign_pattern	74
case_option	74
casex_option	75
casez_option.....	75
case_inside_option	76
check_all_modules	76
check_unread	76
check_below_32.....	77
check_bit_casting.....	77
check_directive_type	78
check_escaped_names	78

check_enum_usage	79
check_globally_defined.....	79
check_include_files	79
check_inst_connections	80
check_input_port.....	80
check_lrm_and_natural_width	81
check_loop_breakers.....	81
check_natural_width_for_static	82
check_natural_width_of_div	83
check_natural_width_of_modulo.....	83
check_natural_width_of_multiplication	83
check_non_static_index.....	84
check_only_missing_default.....	84
check_only_top_module	85
check_parameter.....	85
check_static_natural_width	85
check_task_ports.....	86
checkinalldu	86
checkpwdpin	87
checkRTLInst.....	87
check_top_ports.....	87
checkv2kparamsonly	88
checkInstName	88
clockgate_filename	89
CLOCKNET	89
clk_rst_net	90
CONSIDER_SUPPLY_AS_DRIVER.....	90
comment_acc_max_length	90
combloop_waiver_compat.....	91
concat_size_based_width.....	91
design_attributes.....	92
disable_rtl_deadcode_extended	92
disable_seq_clock_prop	92
dw_name	93
enable_rtl_deadcode	93
encoding_type.....	94
excluded_check_port.....	94
flag_only_instance_ports	94
fanout_factor	95
fanoutmax	95
file_hdr_ignore_path	95

filter_mark_open	96
generate_connpnt_report	96
REPORT_D_FF_COUNT	96
REPORT_ALL_DRIVERS	97
REPORT_ALL_POSEDGE_NEGEDGE_PAIRS	97
REPORT_FILE_LINE_ON_SOURCE	97
report_gated_clock_name	98
REPORT_ONLY_USER_NETS	98
REPORT_UNDRIVEN_RESET	98
REPORT_NONUSERMUX	99
REPORT_UNDRIVEN_CLOCK	99
REPORT_UNUSEDFLOP_COUNT	99
SKIP_MODULE	100
traverse_through_pad_pin	100
traverse_fanin_for_src_net	100
ignore_nets	101
infertop	101
IGNORE_LATCH	102
instance_based_pin_names	102
instance_name_netlist	103
ignore_nonstatic_counter	103
instance_prefix	103
instance_suffix	104
allow_types	104
report_only_libcell	105
ignore_async_blocks	105
ignore_auto_function	106
IGNORE_SAME_SUPPLY_SOURCE	106
IGNORE_SAME_DRIVERS	106
ignore_blocking_hanging_flop	107
ignore_case_compare_op	107
ignore_clk_generator	107
ignore_if_case_statement	108
ignore_chain_cell	108
ignore_chain_cell	108
ignore_const_zero_only	109
ignore_hanging_flop	109
ignore_integer	110
ignore_latch_cell	110
ignore_libcell_inst	110
ignore_libcell_traversal	111

ignore_out_term.....	111
ignore_parameter_type	111
ignore_dim_port_types.....	112
ignore_sub_modport	112
ignore_same_case_item	112
ignore_system_tasks	113
ignore_user_defined_tasks	113
ignore_unique_and_priority.....	114
report_case_type.....	114
ignore_concat_expr	114
ignore_parameter_assignment	115
ignore_vector_bitselect	115
ignore_wildcard_operators	116
MAX_DRIVER_COUNT.....	116
report_blocking_only	116
ignore_parameters.....	117
REPORT_CONST_CONNECT	117
report_internal_netname	117
report_hanging_cell	118
report_local_name	118
report_inferred_cell	119
report_regoutputs_ports	120
strict.....	120
supplyHigh.....	121
supplyLow	121
supplyNetFile	122
case_condition_count	122
consider_sgdc_clock.....	123
checkunusedlatch	123
effort_level	124
ignoreCellName	124
ignoreModName	125
latch_chain_max_length	125
module_based_pin_names.....	126
overlappingLatchLoops	126
check_complete_sensitivity_list	127
check_seq_assign	127
effort_level_sim_race	128
handle_large_bus	128
ignore_typedefs.....	128
waiver_compat.....	129

cell_check_for_race	129
checkcomblogic	130
check_mux_select_pins	130
consider_cont_assign	131
delaymax	131
delaymax_err	132
delaymax_floptoflop	133
delaymax_floptomem	133
delaymax_floptooutput	133
delaymax_inputtoflop	134
delaymax_inputtooutput	134
delaymax_memtoflop	135
delaymax_memtomem	135
ignore_icg_cell	136
ldto and ldfrom	136
logmux_max	138
report_common_source	138
show_connected_net	138
show_viol_count	139
allow_clk_in_condition	139
allviol	140
assume_driver_load	140
avoid_port_signal	141
casesize	141
checkblocking	142
checkfullbus	142
checkfullrecord	143
checkfullstruct	144
check_bbox_driver	144
check_case_type	145
check_complete_design	145
check_concat_max_width	146
checkconstassign	146
check_const_selector	147
check_counter_assignment	147
checkDriverInModule	149
checknonblocking	149
check_genvar	150
check_implicit_senselist	150
check_initialization_assignment	151
check_latch	151

checkOperatorOverload	151
check_next_state	152
check_pattern	153
CHECKSIBLING	153
check_param_association	153
consider_lrm_for_div	154
consider_sub_as_add	154
check_shifted_only	155
check_shifted_width.....	155
check_sign_extend	156
check_static_value.....	156
checksynchronreset.....	159
check_sequential	159
check_temporary_flop	159
check_unsign_overflow.....	160
CHECK_Z_ASSIGN	160
concat_width_nf	161
considerInoutAsOutput	161
control_sig_detection_nf.....	162
datapath_or_control.....	162
do_not_run_W71	163
define_module_list.....	163
DISABLE_CLOCK_SORTING	164
disable_rtl_deadcode	164
disable_signal_usage_report	165
dump_array_bits	165
effort_simloop_level.....	166
fast.....	166
flag_reset_type	167
flag_unconnected_out_flops	167
force_handle_shift_op	168
handle_case_select.....	168
handle_equivalent_drivers	169
handle_large_expr.....	169
handle_lrm_param_in_shift.....	170
handle_shift_op.....	170
handle_static_caselabels.....	172
handle_zero_padding	173
hier_path_length_limit	173
ignore_auto_function_return	173
ignore_below_line.....	174

ignore_cell_list	174
ignore_bitwiseor_assignment	175
ignore_nonstatic_leftshift	175
ignore_cond_having_identifier	175
IGNORE_CONDITIONAL_EXPRESSIONS	176
ignore_counter_with_same_width	176
ignore_equivalent_drivers	176
ignore_forloop_indexes	177
ignore_genvar	177
ignore_generatefor_index	178
ignore_greybox_drivers	178
ignore_hier_scope_var	179
ignore_in_ports	179
ignore_inout	179
ignore_integer_const_variables	180
ignore_internal_signal	180
ignore_flops	180
ignore_fullcase_default	181
ignore_local_variables	181
ignoreModuleInstance	182
ignore_nonBlockCondition	182
ignore_macro_to_nonmacro	182
ignore_multi_assign_in_forloop	183
ignore_param_type	183
ignore_pli_tasks_and_functions	184
IGNORE_SYNC_RESET_FOR_ASYNC_RESET_FLOP	184
ignore_reinitialization	185
ignore_scope_names	185
IGNORE_WITHIN_RANGE_LABELS	185
ignoreSeqProcess	186
latch_reset	186
limit_task_function_scope	186
modport_compat	187
net_count	187
nocheckoverflow	187
not_used_signal	188
only_apply_on_leaf_cell	188
process_complete_condop	189
report_all_connections	189
report_all_hier	190
report_all_messages	190

report_all_unpacked	190
report_at_constant	191
report_blackbox_inst.....	191
report_clock_reset_loops	192
report_declarations	192
report_global_param.....	192
report_hierarchy.....	193
REPORT_LOOP_VAR_EXPR	193
report_less_severity.....	194
report_nonblocking_in_error	194
report_one_hier	194
report_only_bitwise	195
report_only_overflow	195
report_cast.....	196
report_if_blocks_only	196
report_inter_nba	197
report_port_net.....	197
report_single_hierarchy	198
report_single_violation	198
report_sync_rst_only	198
report_to_integer	199
reportLibLatch	199
reportconstassign	199
reportundrivenout.....	200
reset_gen_du.....	200
reset_gen_module	201
report_semicolon.....	201
report_struct_name_only.....	201
reportsimilarassgn	202
REPORT_ONLY_SVCONSTRUCT	202
RESETNET	203
runcapa.....	203
set_message_severity	203
SETNET	204
sign_extend_func_names	204
simplesense.....	205
traverse_function.....	205
treat_concat_assign_separately	206
treat_latch_as_combinational	206
thresholdScanChainLength	206
use_carry_bit.....	207

use_lrm_width	208
use_natural_width	208
verilint_compat	208
W416_vhdl_only.....	209
allow_combo_cloud.....	209
allow_combo_logic_base	210
avoid_generated_supply	210
buf_count.....	211
capa_checkwithinblock	211
capa_max_line_count	211
capa_scope	212
capa_similarity.....	212
capa_size	213
casedepth.....	213
categorize_violations.....	214
checkallblocks	214
check_all	214
check_all_pragmas	215
check_blackboxes.....	215
checkcontassign	216
check_dup_label.....	216
check_enum_decl.....	217
check_gate_primitives.....	217
checkInHierarchy.....	218
check_conversion_function	218
check_default_value	219
check_complex_operators.....	219
check_conditional_assignment	220
check_default_state.....	220
check_names_space	220
check_nested_decl	221
check_op_in_flop	221
check_output_ports	222
checkTopDu	222
checkGateInst.....	223
check_relational_operators	223
check_ifdef_define_include	224
check_in_top	225
check_lhs_rhs_assign_width	225
checktaskarg	225
check_type.....	226

check_unused_typedef	226
check_xassign_casedefault	227
checkregisterdu	227
check_register_type	227
check_unloaded_ports	228
chkTopModule	228
check_var_decl_init	230
comment_string	230
consider_inout	231
depth_ml	231
dccompat	232
define_file_name	232
DEFINE_SAME_NET_START_POINT	232
delta_delay_mode	233
disable_vhdl_keywords	233
dump_ifdef_data	234
directivecheck	234
enum_typedef_regex	235
excluded_cell_list	235
other_typedef_regex	236
struct_typedef_regex	236
union_typedef_regex	236
effort_dissolve_brackets	237
evaluate_for_loop_index	237
FUSE_PIN	238
filter_named_resets	238
reset_reduce_pessimism	238
genlabeltype	239
global_timescale	239
handle_const_expressions	240
handle_greybox	240
handle_hier_clock_reset	240
handle_shift_in_concat	241
handle_signed_truncation	241
hierarchical_name_length	242
ignore_deliberately_unconnected	242
ignore_directive_names	243
ignore_dup_label_stmt	243
ignore_elements	243
ignore_enum_index	244
ignore_explicit_if_blocks	244

ignoreforloop	245
ignore_generate_blocks.....	245
ignore_genvar_condition	246
ignore_inverters	246
ignore_interface_locals.....	246
ignore_inoutport_expr.....	247
ignore_latches	247
ignore_loop_index	248
ignore_module	248
ignore_mult_and_div	248
ignore_multidim.....	249
ignore_non_static_cond.....	249
ignore_non_vector_signals	249
ignore_single_zero	250
ignore_static_condition	250
ignore_sibling_viol.....	251
ignore_sync_reset	251
ignore_top_level_pre_assignment.....	252
ignore_unloaded_inst.....	253
ignore_unloaded_port	253
ignore_unused_flop	254
ignoreforindex.....	254
ignoreInteger.....	254
inoutlevel	255
int_net_fanout	255
inv_count	256
inout_name	256
input_name	256
ignore_initial_blocks	257
latch_name	257
max_delta_delay	257
mem_size.....	258
min_based_width	258
minShiftRegLength	259
modulename.....	259
OperatorCount	260
output_name	260
pragma_list_ml	260
prohibited_data_types.....	261
range_bound	262
report_all_diff_timescale	262

report_all_objects.....	263
report_all_params	263
report_all_pins	264
report_always_cond	264
report_case_block	265
report_default_branch	265
report_for_submodules.....	265
report_floating_source	266
report_generate_block	266
report_hier_path	267
report_if_block.....	267
report_mark_open.....	268
report_max_val_overflow	268
report_onehot_case	269
report_non_const_read	269
report_nonstatic_assign.....	270
report_open_port	270
report_only_first.....	271
report_only_from_one_hierarchy	271
report_sequential_always	271
report_underflow	272
report_underflow_expressions	273
runcapaTag	273
skip_lib_cell_checking	273
showReducedSchematic.....	274
states_to_check	275
synch_reset_list	275
synth_dir_list.....	276
synth_name_length	276
use_detailed_message	277
use_sgdc_reset	277
vector_size	278
word_size	278
wrapperDuCellList.....	279
actlowname	279
allReserveWords	280
archname	280
asyncname	281
check_globals	281
check_task	281
check_regoutput_modules	282

checkalldimension	282
checkLowToHigh	283
check_reserve_word	283
checkTopLevel.....	284
clkname	284
clkname_vector.....	285
comment_max_length.....	286
constname.....	286
enableE2Q.....	286
enable_latch_based_clusters.....	287
funcname	287
gates	288
generate_header_check_report	288
header_check_file.....	289
header_check_file_vh	290
ignore_case_inside	290
ignore_cells	291
ignore_conv_function.....	291
ignore_file	291
ignore_function_port.....	292
ignore_file_ext.....	292
ignore_hanging_flop	292
ignore_iopad.....	293
ignore_incr_decr_by_one	293
ignoreInversionOperation	293
ignoreRealLatch.....	294
ignore_libcells.....	294
ignore_macro_names.....	294
ignore_mux_select	295
ignore_non_reset	295
ignore_libcell_inout_ports.....	296
ignore_modules.....	296
ignoreMultipleInstances.....	296
ignoreNonStaticOverlap.....	297
ignore_memory.....	298
ignoreSRlatch	298
ignore_regoutput_modules	299
ignore_stmt.....	299
indent_level.....	300
inst_max_length	300
inst_min_length	300

invalid_cell_list_warning	301
instname	301
iocells	302
line_max_length	302
limit_if_branch	303
name_max_length	303
name_min_length	304
nocheck_always_init	304
nocheck_with_package	304
no_scripts	305
num_max_fsm_states	305
overlappingLoops	306
only_top_gate_list	306
overlapsimloops	307
override_module_list	307
paramname	307
package_header_check_file	308
processblk_linecount	308
portname	309
portorder	309
procname	311
print_recommended_regexp	312
pwrdsnpin	312
PNDTREE_CELL_LIST	312
regoutname	313
report_all_actlow	313
reportalwayslatch	314
report_flop_clock_loop	314
report_flop_reset_loop	314
reportlatchedoutput	315
report_localparam	315
reportHangingLatch	316
reportLatchHierarchy	316
ignoreLatchHierarchy	317
report_mux_select	317
report_nonstatic_actlow	317
reserve_names	318
reserve_names_vh	318
resetname	319
SigHierName_option	319
sighiernamelist	320

signame	320
snake_length	321
snakepath_ff	321
through	322
topgates	322
triname	322
typename	323
uniq_char_val	323
valid_cell_list	324
varname	324
alwaysblk_optcount	325
arithrel_op_maxsize	325
assign_check	325
assume_driver_load_srp	326
bitwidth_basedconst	327
bit_width_reduction	327
check_all_always	328
check_any_bit_for_xz	328
check_case_keyword	328
check_clock_cell	329
check_dependency	329
checkDriverInDu	330
check_pad_concat	330
checkFirstInstance	330
check_index	331
check_internally_generated_clk	331
check_var_in_condition	332
clkGeneratorName	332
clk_EnableLatch	333
clk_gen_du	333
comb_logic_style	333
const_string	334
fixedinputenableflop	334
flag_only_induction_var	335
functask_match_global_var	335
handle_latch_setreset	336
handle_negation_semantics	336
ignore_based_width	337
ignore_cell	337
ignore_direct_connection	338
ignoreElseIf	338

ignore_empty_lines.....	338
ignore_file_with_multiple_modules	339
ignore_fsm_counter	339
ignore_macro.....	340
ignore_one_hot.....	340
ignore_one_hot_labels	340
ignore_pad_cell.....	342
ignore_parameter_in_port	342
ignore_sv_keywords	342
ignore_vector_bit_selection.....	343
lib_sync_cells.....	343
line_length_max.....	344
maxLoopIter	344
nesting_level.....	344
num_max_bit_changes.....	345
numerator_size	345
param_string	345
processblk_opcount	346
report_allclk.....	346
report_all_datatype.....	347
report_instance_pin	347
reportallnets	348
report_all_rst.....	349
report_full_path	349
report_reset_type.....	349
report_typedef	350
report_violation_type	350
reset_synchronizer_modname	351
rtl_acc_max_length	351
src_master_dir.....	352
src_master_dir_vh.....	352
srlatch.....	352
srp_gen_1316_report.....	353
sr_max	353
starc_file_ext_vhdl	353
starc_file_ext_vlog.....	354
starc_inst_name.....	354
starc_through_block	355
starc2005_always_line1.....	355
starc2005_always_line2.....	355
starc2005_arc_name.....	356

starc_check_sequential_block	356
starc2005_compare_str.....	357
starc2005_define_max	357
starc2005_define_min	358
starc2005_define_string	358
starc2005_define_viol	358
starc2005_ext_string_vhdl.....	359
starc2005_ext_string_vlog.....	359
starc2005_function_max	360
starc2005_function_min	360
starc2005_gate_maxcount.....	361
starc2005_ifelse_branchlevel_reco.....	361
starc2005_ifelse_chainlevel_reco	362
starc2005_ifelse_nestlevel_reco	362
starc2005_maxCaseClause	362
starc2005_maxCondBitWidth	363
starc2005_maxInputSize	364
starc2005_maxOutBitWidth	364
starc2005_maxTableSize	365
starc2005_max_inst_length.....	365
starc2005_max_mod_ent_char_length	366
starc2005_max_mod_ent_length.....	366
starc2005_max_port_char_length.....	366
starc2005_min_inst_length.....	367
starc2005_min_mod_ent_length	367
starc2005_negative_str.....	368
starc2005_number_for_indent	368
starc2005_num_max_fsm_states	369
starc2005_parameter_max	369
starc2005_parameter_min	369
starc2005_port_max.....	370
starc2005_port_min.....	370
starc2005_port_maxcount	370
starc_report_casez	371
priority_set_reset	371
starc2005_process_line1	371
starc2005_process_line2	372
starc2005_reset_string	372
starc2005_signal_max	373
starc2005_signal_min	373
starc2005_size_of_mult_max.....	373

starc2005_top_mod_ent_name	374
starc2005_top_port_name	374
starc_use_natural_width	374
srict_constsig	375
suggest_always_also	375
tag_arguments	376
top_gate_list	376
TRANSMISSION_GATE_CELL_LIST	377
traverse_through_mux	377
tristate_count	377
typedef_regex	378
unsized_param_cutoff	378
user_defined_resets	378
use_inferred_resets	379
use_new_message	379
use_secondary_message	379
valid_conn_types	380
violate_on_mixed_mode	380
viol_wrapper_module	380
whenelse_cnt	381
matchtype	381
latch_clock	382
latch_verbose	382
latch_clock_file	382
latch_reset_file	383
overlappingLatchLoops	383
allow_combo	384
check_depth	384
delay_cell_list	385
ignore_digital_pad	385
ignoreRtlBuffer	385
margin_value	386
spare_cell_name	386
ENABLE_CHECKING_ON_ICGC	387
VC SpyGlass Lint Application Variables	388
Product Reports	389
Rule_24_Report	389
Rule_28_Report	389
all_register_info Report	390
glitch_circuit	391

FlopDataX : Flip-flop data pin driven by an indeterminate value.....	458
FlopSREX : Flip-flop set, reset or enable pin driven by an indeterminate value.....	460
NearOverload : Load on Gate output is close to recommended maximum .	462
Overload : Load on Gate output exceeds recommended maximum	465
Underload : Load on Gate output falls below recommended minimum .	468
checkNetReceiver : Each internal net must have at least one receiver.	471
LatchDataX : Latch data pin driven by an indeterminate value.....	472
LatchEnableX : Latch enable pin driven by an indeterminate value	474
LatchEnableUndriven : Latch enable pin not driven	476
noCombinatorialFeedBack : No combinatorial feedback loops allowed. ...	478
pwrDnPinConnToSeqOrIOCells : PWRDN pins of cells should be connected to either IO cells or sequential elements.	480
resetPinConnectedToResetNet : “Reset Pins” of cells may only be connected to signals predefined in a list of reset nets.....	483
setPinConnectedToSetNet : “Set Pins” of cells may only be connected to signals predefined in a list of set nets	485
clockPinsConnectedToClkNets : “Clock pins” of cells may only be connected to signals predefined in a list of clock-nets	487
checkOPPinConnectedToNet_a : At least one output port of a cell must be connected to a net.....	489
checkOPPinConnectedToNet_b : All the output ports of the cell must be connected to a net.	490
elementsAllowedPerScanChain : List all scan chains, which contain more than n elements, as violating scan chains.....	492
Latch Tags	494
ClockEdges : Reports design units that use both labels of the clock to drive latches	495
LatchFeedback : Reports a latch in which a combinational feedback path exists from output pin to data or enable pin.....	500
LatchGatedClock : Reports latches driven by gated or internally generated clock	504
LatchReset : Reports design units in which the reset pins are used both synchronously and asynchronously.....	510
W122L : Variable read inside a latch always block should be inside sensitivity list	515

W442bL	: Asynchronous reset condition can only use simple condition.	517
W450L	: Reports multi-bit expression used as latch enable condition ...	518
W336L	: Blocking assignment to latch output should be avoided	520
W392aL	: Do not use both edges of an asynchronous reset	522
W392bL	: Do not use both edges of a synchronous reset	525
W428L	: Task should not be called inside a sequential always block.....	529
W442aL	: First statement in an asynchronous always block should be an `if` statement.....	531
W429L	: Task should not be called inside a combinational always block	533
W449L	: Expression used as latch enable may not be synthesizable	535
GatedReset	: Do not use gated or internally generated resets to reset the latches	537
Lint Tags		540
AllocExpr	: Identifies the allocator expressions which are not synthesizable	541
ArrayEnumIndex	: Identifies the arrays using enumeration type as index	543
AssertStmt	: Assertion statements have no significance in synthesis ...	545
badimplicitSM1	: Identifies the sequential logic in a non-synthesizable modelling style where clock and reset cannot be inferred	546
badimplicitSM2	: Identifies the implicit sequential logic in a non- synthesizable modeling style where states are not updated on the same clock phase	549
badimplicitSM4	: Identifies the non-synthesizable implicit sequential logic where event control expressions have multiple edges	552
BlockHeader	: Identifies ports and generics in the block statement header which are not synthesizable	554
bothedges	: Identifies the variable whose both the edges are used in an event control list.....	557
BothPhase	: Identifies the processes that are driven by both the edges of a clock	559
ClockStyle	: A clocking style is used which may not be synthesizable ..	562
DisconnSpec	: Identifies the disconnection specification constructs which are not synthesizable	564
EntityStmt	: Statements in entity block may be ignored by some synthesis tools	567
ForLoopWait	: Identifies the WAIT statements used in FOR-loop constructs which are not synthesizable	568
IncompleteType	: Identifies the incomplete type declarations which are	

not synthesizable	571
infinite loop : While/forever loop has no break control	574
InitPorts : Default initial value of in/out/inout port may be ignored by some synthesis tools.....	576
IntGeneric : Identifies the non-integer type used in the declaration of a generic which is not synthesizable	577
LinkagePort : Identifies the linkage ports which are not synthesizable.	579
LoopBound : Identifies the for loop range bounds that are not locally or globally static	581
mixedsenselist : Mixed conditions in sensitivity list may not be synthesizable (Verilog) Edge and level conditions are mixed in if statement (VHDL).	583
MultipleWait : Identifies multiple wait statements having the same clock expression which are not synthesizable	585
NoTimeOut : Identifies the timeout expression in a wait statement, which is not synthesizable.....	587
PortType : Identifies ports of unconstrained types which are not synthesizable	590
PhysicalTypes : Identifies the physical constructs which are not synthesizable	592
PreDefAttr : Identifies the pre-defined attributes which are not synthesizable	595
readclock : Unsynthesizable implicit sequential logic: clock read inside always block.	597
ResFunction : Identifies the resolution functions which are not synthesizable	598
SigVarInit : Identifies the initial values of signals and variables which are not synthesizable	600
SynthIfStmt : Identifies the IF statements which are not synthesizable	602
UserDefAttr : Identifies the user-defined attributes which are not synthesizable	605
W107 : Do not make bus connections to primitive gates (and, or, xor, nand, nor, xnor)	608
W110 : Identifies a module instance port connection that has incompatible width as compared to the port definition	610
W110a : Use same port index bounds in component instantiation and entity declaration.....	615
W111 : Not all elements of an array are read.....	620

W116 :	Identifies the unequal length operands in the bit-wise logical, arithmetic, and ternary operators.....	623
W120 :	A variable has been defined but is not used (Verilog) A signal/variable has been declared but is not used (VHDL) .	635
W121 :	A variable name collides with and may shadow another variable ...	640
W122 :	A signal is read inside a combinational always block but is not included in the sensitivity list (Verilog) A signal is read inside a combinational process but is not included in the sensitivity list (VHDL)	644
W123 :	Identifies the signals and variables that are read but not set ...	653
W127 :	Delay values should not contain X (unknown value) or Z (high-impedance state).....	662
W126 :	Do not use non-integer delays.....	664
W162 :	Extension of bits in constant integer conversion	665
W164c :	LHS width is greater than RHS width of assignment (Extension) ..	666
W242 :	This Tag has been deprecated (Verilog) A function is calling itself; that is, it is recursive (VHDL)	673
W328 :	Truncation in constant conversion, without loss of data.....	674
W341 :	Constant will be 0-extended	675
W446 :	Output port signal is being read (within the module)	677
W456 :	A signal is included in the sensitivity list of a combinational always block but not all of its bits are read in that block (Verilog) A signal is included in the sensitivity list of a process but not all of its bits are read in that block (VHDL)	679
W488 :	A bus variable appears in the sensitivity list but not all bits of the bus are read in the contained block (Verilog) An array signal appears in the sensitivity list but not all bits of the array are read in the process (VHDL).....	683
W494a :	Input port is not used.....	685
W494b :	Output port is not used.....	686
W500 :	A connection to a clock port is not a simple name	687
W526 :	Use case statements rather than if/else, where feasible, if performance is important.....	689
W529 :	`ifdef is not supported by all tools	692
W128 :	Avoid using negative delays	693
W129 :	Variable delay values should be avoided	694
W143 :	Macro has been redefined	696

W146 : Use named-association rather than positional association to connect to an instance	698
W154 : Do not declare nets implicitly.....	700
W156 : Do not connect buses in reverse order	701
W159 : Condition contains a constant expression	704
W163 : Truncation of bits in constant integer conversion	706
W164a : Identifies assignments in which the LHS width is less than the RHS width	707
W164a_a : LHS width is less than RHS width of assignment (Hard Mismatch).....	732
W164a_b : LHS width is less than RHS width of assignment (Soft Mismatch)	745
W164b : Identifies assignments in which the LHS width is greater than the RHS width.....	754
W167 : Module has no input or output ports	773
W17 : Prefer full range of a bus/array in sensitivity list. Avoid bits or slices	774
W171 : Case label is non-constant.	776
W175 : A parameter/generic has been defined but is not used	778
W18 : Do not infer latches.....	779
W182c : Identifies the time declarations which are not synthesizable ...	781
W182g : Identifies the tri0 net declarations which are not synthesizable....	783
W182h : Reports tri1 net declarations that are not synthesizable.....	785
W182k : Reports trireg declarations that are not synthesizable	787
W182n : Reports MOS switches, such as cmos, pmos, and nmos, that are not synthesizable	789
W187 : The 'default' or 'others' clause should be the last clause in a case statement	791
W188 : Do not write to input ports	793
W189 : Nested Synopsys translate_off comments	794
W19 : Reports the truncation of extra bits	796
W190 : Task or procedure declared but not used	800
W191 : Function declared but not used	801
W192 : Empty block	802
W193 : Empty statement	803
W208 : Nested Synopsys translate_on comments.....	804
W210 : Number of connections made to an instance does not match number of ports on master	805

W213	: Reports PLI tasks or functions that are not synthesizable	808
W215	: Reports inappropriate bit-selects of integer or time variables ...	810
W216	: Reports inappropriate range select for integer or time variable.	813
W218	: Reports multi-bit signals used in sensitivity list	816
W224	: Multi-bit expression found when one-bit expression expected ..	818
W226	: Case select expression is constant	820
W238	: Mixing combinational and sequential styles	823
W239	: Reports hierarchical references that are not synthesizable.....	825
W240	: An input has been declared but is not read	827
W241	: Output is never set	831
W243	: Recursive task enable	835
W245	: Probably intended "or", not " " or " " in sensitivity list	837
W250	: Reports disable statements that are not synthesizable	839
W253	: Data event has an edge.....	842
W254	: Reference event does not have an edge.....	843
W256	: A notifier must be a one-bit register	844
W257	: Synthesis tools ignore delays	845
W259	: Signal has multiple drivers.....	846
W263	: Reports a case expression width that does not match case select expression width	848
W280	: A delay has been specified in a nonblocking assignment.....	855
W287a	: Some inputs to instance are not driven or unconnected.....	856
W287b	: Output port to an instance is not connected	859
W287c	: Inout port of an instance is not connected or connected net is hanging.....	861
W289	: Reports real operands that are used in logical comparisons	863
W292	: Reports the comparison of real operands	865
W293	: Reports functions that return real values.....	868
W294	: Reports real variables that are unsynthesizable.....	871
W295	: Reports event variables that are not synthesizable	873
W306	: Converting integer to real	876
W307	: Converting unsigned (reg type) to real	877
W308	: Converting real to integer	878
W309	: Converting unsigned (reg type) to integer.....	879
W310	: Converting integer to unsigned (reg type).....	880
W311	: Converting real to unsigned (reg type)	882
W312	: Converting real to single bit	883
W313	: Converting integer to single bit.....	884

W314 :	Converting multi-bit reg type to single bit	885
W316 :	Reports extension of extra bits in integer conversion	886
W317 :	Reports assignment to a supply net.....	889
W323 :	Multiply driven inout net	891
W326 :	Event variable appearing in a posedge/negedge expression	892
W332 :	Not all cases are covered - default clause will be used.....	893
W333 :	Unused UDP	894
W336 :	Blocking assignment should not be used in a sequential block (may lead to shoot through)	895
W337 :	Reports illegal case construct labels.....	899
W339a :	Case equal operator (==) and case not equal (!=) operators may not be synthesizable.....	902
W34 :	Macro defined but never used.....	905
W342 :	Reports constant assignments that are X-extended	906
W343 :	Reports constant assignments that are Z-extended	908
W345 :	Presence of an event control in a task or procedure body may not be synthesizable	910
W346 :	Task may be unsynthesizable because it contains multiple event controls	912
W348 :	Unspecified width for integer expression in a concatenation	913
W350 :	A control character appears inside a string	915
W351 :	A control character appears inside a comment	916
W352 :	Reports "for" constructs with condition expression.....	917
W362 :	Reports an arithmetic comparison operator with unequal length	921
W372 :	A PLI function (\$something) not recognized.....	929
W373 :	A PLI task (\$something) is used but not recognized	930
W391 :	Reports modules driven by both edges of a clock	931
W392 :	Reports reset or set signals used with both positive and negative polarities within the same design unit	935
W395 :	Multiple asynchronous resets or sets in a process or always may not be synthesizable	942
W396 :	A process statement has clock signal, but no asynchronous reset signal	945
W397 :	Destination of an assignment is an IN port	947
W398 :	Reports a case choice when it is covered more than once in a case statement	948
W401 :	Clock signal is not an input to the design unit	953
W402 :	If you internally generate reset signals, do so in a single module	

instantiated at the top-level of the design	957
W402a : Synchronous reset signal is not an input to the module.....	959
W402b : Asynchronous set/reset signal is not an input to the module..	961
W414 : Reports nonblocking assignment in a combinational block	963
W415 : Reports variable/signals that do not infer a tristate and have multiple simultaneous drivers	967
W415a : Signal may be multiply assigned (beside initialization) in the same scope	971
W416 : Width of return type and return value of a function should be same (Verilog). Reports functions in which the range of the return type and return value of a function are not same (VHDL)	982
W421 : Reports "always" or "process" constructs that do not have an event control	988
W422 : Unsynthesizable block or process: event control has more than one clock	991
W422L : Multiple clocks in the event control list of latch is not allowed.	995
W423 : A port with a range is redeclared with a different range	996
W424 : Ensure that a function or a sub-program does not sets a global signal/variable.....	998
W425 : Ensure that a function or a sub-program does not uses a global signal/variable.....	1002
W426 : Ensure that the task does not sets a global variable	1006
W427 : Ensure that a task does not uses a global variable	1009
W428 : Ensure that a task is not called inside a combinational block..	1012
W429 : Task called in a sequential block	1015
W43 : Reports unsynthesizable wait statements.....	1017
W430 : The "initial" statement is not synthesizable	1020
W433 : More than one top-level design unit	1022
W438 : Ensure that a tristate is not used below top-level of design....	1023
W442a : Ensure that for unsynthesizable reset sequence, first statement in the block must be an if statement.....	1026
W442b : Ensure that for unsynthesizable reset sequence, reset condition is not too complex	1029
W442c : Ensure that the unsynthesizable reset sequence are modified only by ! or ~ in the if condition	1032
W442f : Ensure that the unsynthesizable reset sequence is compared using only == and != binary operator in the if condition.....	1035
W443 : 'X' value used	1038
W444 : 'Z' or '?' value used.....	1041

W448 :	Reset/set is used both synchronously and asynchronously	1044
W453 :	Large case constructs should not be used.....	1049
W456a :	A signal is included in the sensitivity list of a combinational always block but none of its bits are read in that block (Verilog) A signal is included in the sensitivity list of a combinational process block but none of its bits are read in that block (VHDL).	1051
W464 :	Ensure that the unrecognized synthesis directive is not used in the design	1055
W467 :	Use of don't-care except in case labels may lead to simulation/synthesis mismatch.....	1060
W468 :	Index variable is too short	1063
W474 :	Variable assigned but not deassigned	1064
W475 :	Variable deassigned but not assigned	1066
W476 :	Variable forced but not released.....	1067
W477 :	Variable released but not forced.....	1069
W479 :	Checks if loop step statement variables are not properly incremented or decremented	1070
W480 :	Ensure that the loop index is of integer type.....	1072
W481a :	Ensure that a for loop uses the same step variable as used in the condition	1075
W481b :	Ensure that a for loop uses the same initialization variable as used in the condition.....	1078
W484 :	Possible loss of carry or borrow due to addition or subtraction	1081
W486 :	Reports shift overflow operations	1086
W489 :	The last statement in a function does not assign to the function (Verilog) The last statement in a function does not assign to the function (VHDL)	1091
W490 :	A control expression/sub-expression is a constant	1093
W491 :	Reports case expression with a width greater than the specified value.....	1096
W493 :	A variable is not declared in the local scope, that is, it assumes global scope.....	1098
W494 :	Inout port is not used	1100
W495 :	Inout port is never set	1102
W496a :	Reports comparison to a tristate in a condition expression ...	1104
W496b :	Reports comparison to a tristate in a case statement	1107
W497 :	Not all bits of a bus are set	1110

W498 : Not all bits of a bus are read	1112
W499 : Ensure that all bits of a function are set.....	1114
W501 : A connection to a reset port should not be a static name.....	1116
W502 : Ensure that a variable in the sensitivity list is not modified inside the always block	1118
W503 : An event variable is never triggered	1121
W504 : Integer is used in port expression	1122
W505 : Ensure that the signals or variables have consistent value.....	1124
W527 : Dangling else in sequence of if conditions. Make sure nesting is correct	1129
W528 : A signal or variable is set but never read	1131
W546 : Duplicate design unit.....	1137
W551 : Ensure that a case statement marked full_case or a priority/unique case statement does not have a default clause.	1138
W552 : Different bits of a bus are driven in different sequential blocks	1141
W553 : Different bits of a bus are driven in different combinational blocks. 1143	
W561 : A zero-width-based number may be evaluated as 32-bit number .. 1144	
W563 : Reduction of a single-bit expression is redundant.....	1145
W575 : Logical NOT operating on a vector.....	1146
W576 : Logical operation on a vector.....	1147
W66 : Ensure that a repeat construct has a static control expression .	1149
W69 : Ensure that a case statement specifies all possible cases and has a default clause.....	1152
W701 : Included file is not used.....	1156
W71 : Ensure that a case statement or a selected signal assignment has a default or OTHERS clause	1157
W86 : Not all elements of an array are set	1163
W88 : All elements of a memory are not set	1165
WhileInSubProg : Reports unsynthesizable While statements used inside subprograms.....	1167
W541 : A tristate is inferred	1170
LINT_MULTIASSIGN_BLOCKING_SIG : Signal may be multiply assigned in blocking manner (beside initialization) in the same scope. 1171	
LINT_MULTIASSIGN_NONBLOCKING_SIG : Signal may be multiply assigned in non-blocking manner (beside initialization) in the same scope.....	1173

Miscellaneous Tags.....	1175
ConstSig : Signal has a constant value or can only switch to a constant value.....	1176
DeadCode : Code does not contribute to functionality of the design ...	1182
Mux01 : Connected MUXes should not have a common select.	1185
Reset_check11 : Reports asynchronous resets that are used as both active-high and active-low	1186
MoreLint Tags.....	1192
AlwaysCombExhaustive-ML : The conditions of an if/unique-if/priority-if construct inside an always_comb block should be exhaustive. ...	1193
AlwaysFalseTrueCond-ML : Checks the use of always-false and always-true conditions	1194
AMSKeyword-ML : Verilog-AMS keyword found.....	1198
AnsiPort-ML : The module declaration should use Verilog ANSI style ports only	1200
ArrayPortConnect-ML : Detect port connections used in multidimensional signals with non-static indexes	1203
AsgnNextSt-ML : Next state variable is not assigned in the default state of Finite-state machine (FSM)	1206
AsgnToOneBit-ML : Assigning a 0 or 1 (32-bits) to a 1 bit is not allowed..	1207
AsgnOverflow-ML : Detects assignment overflow.....	1209
AssignPatInInst-ML : Detect assignment patterns in parameter override or port connection area.....	1213
AutomaticFuncOnly-ML : Detects non-automatic functions	1216
AutomaticFuncTask-ML : Use automatic functions/tasks in modules and interfaces	1220
AvoidShortedPort-ML : Avoid connecting one net to different logical inputs of a same instance, except GND, VDD nets.....	1222
BitDataType-ML : Bit data type signal has been used in the design...	1223
BitRangeUsedParam-ML : Bit Range Specified for Parameter.....	1224
BitOrder-ML : Bit order specification should follow recommended convention.	1228
BreakInMultiDimForeach-ML : Reports if break or continue statement is used in a multi-dimensional foreach loop.....	1230
CAPA-ML : Copy-paste should not be done within a design file.....	1234
CheckAnalogPin-ML : Analog output pins should be connected to analog input pins only.....	1236

CGFromSpecifiedCell-ML : Clock gated only with user specified cells	1237
CGFromSpecifiedLib-ML : Clock gated only with a cell from user specified library	1239
CheckAsyncHold-ML : Report violations for flop with an asynchronous hold.	1241
CheckAlwaysCombSenseList-ML : Report violations for signal which are read in side an always comb block and not in the implicit sense list of that always_comb block.	1244
CheckLocalParam-ML : Reports violation if parameters are used in modules or interfaces	1246
CheckImplicitPort-ML : Reports if instances are using implicit port connections in the design	1248
CheckModulesWithoutPorts-ML : Reports violation for modules without any port.	1256
CheckPortType-ML : Complex port types are not allowed in module interface	1259
CheckRegAndWire-ML : All reg / wire uses logic, but excluding multi-drive signals	1262
CheckShiftOperator-ML : Logical and arithmetic shift operation detection on signed and unsigned operand(s) or expression(s) respectively	1266
CheckSignedFunc-ML : Most significant bit should not be concatenated with 0 or 1 in signed function.	1270
CheckTimeUnitandPrecision-ML : Delay used without timeunit and timeprecision in module	1273
ChkCarriageReturn-ML : Carriage return should be last line of a file	1274
CheckAssignToVecBits-ML : Use explicit assignments only for bit-selects, part-selects and full vectors.	1275
CheckDelayTimescale-ML : Delay used without timescale compiler directive	1276
CheckParamSensList-ML : Always block uses parameters in the sensitivity list	1278
CheckSyncReset-ML : Synthesis directive sync_set_reset must be set for all synchronous reset description.	1279
CheckSynthPragma-ML : Pragma option must be set on the command line if translate_on/off blocks are present in a testcase.	1280
ChkSensExprPar-ML : Expression uses parenthesis '()' in the sensitivity list	1281
CheckSwappedIndex-ML : Do not use bus signals with swapped indexes	

while instantiation.....	1283
ChkUndefMacro-ML : Macro is not defined before undefining it.	1287
ConstBitCase-ML : Detect constant bits in case expression and case items 1288	
ConstDrivenNet-ML : All internal net of module should not be assigned a constant value.....	1292
ConstWithoutValue-ML : Avoid constants with unspecified value (deferred constants).	1293
CloseCaseWithX-ML : Others clause does not have X assignment to destination signals	1294
ComplexExpr-ML : Complex Expression should be avoided.	1297
ConcatenationInArrayAssign-ML : Use assignment patterns in unpacked array assignments	1298
CondSigAsgnDelay-ML : Delays should not be used in conditional signal assignment	1303
ConflictVar-ML : Flag where conflicting variable names across different processes are used.....	1304
CoveragePragma-ML : The coverage off pragma should have corresponding the coverage on pragma.....	1305
DetectParamTruncate-ML : Parameter value is truncated	1306
DefineTypeDefStructure-ML : Always define a typedef when using a structure or union	1312
DetectBlackBoxes-ML : Black box instance detected.....	1315
DiffDelayInNonBlock-ML : Checks if all the non-blocking assignments have delays and those delays are equal	1316
DisallowForceOnEquiNet-ML : Reports equivalent hierarchical signals due to which multiple drivers happen on same signal.....	1319
DisallowForceOnPortConn-ML : Reports simulation synthesis mismatch due to force assignment.....	1321
DisallowSVAwaysLatch-ML : Reports the usage of SystemVerilog always_latch	1323
DiffTimescaleUsed-ML : Design should not have different timescales for different modules.....	1325
DirectiveCheck-ML : Dumps information about ifdef, undef, and include directives used in design.....	1328
DisallowCaseX-ML : Do not use casex constructs in the design.....	1334
DisallowCaseZ-ML : Do not use casez constructs in the design	1337
DisallowDWComp-ML : Do not instantiate DesignWare component used for multiplication, division	1340

DisallowMult-ML : Multiplication operator (*) should not be used in the design.....	1341
DisallowTimeArr-ML : Array of time is not allowed	1344
DisallowVal-ML : Flag where the values 'U', '-', 'W', 'H' and 'L' are used... 1345	
DisallowXInCaseZ-ML : Do not use "x" in casez statements.....	1346
DuplicateCase-ML : The conditions of a case/unique-case/priority-case construct should not be duplicated.	1348
DuplicateCaseLabel-ML : Ensure that a case construct does not have duplicate case clauses with the parallel_case pragma specified . 1349	
DuplicatePort-ML : Do not define duplicated ports	1353
EnableXPropagation-ML : Use "?:" construct instead of "if/else" construct to enable X-propagation	1355
EntityCompMismatch-ML : Inconsistent case between entity/module name vs component name	1358
EnumBaseComparison-ML : Ensures that two variables of different enumerated data types are not compared	1359
EnumDeclInGenerate-ML : Report all enumeration declaration inside generate (for, case, if), nested generate	1363
EnumOutOfRangeCast-ML : Reports if a cast occurred into an enumerated type with a value which does not lie inside its range.	1366
EnumStateDecl-ML : The Finite-state machine's (FSM) enum states parameter declaration should appear before the reg declaration 1372	
EventControlInRHS-ML : RHS of assignment statement has a event control	1373
ExplicitSignExtend-ML : Reports explicit sign extension	1374
ExplicitTwosCompliment-ML : Report explicit two's compliment.....	1379
FindStringsInComment-ML : Required string present in commented lines 1382	
FSMNextStateName-ML : State name prefix or suffix	1383
FSMNonConstDefault-ML : Checks whether the next state of FSM is specified as a constant expression in the default branch....	1386
FsmUsage-ML : Finite State machine used	1390
FuncTaskUsedBeforeDefine-ML : Identifies the functions and tasks that have been used before the definition	1393
FuncFileName-ML : FileName should be '<functionname>.v' or '<functionname>.sv' for a file which contains only function	

definition(s)	1395
FlopFeedbackRace-ML : Reports potential race conditions between flip-flop output and its clock/reset pin	1396
GenIndexNonInt-ML : Avoid use of non-integer indexes inside GENERATE constructs	1398
GenvarUsage-ML : Genvar variable has been defined but not used...	1399
GenvarDeclaration-ML : For-Loop variable defined by using genvar must be declared outside of generate block.	1401
GlobalDataTypeDefined-ML : Data type objects (user-defined type declarations and definitions of identifiers) should not be present or visible in global Scope.....	1404
GroupOfAsgn-ML : Verilog treats differently C-like groups of assignments	1409
HierarchyReport-ML : Tag will generate a report with hierarchies of usage of global Verilog entities.....	1410
IfOverlap-ML : The conditions of an if-construct (including unique-if and priority-if) should not overlap (Verilog) The conditions of an if-construct should not overlap (VHDL).....	1413
IfWithoutElse-ML : An "if" clause does not have a corresponding "else" clause and is not preceded by assignment of target signal .	1416
ImplicitParamImports-ML : Report re-definition of a localparam implicitly imported from a package	1421
ImplicitAlways-ML : Reports violations for implicit always blocks (always@*) that will never trigger during simulation.....	1426
ImproperRangeIndex-ML : Possible discrepancy in the range index or slice of an array.....	1430
InActiveReset-ML : Do not use inactive sets/resets on flip-flops	1437
IncludeFileForEachModule-ML : Checks if include file is explicitly included in each child module.....	1439
InlineComment-ML : Detect absence of space before/after an inline comment	1441
InstNameSameAsMaster-ML : Do not use an instance name identical to involved master name	1442
InstNameStyle-ML : Preferably name instances as per <instance_prefix><component_name><instance_suffix> .	1444
InterfaceNameConflicts-ML : Interface Signal Name conflicts with other signal name	1447
InterfaceWithoutModport-ML : Interfaces should make use of modport	

declarations.	1449
IntRange-ML : It is necessary to specify a range for integer/real signal... 1452	
InstPortConnType-ML : Output and inout port connection in instantiation should not be of type, reg.	1454
InvalidNetType-ML : Invalid net type detected.....	1459
InvalidRepeatCondition-ML : X or Z values are not allowed in repeat condition	1461
InvalidTypedefName-ML : Invalid typedef name detected.....	1462
LiteralOverflow-ML : Check if the given bit string literal is of a length longer than specified by the width specification.....	1465
InvalidVariedTypedefNameStyle-ML : Reports type names defined by typedefs of struct, union, enum, or other that do not follow a specified or default naming style pattern.....	1468
LogicEnumBase-ML : Always declare enum base type explicitly as sized logic type	1473
LongHierName-ML : Long hierarchical name inferred for the construct 1475	
MacroWithoutUndef-ML : Macro is not undefined	1479
MacroFileName-ML : FileName should be '<macroname>.v' or '<macroname>.sv' for a file, which contains only macro definition(s)	1483
MissingPortInModport-ML : All the ports of interface should be used in modport declarations	1484
MissingPinFunctionCall-ML : Use input pin for all read signals, which are not declared within function scope	1486
MixedResetEdges-ML : Do not use same asynchronous reset with different polarities inside same process block	1491
MaxDelay-ML : Delay is more than the threshold value	1494
MaxFaninLogic-ML : Report a violation on all flops that have drivers greater than ALLOWED_DRIVER_LIMIT	1506
ModuleName-ML : Module name does not follow the naming convention.. 1507	
MultiModuleInCellDefine-ML : There are more than one module definitions in between a single `celldefine and `endcelldefine ... 1508	
MultiOpInModule-ML : Circuit of multiple operators are generated in the same module	1509
MultOperVar-ML : Avoid multiple operations on variable in the same	

expression	1511
MultipleFilesCellDefine-ML : Scope of `celldefine spans across multiple files.	1515
NegativeIndex-ML : Do not use negative or enum array index	1516
NestedInterfaceInst-ML : Detect interface instances in interfaces...	1521
NeverExecutingForLoop-ML : Zero loop count.....	1525
NoArithOp-ML : Arithmetic Operators should be avoided.	1527
NoArithShiftInConcat-ML : Tag will report violation for arithmetic operations inside concatenation.	1529
NoArray-ML : Detects usage of arrays in design	1536
NoBitArray-ML : Do not use one-pin buses.....	1537
NoConstSourceInAlways-ML : Checks the use of constant/inferred constants in RHS always block	1538
NoFuncOrProc-ML : Avoid use of functions and procedures	1540
NoGenericMap-ML : Instantiation must have generic map for a generic declared in the entity/component.....	1541
NoOpen-ML : Flag where open keyword is used	1543
NoVerilogPrims-ML : Verilog primitives are prohibited.....	1544
NonBlockingCounters-ML : Non-blocking counter assignment found	1545
NonConstShift-ML : Checks whether there is any non-constant shift expression (Right operand).	1547
NoFeedThru-ML : Do not use feedthrough signals	1554
NonLocalLoopIndex-ML : Catches loop indexes that are not locally defined	1556
NonOverridenParams-ML : Tag reports violation for non-overridden parameters	1558
NonSynthRepeat-ML : Repeat statements are non-synthesizable.....	1562
NonVoidFunction-ML : Checks if the function is explicitly casted to void .	1563
OneLineComm-ML : Use one-line comments	1567
OperShortCircuit-ML : Checks if operator short circuiting occurs across the function call	1568
ParameterCheck-ML : Do not use parameters.....	1572
ParamValueOverride-ML : Detect overridden parameters in RTL design ..	1574
PortConnectToFixedVal-ML : Port of a module connected to a fixed value.	1576
PortNameSameAsModule-ML : Do not use a port name identical to the unit name	1579

PortRange-ML : Top level output port is driven by a constant.	1580
PortUsedBeforeDefine-ML : Port used before defining	1581
PortWithoutDirection-ML : Port is defined without direction	1583
PortWithoutType-ML : Port is defined without type.....	1586
PortIOConnectionSanity-ML : Checks if primary input/output ports are connected to a valid input/output PAD cell pin	1588
ProhibitedDataTypes-ML : Checks for prohibited data types being used directly.....	1591
RealValueUsed-ML : Checks if real literals are used	1599
RedundantLogicalOp-ML : Logical operation result is same as one of the operands or is a constant.....	1603
RedundantLogicalOpInExpr-ML : Redundant expressions in logical expressions.....	1606
RepeatedLogic-ML : Report cases when arithmetic or bitwise expressions inside 'conditions in loops' or 'condition bodies' can be placed outside.....	1611
ReportCompOperator-ML : Operator <= or >= used in binary expressions with std_logic, std_ulogic, , std_logic_vector or std_ulogic_vector type operands.....	1618
RegisterInfo-ML : Generate a report which includes all registers in RTL design	1626
ResetPortConnectedToSupply-ML : Do not have reset connected to static value in instantiation.....	1632
ScopedVarUsedBeforeDefine-ML : Reports scoped variables that are used before defining.....	1635
SepTFMacro-ML : Task/functions/macros should be defined in separate file 1637	
SimSynthMismatchInVarInit-ML : Simulation - synthesis mismatch in variable initialization	1638
SigAsgnDelay-ML : Delays should not be used in Signal assignment.	1641
SigAssignX-ML : Signal driven by X.....	1643
SigAssignZ-ML : Signal driven by Z.....	1647
SignalModifiedAfterRead-ML : Report, if signal is set after it is read in the always_comb block.	1651
SignedPackedMDA-ML : Reports if a signed packed multi-dimensional array is used	1660
SimSynthMismatchInVarInit-ML : Simulation - synthesis mismatch in variable initialization	1665
SingleEntInFile-ML : There should be only one entity or one entity and its	

architecture in a file	1671
SpareCaseExprBit-ML : Detect redundant bits in case expression	1673
SpecifyBlockUsed-ML : Specify blocks are not supported.....	1676
SyncRstFirstUse-ML : Synchronous reset should be used in first if condition of always block	1677
SynchValueUsed-ML : Synchronized value should not be used asynchronously.....	1679
SynthElabDuName-ML : Synthesized module name is longer than required limit.	1680
SVConstruct-ML : SystemVerilog construct used	1681
TwoStateData-ML : Avoid using two state data type signals in RTL...	1682
TypedefNameConflict-ML : Ensures that variable name are not same as that of the System Verilog type-defined name	1685
UncoveredEnumCase-ML : All enum items should be defined in case statement	1687
UniqueIfMissingCond-ML : Reports violation if priority/unique type if/case statement is used but all the conditions are not covered....	1693
UnInitParam-ML : A parameterized module/entity has been instantiated but parameters/generics are not explicitly specified.	1695
UnInitTopDuParam-ML : Top level module/entity has parameters/ generics but the parameters/generics are not explicitly specified on the command line.....	1696
UnifiedNetList-ML : Preferably have "UNIQUEFIED" netlist.....	1697
UninitializedReset-ML : Ensure that there are no uninitialized signals in reset block of a flip-flop.	1699
UniqueCase-ML : Unique, Unique0, or priority case constructs used with full_case or parallel_case or both.	1702
UniquePriorityMisuse-ML : Detects if unique, unique0, and priority keywords are misused in case constructs	1705
UnsuppCompDir-ML : `ifdef, `resetall, `default_nettype and `unconnected_drive are not supported by all tools	1707
UnusedFunctionInput-ML : Function input has been declared but it is not read	1708
UnpackedStructUsed-ML : Do not declare unpacked structure outside the package.....	1710
UseBusWidth-ML : Bits of a bus should be specified when a bus is used as a whole	1714
UseImplicitAlways-ML : Sensitivity list must be implicit in combinational always blocks	1717

UseLogic-ML : Checks for reg and wire usage in the design	1722
UseSVCasting-ML : Checks if System Verilog casting constructs are used for assigning a packed bit vector to a packed structure	1724
VarInCaseInside-ML : Variable or function call in case-inside item...	1727
V2KConstruct-ML : Verilog 2001 construct used	1732
ValueSizeOverFlow-ML : Input value should not be greater than size specified in IEEE conversion functions.....	1733
WrapInstance-ML : Black box cell should not be contained on its own in a wrapper module	1736
ZeroReplicationMultiplier-ML : Reports replications with a zero replication multiplier in Verilog	1737
MemConflict-ML : Possible memory conflict (overwrite) detected	1738
MultiAssign-ML : Signal or Variable may be multiply assigned (beside initialization) in the same process scope.	1740
NestedCaseStmt-ML : Nesting depth of case construct exceeds specified depth.	1742
NestedCellDefine-ML : There is a `celldefine in the source before a `endcelldefine to close the previous `celldefine	1743
NoAssignX-ML : Ensure RHS of the assignment does not contains 'X'	1744
NoDisableInFunc-ML : Disable statements in functions may not be synthesizable	1748
NoDisableInTask-ML : Disable statements in tasks may not be synthesizable	1749
NoExprInPort-ML : Port connections in instances should not contain expressions.....	1750
NoGenLabel-ML : Unnamed generate block or duplicate generate block names used in module.....	1752
NoInoutPort-ML : Do not use bi-directional ports.....	1757
NonConstReset-ML : Signals must be reset to constant values.	1759
NonStaticMacro-ML : Macro evaluates to non-static value	1764
NonWireSignal-ML : The signal used in a design should preferably be a wire net.....	1765
NoOthersInAsgn-ML : Flag other constructs in assignment for slice of a vector	1766
NoParamMultConcat-ML : Do not use parameter as left operand in multiple concatenation expression.....	1767
NoRealFunc-ML : Functions of type real are not synthesizable	1769
NoSigCaseX-ML : Do not use signals in casex and casez constructs ..	1770
NoStrengthInput-ML : Synthesis tools ignore strength inputs	1772

NoWidthInBasedNum-ML : Width should be specified for all based numbers.....	1773
NoXInCase-ML : Ensure that case expression and case choices do not have 'X'	1775
NullOthers-ML : Others clause of a case statement should not have null statement	1777
NullPort-ML : Null Ports should not be defined	1778
OneModule-ML : At most one module per file	1779
ParamOverrideMismatch-ML : Mismatch in the number of parameter over-rides and number of parameters in the instantiated module	1780
ParamWidthMismatch-ML : Parameter width does not match with the value assigned.....	1782
PartConnPort-ML : Port is unconnected or partially connected	1784
PartSelectRange-ML : Vector part select is probably wrongly written, use '+:' instead of ':'	1786
PragmaComments-ML : Detects Pragma Comments	1788
RegInput-ML : Module input port should be registered	1791
ReportPortInfo-ML : Generate a report of all ports of top level block and black box instances	1792
ReserveNameSystemVerilog-ML : SystemVerilog IEEE-1800 reserved word used.....	1796
ReserveNameV2K-ML : Verilog 2001 reserved word used	1799
SameDu-ML : Same design unit in Verilog and VHDL.....	1801
SameLabelsInGenerate-ML : Flag violation when same labels are used inside different generate block	1802
SameLoopIndexUsed-ML : Do not use same for-loop index variable inside different always @* blocks.	1804
SelfAssignment-ML : Same operand on both sides of combinational assignment detected	1806
SelfDeterminedExpr-ML : Self determined expression present in the design.	1808
SensListRepeat-ML : A signal is repeated multiple times in the sensitivity list	1812
SetBeforeRead-ML : Variable may not be set before being read (VHDL) Signal is read before being set (Verilog).....	1814
SetResetConverge-ML : Fan-in cone of Set and Reset pin of a flip-flop/ latch converge.....	1819
SignedUnsignedExpr-ML : Do not mix signed & unsigned variables/	

constants in expressions	1823
SynchReset-ML : Do not use synchronous reset in the design	1827
TaskFileName-ML : FileName should be '<taskname>.v' or '<taskname>.sv' for a file, which contains only task definition(s) 1828	
UnConstrLoop-ML : Number of iterations in <for while> -loop exceeds max limit (Verilog) Unconstrained loops should not be used (VHDL)	1829
UnrecSynthDir-ML : Synthesis directive is not recognized	1832
UseSVAIways-ML : Checks the use of System Verilog constructs for modeling of procedural combinatorial and sequential logic.	1834
WrapInstance-ML : Black box cell should not be contained on its own in a wrapper module	1837
ConstantInput-ML : Input to a particular module is tied to a constant value 1839	
HangingInstInput-ML : Net connected to input port of instance is undriven 1841	
HangingInst-ML : Block is not connected	1844
HangingInstOutput-ML : Net connected to output port of instance is unconnected	1845
NoBusPartClock-ML : Bus bits or slices should not be used as clocks	1846
NoFeedThrus-ML : Block should not contain feed-throughs	1848
ReEntrantOutput-ML : The re-entrant outputs should be avoided	1850
RegInputOutput-ML : Register the module output and input ports..	1851
ResetFlop-ML : All the flip-flops should have either asynchronous set/reset or synchronous set/reset	1856
RptNegEdgeFF-ML : Negative edged flip-flops should not be used in the design.	1859
SameControlDataNet-ML : Control (set/reset) signal used as data for the same flip-flop.	1862
TristatePort-ML : Tristate port detected	1863
TristateSig-ML : Tristate signal detected	1865
Undriven-ML : Undriven nets/terminals detected in the design	1867
UndrivenInTerm-ML : Detects undriven but loaded input terminal of an instance	1868
UndrivenNet-ML : Undriven but loaded net is detected in the design	1878
UndrivenNUnloaded-ML : Undriven and Unloaded nets/terminals detected in the design	1881
UndrivenOutPort-ML : Undriven but loaded output port of a module	

detected	1885
UndrivenOutTermNLoaded-ML : Undriven output pins connected to instance input.....	1887
UniqueInputOutputSampling-ML : Checks the existence of unique output flip-flops to output interface and unique input flip-flops to input interface	1889
UnloadedInPort-ML : Unloaded but driven input port of a module detected	1892
UnloadedNet-ML : Unloaded but driven net detected in the design ...	1898
UnloadedOutTerm-ML : Unloaded but driven output terminal of an instance detected.....	1901
CheckKeywordsOfCaseStmt-ML : tag will check the usage of unique/unique0/priority keyword with case, casez, casex and case inside statements.....	1904
ComplexModport-ML : Detects complex modport declarations	1908
ConcatUnsizeNumbers-ML : tag will report violations for concatenation of unsize numbers.....	1912
DetectNonBlockGenFor-ML : For-loop inside generate statements must have a begin/end block.....	1915
DirectTopInputToInout-ML : Checks the connection between top input and inout ports.....	1918
ShiftRegister-ML : Reports case coding style shift register	1921
UseParamWidthInOverriding-ML : Parameters should not be overridden without specifying the width in instantiations	1933
IncompatibleSVAssign-ML : SV assign operators should not be used in expressions.....	1938
IncrementalForLoop-ML : Tag will report violations for usage of ++ and - operators in the for loop.	1947
DisallowUnknownInCase-ML : Tag will check the usage of 'x', 'z' or '?' in all case statements.	1951
DivideByConstant-ML : Unverifiable formality issue due to division of variable signal by constant.	1954
NoOutputPort-ML : Module with no output.....	1958
FuncWithIntReturn-ML : Function with integer return value	1960
InoutPortConnectionSanity-ML : Checks if inout port is connected to right type of IO cell pin	1962
InstanceNameRequired-ML : Instance without name	1964
HangingFlopOutput-ML : Report unused flip-flops in the design	1965
LoopVarFreeAssign-ML : Loop variable independent assignments need	

optimization.....	1966
LoopVarInAlways-ML : For-Loop variable defined by using integer must be declared outside of always block.....	1970
NonReusableParametricModule-ML : This tag categorizes parameters based on the usage of the parameter	1976
PackedArrayInRange-ML : Packed arrays should not be used as range values in a range.	1979
SameAssignment-ML : Report same assignment expression in different branches of a conditional block.	1986
SupplyNet-ML : Signal is defined as Supply0/1.	1991
SV09Construct-ML : System Verilog 2009 construct used	1994
TimescaleUsed-ML : Timescale usage found in the design.....	1995
PortTypeMismatch-ML : Port type mismatch between module declaration and net connection in the instantiation.	1997
ConcatInUnpacked-ML : Unpacked array concatenation must be avoided 2001	
MaxFanoutCount-ML : High fan-out detected for internal node.....	2002
PortConnectedToMux-ML : MUX detected in fan-in cone of specified pin. 2003	
UnusedFlopOutput-ML : Unreachable sequential element must be removed from the design.....	2006
ResetPreventSRL-ML : Reset used in shift register will prevent SRL inference	2010
LINT_DETECT_EMPTY_MODULE : Reports violations for empty modules in RTL design.	2011
LINT_IMPROPER_RANGE_INDEX : Possible discrepancy in the range index or slice of an array	2015
LINT_INSTANCE_WITHOUT_NAME : Instance without name	2020
LINT_LOOP_THRU_FLOP_WITHOUT_SETRESET : Avoid synchronous loops that contain flip-flops without any set/reset.....	2022
LINT_LOGICAL_EXPR_USED_IN_SENS_LIST : Logic expression used in sensitivity list.....	2026
LINT_ALL_BITS_SHIFT_OUT : All bits shifted out	2028
LINT_MACRO_MODULE_USED : Macromodules are not synthesizable 2032	
LINT_DETECT_UNSYNTHESIZABLE_NET : Net type not synthesizable... 2033	
LINT_SV_STRING_USED_IN_DESIGN : Avoid using SystemVerilog string in the design	2034

LINT_SNPS_DIRECTIVE_USED : Synopsys directive translate_off/ translate_on used	2036
LINT_FSM_WITHOUT_EXIT_STATE : Do not have states without exit in FSMs.....	2038
LINT_FSM_UNREACHABLE_STATES : Do not have unreachable states in FSMs.....	2041
LINT_FSM_WITHOUT_INITIAL_STATE : Do not have FSM without the initial state.....	2044
LINT_GTECH_CELL_USED : GTECH cell is used	2048
LINT_SUB_MODULE_NAMING_STYLE : Checks the naming convention of sub-modules.....	2049
LINT_INVALID_NET_TYPE_IN_MODULE_INSTANTIATION : REG output port is assigned to a WIRE port connection	2051
LINT_UNSPECIFIED_MACRO_NAME : Macro name is unspecified in `ifdef, `ifndef, `elsif usage.....	2055
LINT_UNDEFINED_MACRO_USED : Macro is defined after it is used	2058
LINT_INVALID_NET_TYPE_IN_DU_INSTANTIATION : REG output port is assigned to a WIRE port connection	2061
LINT_CLKGEN_AVOID_CONTASSIGN : Use of continuous assign statement in clock generator module	2065
LINT_CLKGEN_AVOID_SEQSTMT : Use of sequential process statement in clock generator module	2067
LINT_RSTGEN_AVOID_SEQSTMT : Use of sequential process statement in reset generator module	2069
LINT_NONSTANDARD_LIB_USED : Report direct use of library cell in the RTL.....	2071
OpenMore Tags	2073
ActLowName : Active-low signal name does not follow the naming convention	2074
AlwaysComment : Always block is not commented.....	2078
ConstantComment : Constant declaration does not have a comment	2080
ArrayIndex : Bus signals are declared with low-order bit first.....	2081
CaseOverIf : Use case statements instead of if/else, where feasible, if performance is important. (Verilog) Use case statements instead of if/else, where feasible, if performance is important. (VHDL)	2088
ConsCase : Name not used consistently with same case.....	2092
ConstName : Constant does not follow recommended naming convention . 2096	

DefaultState :	Finite-state machine does not have a defined default state 2101
ExprParen :	Parenthesize complex expressions to force evaluation precedence in the order you intend. 2106
FileHdr :	Standard header not found at the top of source file 2110
FuncName :	Function/procedure does not follow recommended naming convention..... 2113
FunctionComment :	Function is not commented 2117
HardConst :	Do not use hard-coded constants other than 0 or 1 2118
Indent :	Code is not correctly indented..... 2121
InstName :	Instance name does not follow the naming convention ... 2125
InferFF :	Flip-flop inferred 2128
InstNameLength :	Instance name length does not follow the convention . 2130
InvSigType :	Signal type is type bit or bit_vector 2132
LineLength :	Line-length exceeds defined limit..... 2134
NamedAssoc :	Use named-association rather than positional association to connect to an instance..... 2136
NoDup :	Signal, variable, entity or other named object names are duplicated..... 2139
NotInSens :	A signal is read inside a combinational always block but is not included in the sensitivity list (Verilog) A signal is read inside a combinational process block but is not included in the sensitivity list (VHDL)..... 2141
NotReqSens :	A signal is included in the sensitivity list of a combinational always block but not all (or none) of its bits are read in that block (Verilog) A signal is included in the sensitivity list of a process block but only some or none of its bits are read in that block (VHDL) 2145
RegOutName :	Register output signal does not follow naming convention. 2149
NoTab :	Tab detected in design 2152
NoDefine :	Macro definitions should not appear in files containing module definitions..... 2153
NoGates :	Directly instantiated gate detected..... 2155
NoTopGates :	Gates instantiated at the top-level of the design..... 2158
NoTopLogic :	Logic inferred at the top level of the design 2161
OnePortLine :	No more than one port should be declared per line..... 2164
OneStmtLine :	More than one statement detected on a line 2166

PortComment : Port is not commented	2168
PortGroups : No blank line between connections to input ports and output ports	2170
PortOrder_A : Ports not declared in recommended order	2173
SigHierName : Non-clock signal does not preserve a similar name through the hierarchy	2177
SignalComment : Signal declaration does not have a comment	2181
SigType : Signal type is not a type or subtype of std_logic or std_logic_vector	2183
TriStateName : Tristate signal does not follow naming convention	2185
TypeComment : Type/Sub-type is not commented	2187
TypeCount : Excessive number of subtypes created	2188
VariableComment : Variable declaration does not have a comment ..	2190
ParamName : Parameter/generic name does not follow the naming convention	2192
PortGrpComment : Port group is not commented	2195
PortName : Port name does not follow the naming convention	2196
ProcessComment : Process is not commented	2200
SigName : Signal name does not follow the naming convention	2201
TaskComment : A comment does not appear immediately before a task definition	2205
TypeName : Type or sub-type does not follow recommended naming convention	2206
UseDefine : Hard constants used to encode finite-state machine	2208
VarName : Variable does not follow recommended naming convention	2212
BufClock : Detects clock buffers	2216
ClkHierName : Clock signal does not keep the same name through the hierarchy	2221
ClkName : Clock name does not follow the naming convention	2224
CombLoop : Detects combinational loops	2227
GatedClock : Gated/generated clocks are generated from more than one block in the design	2233
InferLatch : Latch inferred	2236
IntClock : Internally generated clock detected	2240
IntReset : Detects internally generated resets	2243
NoMixedSynch : Do not mix synchronous and asynchronous logic in one module	2248
NonBlockAssign : Do not use blocking assignments in sequential blocks ..	

	2251
RegOutputs : Some output ports from a top module are not registered	2253
ResetName : Reset name does not follow the naming convention.	2259
UseMuxBusses : Detect tristate nets	2264
NoVar : Use Signals instead of variables to avoid simulation/synthesis mismatch	2267
SepFSMLogic : Finite-state machine (FSM) and non-FSM logic is mixed in one module.....	2269
SepStateMachine : Combinational and sequential parts of a finite-state machine are combined in one process.....	2272
ArchHdr : Standard header not found before architecture declaration	2277
ArchName : Architecture does not follow recommended naming convention	2279
AsyncName : Asynchronous signals do not follow naming convention	2282
AvoidAsync : Asynchronous reset used	2285
ClockDomain : More than one clock domain in a module	2288
ClockPhase : Both positive and negative edges of a clock are used in this design.....	2291
DesgPack : Constants and function definitions defined in design file. .	2294
EntHdr : Standard header not found before entity declaration.....	2296
GateClockAtTop : Gated/generated clocks exist in the designs that are not all generated from one top-level block	2298
GateResetAtTop : Gated/generated resets exist in the design that are not all generated from one top-level block	2301
IEEEType : Signal type is not an IEEE type	2305
ModConst : Modification of constant parameter in subprogram	2307
NoBlock : Block Statement detected	2309
NameLength : Name length is inconsistent with convention	2311
NoGenerate : Generate Statement detected	2315
NoScripts : Synthesis script detected	2317
OneFile : Some of the design units (entity, architectures, configurations, packages) relating to an entity are not found in the same source file.....	2320
PackHdr : Standard header not found before package declaration	2323
ProcName : Process label does not follow the naming convention	2325
ReserveName : A reserve name has been used.....	2328
PortOrder_C : Ports not declared in recommended order.....	2331
SepClock : Both positive and negative edge clocks are used in the same	

module.....	2333
Uniq8Char : Signal, variable, port or other named object is not unique up to defined character limit	2336
Simulation Tags.....	2341
sim_loop01 : Possible mismatch in gate-RTL simulation due to simulation event loops	2342
sim_race04 : Avoid Event Counting Code	2348
sim_race05 : Avoid Time Zero race	2350
sim_race01 : Reports signals that are assigned and used in the same simulation cycle.....	2352
sim_race02 : Reports signals that have multiple assignments in the same simulation cycle.....	2359
sim_race03 : Avoid assigning flip-flop outputs in blocking manner	2363
sim_race06 : Flip-flop data path should not be entirely blocking	2365
sim_race07 : Non-blocking assignment should not be used in clock or enable path.....	2367
sim_race08 : Control signals of a sequential device should not be multiply driven.	2369
STARC05 Tags	2372
STARC05-1.1.1.1 : Source file name should be same as the name of the module in the file. (Verilog) File names containing entities should be <entity_name>.vhd or .vhdl (VHDL)	2373
STARC05-1.1.1.2 : Names should follow recommended naming convention	2377
STARC05-1.1.1.5 : Do not use case variants of names in the same scope. (Verilog) Do not use names that differ in case only, within the same scope (VHDL)	2380
STARC05-1.1.4.1va : File name containing package(s) must end strings specified by parameter starc2005_ext_string_vhdl. (VHDL)	2384
STARC05-1.1.4.4 : Macros defined outside a module must not be used in the module. (Verilog)	2385
STARC05-1.1.4.4v : Constants should not be redefined. (VHDL)	2386
STARC05-1.1.4.6a : Fixed value should not be connected directly to the output ports.....	2387
STARC05-1.2.1.1a : Design should use only one edge of a clock	2388
STARC05-1.2.1.3 : Remove combinational loops	2390
STARC05-1.3.1.6 : Signals must not be used both as asynchronous reset	

and synchronous reset	2393
STARCO5-1.3.1.7 : A flip-flop must not have both asynchronous set and asynchronous reset.....	2397
STARCO5-1.3.2.1b : Reset must be generated in separate module instantiated at top-level.....	2401
STARCO5-1.3.2.2 : Flip-flop outputs must not be used as reset signals	2404
STARCO5-1.4.1.1 : Preferably get all clock nets from primary or centrally generated clock	2408
STARCO5-1.4.3.1c : Both clock edges should not be used in the same module.....	2410
STARCO5-1.4.4.2 : One hierarchical level should have a single clock only. (Verilog).....	2414
STARCO5-1.6.2.1 : Top-level module outputs should be directly driven by flip-flops.	2416
STARCO5-1.6.2.2 : Combinatorial Path should not span more than specified number of modules	2419
STARCO5-2.1.1.1v : Package IEEE.std_logic_1164 must be included in each entity. (VHDL)	2422
STARCO5-2.1.1.2 : A function description must assign return values to all possible states of the function. (Verilog)	2423
STARCO5-2.1.2.1 : Function statement should not be used for asynchronous reset line logic in an always construct for FF inference. (Verilog)	2426
STARCO5-2.1.2.1v : Use std_logic or std_logic_vector data types to describe ports of an entity. (VHDL)	2427
STARCO5-2.1.2.2 : Do not use nonblocking assignment in function description. (Verilog).....	2428
STARCO5-2.1.2.2v : Use std_logic, std_logic_vector, integer, Boolean, unsigned, signed, time, or user-defined data type inside the architecture. (VHDL)	2429
STARCO5-2.1.2.3 : Global signals must not be read in a function description. (Verilog).....	2430
STARCO5-2.1.2.4 : task constructs should not be used in the design. (Verilog).....	2431
STARCO5-2.1.2.4v : Do not use bit and bit_vector data types in the design. (VHDL).....	2432
STARCO5-2.1.2.5 : Clock edges should not be used in a task description. (Verilog).....	2433

STARC05-2.1.2.5v : Do not use attribute enum_encoding. (VHDL) ...	2435
STARC05-2.1.3.1 : Ensure that the bit-width of the function arguments match the bit-width of the corresponding function inputs.(Verilog)	2437
STARC05-2.1.3.4 : Return value assignment statement must be the last statement in function description. (Verilog)	2443
STARC05-2.1.3.4v : Input port must not be described with initial value. (VHDL)	2445
STARC05-2.1.4.1 : Parenthesis should be used in expressions with multiple operators. (Verilog)	2446
STARC05-2.1.4.5 : Use bit-wise operators instead of logic operators in multi-bit operations.....	2447
STARC05-2.1.5.1 : Conditional operators (?) should not be nested. (Verilog)	2449
STARC05-2.1.5.3 : Ensure that conditional expressions evaluate to a scalar.	2450
STARC05-2.1.6.1v : Specification of one-dimensional array should be downto (VHDL)	2453
STARC05-2.1.6.2 : LSB of vectors/memory should be zero. (Verilog) LSB of one-dimensional arrays should be zero. (VHDL)	2454
STARC05-2.1.6.5 : Do not use x and z for an array index	2456
STARC05-2.2.1.2 : Design should not have latches.....	2458
STARC05-2.3.1.1 : Blocking assignment must not be used in sequential always blocks. (Verilog) Variable assignment statement must not be used in flip-flop description (VHDL)	2460
STARC05-2.3.2.2v : Variable assignment statement should not be used in a process block. (VHDL)	2462
STARC05-2.4.1.4 : Avoid combinational loops that contain latches ...	2463
STARC05-2.10.1.3 : Comparison should not be made with X, H, L, U, W, - or Z. (VHDL)	2467
STARC05-2.10.1.4a : Signals must not be compared with X or Z. (Verilog)	2469
STARC05-2.10.1.4b : Signals must not be compared with values containing X or Z. (Verilog)	2470
STARC05-2.10.1.4c : Signals must not be compared with values containing X or Z in case statement. (Verilog)	2471
STARC05-2.10.1.4v : Do not assign 'X' except for the others choice of case statements (VHDL)	2472

STARCO5-2.10.1.5 : Do not use values including 'X','Z','U','-','W','H','L'. (VHDL).....	2473
STARCO5-2.10.1.6 : Constants must not contain X or Z. (Verilog)	2474
STARCO5-2.10.2.3 : Do not perform logical negation on vectors.	2475
STARCO5-2.10.3.1v : Bit-width of the operands of relational operator should match. (VHDL)	2478
STARCO5-2.10.3.2b : LHS and RHS bit-widths in signal assignments must match. (Verilog)	2479
STARCO5-2.10.3.2b_sa : RHS width is greater than LHS width in assignment.....	2485
STARCO5-2.10.3.2b_sb : RHS width is lesser than LHS width in assignment.....	2491
STARCO5-2.10.3.7 : Match the bit-width with the base number part. (Verilog).....	2497
STARCO5-2.10.4.1 : Do not use data types other than reg, wire, and integer. (Verilog)	2500
STARCO5-2.10.4.4 : Negative value should not be assigned to integer. (VHDL).....	2501
STARCO5-2.10.4.5 : Integer objects must not be assigned negative values. (Verilog).....	2502
STARCO5-2.10.4.6 : Signals should not be assigned negative values. (Verilog).....	2503
STARCO5-2.10.4.6v : Signed data type must be used in signed operation. (VHDL).....	2505
STARCO5-2.10.8.1 : Do not use arithmetic and logical expressions at the right and left sides of the division or modules operator.	2506
STARCO5-2.11.3.1 : Ensure that the sequential and combinational parts of an FSM description should be in separate always blocks. (Verilog) Ensure that the sequential and combinational parts of an FSM description should be separated (VHDL).....	2508
STARCO5-2.2.2.3a : Multiple event Expressions should not be described with always constructs. (Verilog)	2515
STARCO5-2.2.3.3 : Do not assign over the same signal in an always construct for sequential circuits	2516
STARCO5-2.3.1.2a : Do not use quasi-continuous assignments. (Verilog) . 2522	
STARCO5-2.3.1.2b : Do not use deassign statements. (Verilog)	2523
STARCO5-2.3.1.2c : Do not use unsynthesizable User-Defined Primitives (UDPs).	2524

STARCO5-2.3.1.2va : (Do not use rising_edge or falling_edge of clock for flip-flop description. (VHDL)	2527
STARCO5-2.3.1.2vb : Do not use stable attribute for flip-flop description. (VHDL)	2528
STARCO5-2.3.1.2vc : Do not use non-synthesizable clocking styles for flip-flop description. (VHDL)	2529
STARCO5-2.3.1.5b : Ensure that the delay values are non-negative..	2531
STARCO5-2.3.1.6 : Check the logic level of the reset signal as specified in the sensitivity list of the always block.	2533
STARCO5-2.3.3.1 : Do not use edges of multiple clocks in same always block. (Verilog) Do not use multiple clock signals in a single process block (VHDL)	2535
STARCO5-2.3.3.2b : Do not describe multiple clock edges in a single process/always block for the same edge of a single clock ..	2538
STARCO5-2.3.4.2 : Reports initial construct used.	2539
STARCO5-2.3.6.2a : Multiple sets/resets should not be used in an always block. (Verilog) Multiple asynchronous resets should not be used in a process block. (VHDL)	2541
STARCO5-2.3.6.2b : Asynchronous reset description should use negedge only.	2543
STARCO5-2.4.1.2 : Latch descriptions should not be mixed with other descriptions in same module.....	2545
STARCO5-2.5.1.1 : Tristate descriptions must not be mixed with other descriptions in same module.....	2547
STARCO5-2.6.2.2 : A variable in sensitivity list is modified inside the same always block. (Verilog) A variable in sensitivity list is modified inside the same process block. (VHDL)	2549
STARCO5-2.7.1.3b : An "if" clause must have a corresponding "else" clause in describing sequential circuits. (Verilog) An IF clause must have a corresponding ELSE clause in sequential circuits (VHDL)	2551
STARCO5-2.7.2.2 : Avoid describing case constructs.....	2553
STARCO5-2.7.2.3 : Avoid null condition expressions.	2554
STARCO5-2.8.1.5 : Do not use the full_case directive. (Verilog)	2556
STARCO5-2.8.3.3 : Do not use //synopsys full_case pragma when all conditions are not described as case clause or the default clause is missing. (Verilog)	2557

STARCO5-2.8.3.4 : A signal that is assigned don't care value in a case default clause should not be used in conditional constructs (if, if-elsif, case, ternary).....	2558
STARCO5-2.8.3.4a : A signal that is assigned don't care value in a case default clause should not be used in if conditions. (Verilog) A signal or variable, which is assigned don't-care value in an OTHERS clause should not be used in if-elsif conditions. (VHDL)	2559
STARCO5-2.8.3.4b : A signal that is assigned don't care value in a case default clause should not be used in "ternary" and "case" constructs. (Verilog) A signal that is assigned a don't care value in others clause should not be used in the case construct. (VHDL).....	2562
STARCO5-2.8.3.5 : default clause in case constructs must be the last clause. (Verilog)	2564
STARCO5-2.8.5.1 : Do not use the parallel_case directive. (Verilog) .	2565
STARCO5-2.8.6.1 : 'case' statements and nested 'if' statements co-exist, better to have fewer conditionals with multiple matches than fewer matches with multiple conditionals.	2566
STARCO5-2.9.1.2e : Do not use loop variable outside of the "for" loop. (Verilog).....	2573
STARCO5-3.1.4.4 : Multiple assignments should not be made in one line. (Verilog) Multiple assignment statements should not be made in one line (VHDL).....	2574
STARCO5-3.2.2.2b : Define macros should be read using include files. (Verilog).....	2576
STARCO5-3.2.2.4 : Include files must be specified with more than 1 level higher relative path. (Verilog).....	2577
STARCO5-3.2.3.1 : Port connections in instantiations must be made by named association rather than positional association. (Verilog) .	2578
STARCO5-3.2.3.1v : Port/Generic connections in instantiations must be made by named association rather than positional association. (VHDL).....	2579
STARCO5-3.2.3.2 : Bit-width of the component port and its connected net must match. (Verilog)	2580
STARCO5-3.2.4.3 : Do not use defparam statements. (Verilog)	2583
STARCO5-3.5.3.1 : A standard header should be provided in every source file.....	2584

STARC05-1.2.1.2 : Do not create an RS latch using primitive cells such as AND, OR.....	2586
STARC05-1.3.1.3 : Do not use asynchronous reset/preset signals as non-reset/preset or synchronous reset/preset signals.....	2588
STARC05-1.4.3.1a : Clocks should not be inverted	2595
STARC05-1.4.3.1b : Clocks should not be gated	2597
STARC05-1.4.3.2 : Do not use flip-flop outputs as clocks of other flip-flops 2600	2600
STARC05-1.4.3.4 : Do not use flip-flop clock signals as non-clock signals . 2603	2603
STARC05-2.1.1.1 : Combinational logic should consistently be described using either always constructs or function and assign constructs. (Verilog)	2608
STARC05-2.2.3.1 : Non-blocking assignment should not be used in combinational always blocks (may lead to shoot through). (Verilog)	2611
STARC05-2.3.1.7a : A flip-flop should not have two asynchronous resets . 2612	2612
STARC05-2.3.4.1v : Ensure that a flip-flop does not have an initial value in signal/variable description.....	2613
STARC05-2.3.5.1 : Flip-flops should not have fixed inputs	2617
STARC05-2.3.6.1 : Do not mix descriptions of flip-flops with asynchronous reset and flip-flops without asynchronous reset in the same process/always block.....	2618
STARC05-1.1.1.10 : Instance or module name must be different from the cell names of the ASIC library used. (Verilog) Instance/Entity name should be different from those in the ASIC library (VHDL)	2621
STARC05-1.1.1.3 : Do not use the reserved Verilog, SystemVerilog, or VHDL keywords. (Verilog)	2623
STARC05-1.1.1.3va : Do not use reserved keywords of Verilog-HDL and SystemVerilog. (VHDL)	2627
STARC05-1.1.1.3vb : Do not use standard VHDL names (VHDL)	2630
STARC05-1.1.1.4 : Do not use names starting with the user specified pattern (specified by the parameter 'starc2005_compare_str')(Verilog).	2631
STARC05-1.1.1.6 : Primary port names or module names must follow recommended naming convention. (Verilog)	2633
STARC05-1.1.1.6v : Component name should be same as its	

corresponding entity name. (VHDL).....	2634
STARCO5-1.1.1.7 : Negative logic signals must follow recommended naming convention	2635
STARCO5-1.1.1.8 : Module instance names should be same as or based on the module name. (Verilog) Instance names should follow recommended naming convention. (VHDL).....	2639
STARCO5-1.1.1.8v : Instance names should follow recommended naming convention. (VHDL).....	2641
STARCO5-1.1.1.9a : Top-level module names should follow recommended naming convention. (Verilog) Top-level entity names should follow recommended naming convention. (VHDL).....	2642
STARCO5-1.1.1.9b : Top-level module name should be less than or equal to 16 characters in length (Verilog) Top-level entity name should be less than or equal to 16 characters in length (VHDL)	2644
STARCO5-1.1.1.9c : Top-level port names should follow recommended naming convention	2645
STARCO5-1.1.1.9d : Top-level port name should be less than or equal to 16 characters in length.	2646
STARCO5-1.1.2.1a : Module names should be between number of characters in length specified by parameters starc2005_min_mod_ent_length and starc2005_max_mod_ent_length. (Verilog) Entity names should be between number of characters in length specified by parameters starc2005_min_mod_ent_length and starc2005_max_mod_ent_length. (VHDL).....	2647
STARCO5-1.1.2.1b : Instance names should be between number of characters in length specified by parameters starc2005_min_inst_length and starc2005_max_inst_length. ...	2649
STARCO5-1.1.2.2 : Top layer instance names should start with the layer identification character	2650
STARCO5-1.1.2.3 : Instance names should start with the first character of the parent module name. (Verilog) Instance name should start with layer identification character (VHDL).....	2652
STARCO5-1.1.2.4 : Output port names should start with the first character of the module name. (Verilog)	

Output pin name of each entity should start with the layer identification character followed by '_' (VHDL)	2654
STARCO5-1.1.2.5 : Port names for each block should be in uppercase. (Verilog)	
Port names for each entity should be in uppercase (VHDL)	2656
STARCO5-1.1.2.6 : Connected net name of a module instance must be same as or based on the master module input/output port name. (Verilog)	
Connected net name of a instance must be same as or based on the corresponding master entity input/output port name. (VHDL)	2657
STARCO5-1.1.2.6a : Connected net name of a module instance must be same as or based on the master module output port name. (Verilog)	
Connected net name in an instance must be same as or based on the corresponding master entity output port name (VHDL)	2658
STARCO5-1.1.2.6b : Connected net name of a module instance must be same as or based on the master module input port name. (Verilog)	
Connected net name of a instance must be same as or based on the corresponding master entity input port name. (VHDL)	2661
STARCO5-1.1.3.1 : Internal signal names should have at least one lowercase alphabet. (Verilog)	
Internal signal names should have at least one lowercase character (VHDL)	2664
STARCO5-1.1.3.3 : Signal names, port names, parameter names, generic names, function names, and define names should be between 2 and 40 characters in length	2665
STARCO5-1.1.3.3a : Signal name should be between 2 and 40 characters in length	2666
STARCO5-1.1.3.3b : Port name should be between 2 and 40 characters in length	2667
STARCO5-1.1.3.3c : Parameter name should be between 2 and 40 characters in length. (Verilog)	
Generic name should be between 2 and 40 characters in length. (VHDL)	2668
STARCO5-1.1.3.3d : Function name should be between 2 and 40 characters in length	2670
STARCO5-1.1.3.3e : Define macro names should be between 2 and 40 characters in length. (Verilog)	2671

STARCO5-1.1.4.1 : Include files must have extensions that match any of the extensions specified by the parameter starc2005_ext_string_vlog. (Verilog).....	2672
STARCO5-1.1.4.1vb : Package and entity descriptions must not be mixed in same file (VHDL).....	2673
STARCO5-1.1.4.2a : Parameter names should follow recommended naming convention. (Verilog)	2674
STARCO5-1.1.4.2b : Define macro names should start with "D_". (Verilog) 2675	
STARCO5-1.1.4.2v : Constant names should follow recommended naming convention. (VHDL).....	2676
STARCO5-1.1.4.3 : Do not use parameters with same name for different modules. (Verilog)	2677
STARCO5-1.1.4.3v : Do not use constant declarations with same name in different entities/architectures. (VHDL)	2678
STARCO5-1.1.4.5 : Parameters used only in a single layer must start with the layer identification character. (Verilog)	2679
STARCO5-1.1.4.7 : Vector port bit-widths should be specified using parameters. (Verilog) Port bit-width should be specified using generics. (VHDL)..	2680
STARCO5-1.1.4.8 : Parameters with numeric value more than nine must have base type specification. (Verilog).....	2682
STARCO5-1.1.4.8v : Initial value in vector constant declaration must have base specifier. (VHDL)	2683
STARCO5-1.1.4.9 : Bit-width must be specified for parameters with more than 32 bits. (Verilog)	2684
STARCO5-1.1.4.9v : Generic must not be used at top-level module. (VHDL) 2685	
STARCO5-1.1.6.1 : Architecture name must contain "RTL". (VHDL)...	2686
STARCO5-1.1.6.4 : An entity and its architectures must be described in same file (VHDL)	2687
STARCO5-1.3.1.2 : Do not use synchronous reset in the design	2688
STARCO5-1.3.1.5a : Do not use Design Compiler synchronous set and reset directives.....	2691
STARCO5-1.3.1.5b : Do not use Design Compiler asynchronous set and reset directives.....	2692
STARCO5-1.4.3.6 : Do not use Flip-Flops with inverted edges	2693
STARCO5-1.6.1.4 : Clock generation, reset generation, RAM, Setup/Hold ensure buffers, and I/O cells must be a module at top-level.....	

	2694
STARC05-2.1.1.2v : At least one of the packages IEEE.std_logic_arith and IEEE.std_logic_unsigned must be included in each entity. (VHDL)	2695
STARC05-2.1.1.3 : When using IEEE.numeric_std, do not mix with std_logic_unsigned or std_logic_arith	2696
STARC05-2.1.10.1 : Do not use block statement in the design (VHDL).....	2697
STARC05-2.1.10.10 : Alias declarations should not be used in the design. (VHDL)	2698
STARC05-2.1.10.11 : Bus and register signals should not be used in the design. (VHDL)	2699
STARC05-2.1.10.12 : Disconnect specification must not be used in the design. (VHDL)	2700
STARC05-2.1.10.13 : Waveforms should not be used in the design. (VHDL)	2701
STARC05-2.1.10.2 : Do not use object of type record (VHDL)	2702
STARC05-2.1.10.3 : Do not use shared variable in the design (VHDL).....	2703
STARC05-2.1.10.4 : Do not use while-loop statement in the design (VHDL)	2704
STARC05-2.1.10.5 : Do not use procedure in the design (VHDL)	2705
STARC05-2.1.10.6 : Do not use selected signal assignment in the design (VHDL)	2706
STARC05-2.1.10.8 : Do not use attributes, which are defined in Synopsys library (VHDL)	2707
STARC05-2.1.10.9 : Access type should not be used in the design. (VHDL)	2708
STARC05-2.1.2.3v : Use range specification for integer types. (VHDL)	2709
STARC05-2.1.2.6 : Specify Range for std_logic_vector (VHDL)	2711
STARC05-2.1.3.1v : Linkage or buffer ports must not be described. (VHDL)	2712
STARC05-2.1.3.2 : Bit-width of function return value must match the bit-width of assignment destination signal. (Verilog)	
Do not use buffer and linkage (VHDL)	2713
STARC05-2.1.3.3 : Port mode must be explicitly specified (VHDL)	2715
STARC05-2.1.3.5 : Global signals must not be assigned in function description. (Verilog)	2716

STARCO5-2.1.4.2 : Bit-wise operators should be used instead of logic operators in single-bit operations. (Verilog).....	2717
STARCO5-2.1.4.2v : Shift operator should not be used in the design. (VHDL).....	2718
STARCO5-2.1.4.6a : Reduction of a single-bit expression should not be performed (Verilog)	2719
STARCO5-2.1.4.6b : Reduction of Large expression should not be performed (Verilog)	2720
STARCO5-2.1.5.1v : When-else nesting in conditional signal assignment statement must not exceed 5 levels. (VHDL)	2721
STARCO5-2.1.6.1 : Array specification should follow recommended convention. (Verilog).....	2722
STARCO5-2.1.6.3 : Expressions should not be used as array indexes. (Verilog) Expression should not be used as array indices. (VHDL)	2723
STARCO5-2.1.6.4 : Index variable width should not be short (Verilog)	2724
STARCO5-2.1.7.1 : Direct assignment must be used for aggregates (VHDL)	2726
STARCO5-2.1.7.3 : Bit position based and others based assignment should not be used together for aggregate specification (VHDL) ...	2727
STARCO5-2.1.8.1 : Constrained arrays should not be used as sub-program arguments (VHDL)	2728
STARCO5-2.1.8.10 : No other statement should follow the return statement in function description (VHDL)	2729
STARCO5-2.1.8.2 : Specify range for return values in function description when return type is array (VHDL)	2730
STARCO5-2.1.8.4 : Only OTHERS clause must be used when specifying aggregate in sub-program description (VHDL)	2731
STARCO5-2.1.8.5a : Do not use nested sub-program description in the design (VHDL).....	2732
STARCO5-2.1.8.5b : Do not call a sub-program within a subprogram (VHDL).....	2733
STARCO5-2.1.8.6 : Global signals must not be read in a subprogram description (VHDL).....	2734
STARCO5-2.1.8.9 : A function must have the return statement and return a valid value in all possible states of the function. (VHDL) .	2735
STARCO5-2.1.9.4 : Do not use built-in attribute except range, length, left, right, high, low, reverse_range, event (VHDL)	2736

STARCO5-2.1.9.5 : Do not use user-defined attributes in the design (VHDL)	2737
STARCO5-2.10.1.2 : Parentheses should be used for logical operations to improve readability. (Verilog) Parenthesis should be used in expressions with multiple operators (VHDL).....	2739
STARCO5-2.10.1.7 : Do not use RAM output signals for a conditional expression of if statements. (Verilog)	2741
STARCO5-2.10.1.8 : Do not use RAM output signals for the selection expression of case statements that do assign 'x' in the default clause. (Verilog)	2742
STARCO5-2.10.3.1 : Bit-widths of operands of a relational operator should match. (Verilog)	2743
STARCO5-2.10.3.2a : Bit-width of operands of a logical operator must match. (Verilog)	2747
STARCO5-2.10.3.2c : Unequal length operands in bit/arithmetic operator (Verilog).....	2749
STARCO5-2.10.3.5 : Base should be specified for constant value used. (Verilog).....	2751
STARCO5-2.10.3.6 : Bit-width should be specified for based constants of width more than 4 bits. (Verilog)	2753
STARCO5-2.10.4.3 : Std_logic_vector(conv_unsigned(&, &)) should be used to convert integer to std_logic_vector (VHDL)	2755
STARCO5-2.10.4.7 : Qualified expressions should not be used in the design. (VHDL)	2756
STARCO5-2.10.4.8 : Function To_stdlogicvector should not be used in the design. (VHDL)	2758
STARCO5-2.10.5.1 : Arithmetic/relational operation with operand bit-width more than 8 in multiple conditional expressions is same. (Verilog)	2759
STARCO5-2.10.5.2 : Do not use any arithmetic operator twice in the same "if" statement.....	2761
STARCO5-2.10.5.3 : Do not use any arithmetic operator thrice or more in the same "if" statement.	2762
STARCO5-2.10.5.3a : Reports violation when there are two arithmetic operations in the Always block	2763
STARCO5-2.10.5.3b : Reports violation when there are three or more arithmetic operations in Always block	2767
STARCO5-2.10.5.5 : Do not describe arithmetic operations with conditional operators (?) in assign statement. (Verilog).....	2771

STARCO5-2.10.6.1 : Possible loss of carry or borrow in addition or subtraction (Verilog)	2772
STARCO5-2.10.6.5 : Large multipliers must not be described using the multiplication operator.....	2778
STARCO5-2.10.6.6 : Do not describe more than one arithmetic operation in one line	2781
STARCO5-2.10.7.1 : Do not use arithmetic operation in the conditional expression of if statements.	2782
STARCO5-2.10.7.2 : Common operational expressions should be described separately. (Verilog) Separate the common operational expressions in if-elsif branch conditions (VHDL).....	2785
STARCO5-2.10.8.2 : Keep the left side of the division or modulus operator within 12 bits.	2787
STARCO5-2.10.8.3 : If right side of the division or modulus operator is not a power of two, keep it within 8 bits.	2788
STARCO5-2.11.1.2 : Bit change in state transition should be minimal (use Gray code). (Verilog) Bit change in FSM state transitions should be minimized (use Gray code). (VHDL)	2790
STARCO5-2.11.1.4 : Number of states in a FSM should not exceed specified limit.....	2792
STARCO5-2.11.2.1 : FSM description should not be mixed with other descriptions in the same module. (Verilog) FSM description should not be mixed with non-FSM descriptions in the same module (VHDL)	2794
STARCO5-2.11.4.1 : One-hot encoding should be used for allocating states	2796
STARCO5-2.11.4.2 : State values should be defined using parameters or macros. (Verilog) Use constant to facilitate allocation changes (VHDL)	2797
STARCO5-2.11.5.2 : Do not assign state encoding by attaching attributes to the state variable which is declared as a type (VHDL) ...	2798
STARCO5-2.2.2.1 : Signals read inside a combinational always block must be present in the sensitivity list (Verilog) Signals read inside a combinational process block must be present in the sensitivity list (VHDL).....	2799
STARCO5-2.2.2.2a : A variable specified in the sensitivity list must be read in the contained block. (Verilog) A signal should not be included in sensitivity list of a process if it	

is not read in that process. (VHDL).....	2801
STARCO5-2.2.2.2b : Do not define constants in the sensitivity list	2804
STARCO5-2.2.2.3v : Do not use wait statement in process statements (Sensitivity list should be used). (VHDL)	2805
STARCO5-2.2.2.3b : At least one event expression must be described for an always construct. (Verilog)	2806
STARCO5-2.2.3.2 : Do not use nonblocking assignments for assigning initial values in always blocks. (Verilog)	2807
STARCO5-2.3.1.3 : Flip-flop data paths should have delays.....	2808
STARCO5-2.3.1.4 : Do not use delay values other than in always constructs that infer FFs. (Verilog) Paths other than flip-flop data paths must not have delays. (VHDL)	2810
STARCO5-2.3.1.5a : Delay values must be integral.....	2812
STARCO5-2.3.1.7b : Do not use function in asynchronous reset description. 2814	
STARCO5-2.3.1.8 : Do not use wait construct to describe clock style (VHDL)	2815
STARCO5-2.3.1.9 : Functions 'rising_edge/falling_edge' should not be used in the design. (VHDL)	2816
STARCO5-2.3.2.2 : Do not use blocking and nonblocking assignments together in same always block. (Verilog)	2817
STARCO5-2.3.2.4 : Variables, if used, must be assigned to a signal before the end of the process (VHDL)	2819
STARCO5-2.3.3.2a : Do not use both edges of the same clock in an always block. (Verilog) Do not use both edges of the same clock in a single process block (VHDL)	2823
STARCO5-2.3.4.1 : Do not specify flip-flop initial value using initial construct. (Verilog)	2825
STARCO5-2.5.1.5b : A signal without a resolution function must not have multiple drivers. (VHDL)	2827
STARCO5-2.6.1.2 : Use intermediate variables when same logic is used in more than one place.	2828
STARCO5-2.6.1.3 : always construct has more than five output signals. (Verilog) A process block should not have more than five outputs (VHDL) 2829	
STARCO5-2.6.1.4a : Always/Process construct should not have more than	

300 lines	2831
STARCO5-2.6.1.4b : Always/Process construct should not have more than 2000 lines.....	2833
STARCO5-2.6.2.1 : Do not describe more than one statements (if/case/while/for/forever/repeat) separately within a single always block (Verilog)	
Do not describe more than one statements (if/case/while/for/loop) separately within a single process block (VHDL)	2835
STARCO5-2.7.1.3a : An "if" clause must have a corresponding "else" clause in describing combinational circuits. (Verilog)	
An IF clause must have a corresponding ELSE clause in combinational circuits (VHDL)	2838
STARCO5-2.7.2.1 : Common part of conditional expressions should be prioritized. (Verilog)	
Common part of conditional expressions must be prioritized (VHDL)	2840
STARCO5-2.7.3.1a : Nesting in if-else constructs should not be deeper than the number of levels as specified by parameter ifelse_nestlevel_reco (default 7).	2843
STARCO5-2.7.3.1b : Where feasible case statements should be used rather than 'if-else' statements, if performance is important. (Verilog)	
Use "case" statements instead of if/else statements, where feasible, if performance is important. (VHDL)	2845
STARCO5-2.7.3.1c : Chain of if...else if constructs must not be longer than number of levels as specified by parameter starc2005_ifelse_chainlevel_reco (default 7). (Verilog)	
The chain of 'if-elsif' construct must not be longer than number of levels specified by parameter starc2005_ifelse_chainlevel_reco (default 7). (VHDL)	2848
STARCO5-2.7.3.4 : Nested if-statements should be merged	2850
STARCO5-2.7.4.2 : Attach begin-end to "if" statements.	2853
STARCO5-2.7.4.3 : Do not use fork-join constructs. (Verilog).....	2854
STARCO5-2.8.1.3 : case constructs should not have overlapping clause conditions. (Verilog)	2855
STARCO5-2.8.1.4 : case constructs must have the default clause. (Verilog)	
Case statements and selected signal assignment statements must always have OTHERS clause (VHDL)	2857
STARCO5-2.8.1.6 : The case expression width should match case select expression width (Verilog)	2860
STARCO5-2.8.2.1 : if constructs should be used instead of large case	

constructs.....	2863
STARC05-2.8.2.2 : Item count should be within the value specified by parameter 'starc2005_maxTableSize'.....	2865
STARC05-2.8.3.1 : default clause output in case constructs should not be a constant value. (Verilog) Constant values should not be assigned in OTHERS clause of case constructs (VHDL).....	2866
STARC05-2.8.3.2 : Either default clause does not exist or 'X' is not assigned in the default clause. (Verilog) Either others clause does not exist or 'X' is not assigned in the others clause. (VHDL).....	2868
STARC05-2.8.3.6 : Do not use a signal to which don't care is assigned for selection expression of case statements that do not assign 'X' in default clause. (Verilog) Do not use a signal to which don't care is assigned for selection expression of case statements that do not assign 'X' in 'others' clause. (VHDL)	2870
STARC05-2.8.3.7 : Do not use a signal to which don't care is assigned for selection expression of casex statements. (Verilog)	2871
STARC05-2.8.4.1a : casex or casez constructs must contain a condition with don't care (Verilog)	2872
STARC05-2.8.4.1b : Coding style of don't-care condition in casex or casez branches is not proper (Verilog)	2873
STARC05-2.8.4.3 : Design should not use casex or casez constructs. (Verilog)	2875
STARC05-2.8.4.4 : casex or casez constructs must not have complex clause conditions. (Verilog)	2876
STARC05-2.8.5.2 : Do not use fixed values in case construct condition. (Verilog)	2877
STARC05-2.8.5.3 : Do not use variables or expressions as case clause conditions. (Verilog)	2878
STARC05-2.8.5.4 : Logical, arithmetic, bit wise or reduction operations should not be used in case construct condition. (Verilog)...	2880
STARC05-2.9.1.1 : "for" constructs must be used only for simple repeating statements. (Verilog) FOR constructs must be used only for simple repeating statements (VHDL)	2881
STARC05-2.9.1.2a : Loop variable of "for" construct must have constant initial value. (Verilog)	2884
STARC05-2.9.1.2b : Terminating condition of "for" constructs must be a	

constant. (Verilog)	2885
STARCO5-2.9.1.2c : The value of loop variable must not be modified inside the "for" construct. (Verilog)	2887
STARCO5-2.9.1.2d : The loop termination condition must not be a constant. (Verilog)	2889
STARCO5-2.9.2.1 : Arithmetic operations on non-constants within for-loop must not be performed	2890
STARCO5-2.9.2.2 : Logical and Relational operations on non-constants within for-loop should not be performed.....	2893
STARCO5-2.9.2.3 : If logical or relational operation of non-constant variables exists in a "for" construct, the number of loop iterations must not be more than the number specified by the parameter 'maxLoopIter' (Default value 10).	2896
STARCO5-2.9.2.4 : Separate for loops must be used in reset and logic parts of flip-flop descriptions. (Verilog + VHDL)	2899
STARCO5-3.1.2.7 : Do not use embedded dc_shell scripts. (Verilog) Do not use embedded dc_shell scripts in the design (VHDL)	2903
STARCO5-3.1.3.1 : The order of module port declarations and instance port connection lists should be same as the order in the module port map.(Verilog).....	2905
The order of entity port declarations and instance port map should be same as the order in the entity port map. (VHDL)	2905
STARCO5-2.9.3.1 : Exit or next statement must not be used in a for loop (VHDL).....	2909
STARCO5-3.1.3.2vc : Ports not declared in recommended order. (VHDL) .	2910
STARCO5-3.1.3.4a : Only one port or signal declaration should be in one line. (Verilog) Define one signal per line in I/O, reg declaration, and wire declaration (VHDL)	2911
STARCO5-3.1.3.4b : Port and signal declarations should have comments. (Verilog) Declarations should have comments (VHDL).....	2912
STARCO5-3.1.3.5 : Description order of declarations should be type/ subtype declaration, constant declaration, signal declaration and component declaration. (VHDL)	2913
STARCO5-3.1.4.2 : Code is not correctly indented. (Verilog) Standardize the number of indents (two spaces). (VHDL) ..	2915
STARCO5-3.1.4.3 : Tabs should not be used. (Verilog)	

Tabs should not be used in the RTL description. (VHDL)	2917
STARCO5-3.1.4.5 : The maximum number of characters in one line should not be more than 110.....	2918
STARCO5-3.1.5.2 : Vector signal bit-widths should be specified using parameters. (Verilog)	2919
STARCO5-3.1.5.2v : Non-port vectors bit-width should be specified using generics. (VHDL)	2920
STARCO5-3.1.6.1 : Generic should be used in conditional expression of if-generate statement (VHDL).....	2921
STARCO5-3.1.6.2 : Generic should be used in loop parameter range specification of for-generate statement (VHDL)	2922
STARCO5-3.2.2.1 : Parameters should be used to specify constants (Verilog)	2923
STARCO5-3.2.2.1v : Do not use hard-coded constant in the design. (VHDL)	2924
STARCO5-3.2.2.3 : Parameters should be read using include files. (Verilog)	2925
STARCO5-3.2.2.4 : Include files must be specified with more than 1 level higher relative path. (Verilog).....	2926
STARCO5-3.2.2.5 : Text macros should not be nested. (Verilog).....	2927
STARCO5-3.2.2.7 : Constants should be defined using parameters only. (Verilog)	2928
STARCO5-3.2.3.3 : Do not use entity instantiation in the design (VHDL)...	2929
STARCO5-3.2.4.1 : Non-integer type used in the declaration of a generic may be unsynthesizable. (VHDL).....	2930
STARCO5-3.5.2.1 : File references must be relative path names including the master directory. (Verilog).....	2931
STARCO5-3.5.2.1v : Each file's reference should be specified by a relative path (return to the master directory once). (VHDL)	2932
STARCO5-3.5.6.2 : Add comments for sub-programs, always construct and condition statements, indicating objective and contents. (Verilog)	2933
STARCO5-3.5.6.2va : Process blocks should have comments. (VHDL)	2934
STARCO5-3.5.6.2vb : Sub-program descriptions should have comments. (VHDL)	2935
STARCO5-3.5.6.3a : Each declaration should be described in separate line	2936
STARCO5-3.5.6.3b : Declarations should have comments (VHDL).....	2937

STARC05-3.5.6.4	: Comments should use English characters only. ...	2938
STARC05-3.5.6.6	: Japanese comment should be described in EUC code. .	2939
STARC05-3.5.6.7	: Comments should start with "// " (Verilog).....	2940
STARC05-1.1.5.2b	: Reset signal names should follow the recommended naming convention	2941
STARC05-1.3.2.1a	: Asynchronous resets or sets must not be gated	2943
STARC05-2.1.2.2a	: Do not use nonblocking assignment in task description. (Verilog).....	2945
STARC05-2.3.2.1	: Blocking assignment used in flip-flop description. (Verilog).....	2947
STARC05-2.4.1.3	: Latches should not have asynchronous set or asynchronous reset. (Verilog) Latches must not have asynchronous set or asynchronous reset (VHDL).....	2948
STARC05-2.4.1.5	: Do not use two level latches in the same phase clock .	2949
STARC05-2.5.1.2	: Ensure that logic does not exist in tristate enable conditions.....	2952
STARC05-2.5.1.4	: Identifies the tristate bus that is driven by more than specified number of drivers	2960
STARC05-2.5.1.5a	: Nets other than tristate nets must not be driven by multiple drivers. (Verilog) Net that is not tristated has multiple drivers. (VHDL)	2964
STARC05-2.5.1.7	: Do not use tristate output in the conditional expression of an if statement	2968
STARC05-2.5.1.8	: Tristate output should not be used in the selection expression of a case statement that assigns a fixed value in others choice	2974
STARC05-2.5.1.9	: Do not enter a tristate output in the selection expression of casex and casez statement.	2975
STARC05-3.3.1.4b	: A flip-flop should have an asynchronous set or an asynchronous reset.....	2978
STARC05-3.3.3.1	: A clock must not be connected to the D input of a FF .	2980
STARC05-1.1.4.6b	: Fixed value should not be connected to the input ports	2981
STARC05-3.3.6.2	: Do not mix clock lines and reset lines.....	2982
STARC05-3.1.3.2a	: Ports should be declared in recommended order.	

(Verilog)	2984
STARC05-3.1.3.2b : Ports should be declared in recommended order.	
(Verilog)	2985
Starcad-21 Tags	2986
ALLOW_CELL : Allowed cells should be used in the design	2987
CHIP_MODULE_NAME_LENGTH : The name lengths of chip-level modules should not be longer than 16	2988
BOTH_SETRESET : Flip-flops or latches must not have both asynchronous set and asynchronous reset.	2989
CHIP_PORT_NAME_LENGTH : The name lengths of chip-level ports should not be longer than 16	2990
DISALLOW_CELL : Disallowed cells that are specified by user are used in the design.....	2991
RESERVE_NAME : VDD/VSS/VCC/GND/VREF are used as the name of modules/instances/nets/ports	2992
CELL_INFO : Informations of all cell names and the number of each cell ...	2993
MODULE_PORTS : Information about number of ports in each module	2994
Timing Tags	2995
DeepMux : Deep MUX detected - potential performance problem	2996
ClockEnableRace : Reports a race condition between clock and enable pins of a flip-flop	3000
LogicDepth : Reports logic paths where depth exceeds the specified number of levels	3004
LogNMux : Reports multiplexer constructs with select inputs more than the specified number	3015
MaxFanout : Fan-out of a net exceeds a critical threshold	3021
ShiftReg : Large shift register detected.....	3023
Netlist Tags	3026
Rule_1-1 : Checks for the floating primary port	3027
Rule_1-2 : Checks if top level primary port is not connected to the pad cell	3028
Rule_1-3 : Checks if pad pin is connected properly	3030
Rule_1-4 : Checks if primary port is connected to multiple pad pins ..	3032
Rule_1-5 : Checks if inout pad pin is connected to a wrong inout primary port/pad pin.....	3034
Rule_2 : Detects open input terminals	3035
Rule_2-1 : Detects open inout terminal	3037

Rule_3 :	Reports variable/signals that do not infer a tristate and have multiple simultaneous drivers	3038
Rule_4 :	Checks if output pin of an instance is tied to VDD/GND	3042
Rule_5 :	Check pin voltage difference between source and destination.....	3045
Rule_8 :	Rule detects if transmission gate signal or instance are connected in series.	3049
Rule_9 :	Checks for incorrect parametric NAND connections	3051
Rule_12 :	Checks if output of a combinational cell is connected to its own input.....	3053
Rule_14 :	Checks if PI/POE/PO pins of parametric NAND tree are unconnected	3054
Rule_16 :	Checks if fan-in cone of set and reset pin of a flip-flop/latch converge	3056
Rule_19 :	Checks if an invalid cell is used in the analog path.....	3059
Rule_20 :	Checks if a hold fix delay cell is used in the clock/enable path...	3061
Rule_22 :	Checks if design has more than one parametric NAND trees	3063
Rule_23 :	Checks if control pin of an instance is tied to VDD or GND .	3065
Rule_24 :	Detects combinational loops	3066
Rule_25 :	Checks if gated/generated clocks are generated from more than one block in the design.....	3070
Rule_26 :	Checks if all the inputs of a cell are tied to VDD or GND	3072
Rule_26-1 :	Report library cell input pin's not connected to 'TIE' Cell.	3074
Rule_27 :	Checks if multiple inputs of the same cell are tied together	3076
Rule_28 :	Detects combinational loops, including loop through the set/reset and output pins of a flip-flop	3077
Rule_29 :	Checks if output data pin is connected to input data pin of the same storage element.....	3081
Rule_32 :	Reports variable/signals that do not infer a tristate and have multiple simultaneous drivers	3083
Rule_34 :	Detects undriven and unloaded nets/terminals in the design	3086
Rule_35 :	Detects undriven but loaded net in the design.....	3089
Rule_36 :	Checks if all outputs of an instance are unconnected.....	3092
Rule_41 :	Primary port of a block is tied to global signal	3093
Rule_42 :	Analog pin is tied to power or ground.	3095
Rule_43 :	Memory Efuse Pin should be directly connected to PadCell .	3099

CellInfo : List of cells used in the design	3103
CheckAnalogConnection : Report connection between analog and non-analog pin.....	3105
InoutPin : Inout pin of a cell should be directly connected to an inout port or another inout pin	3107
ModulePinExistence : Module has no input or output ports	3126
UnmixableCellCheck : Detecting unmixable standard cells in a module....	3128
ExternalPortAndIOPinCheck : Pad pin of I/O should be directly connected with external port	3131
ExternalPortBranchCheck : Branching of the net which is directly connected to an external port should be avoided	3132
InoutIOOutput : Output pin of inout cell has been connected to clock/reset/set/enable pin of a flop/latch	3134
InitialConnectionCheck : Detects the initial cell which does not connect directly to an external port	3140
HierarchicalPathLengthCheck : Too long hierarchical path name is not allowed.....	3142
SameNetConnectionCheck : Detects the circuit which affects delay calculation	3143
GlitchGenCircuitCheck : Detects incorrectly used glitch generation circuits	3145

Functional Lint Tags.....3149

W164a :	3151
W362 :	3153
W263 :	3155
W116 :	3157
W110 :	3158
STARC05-2.1.3.1 :	3160
STARC05-2.10.3.2a :	3162
STARC05-2.2.3.3 :	3164
sim_race02 :	3166
W415a :	3168
W71 :	3170
DuplicateCaseLabel-ML :	3172
SignedUnsignedExpr-ML :	3174
W467 :	3176

W551 :	3178
ImproperRangeIndex-ML :	3180
MissingFsmStateTransition : Reports violation for a missing state transitions, which are not possible in FSM	3182
NotReachableFsmState : Reports violation for not reachable state in FSM 3185	
NoExitFsmState : Reports violation if a state is reachable, and no other transition is possible in any of the clock cycle	3187
RegisterStuckInResetState-ML : Reports violation for the registers that are stuck in a constant initial state due to bad coding style	3189
DeadCode-ML : Reports redundant logic in the design.....	3191
MissingCaseItems-ML : Reports violations for reachable case items that are not specified.	3193

Preface

About This Book

The VC SpyGlass Lint Tags Reference Guide contains information on the parameters, reports, and Tags used in the VC SpyGlass Lint product.

Typographical Conventions

This document uses the following typographical conventions:

To indicate	Convention Used
Program code	OUT <= IN;
Object names	OUT
Variables representing objects names	<sig-name>
Message	Active low signal name '<sig-name>' must end with _X.
Message location	OUT <= IN;
Reworked example with message removed	OUT_X <= IN;
Important Information	NOTE: This Tag...

The following table describes the syntax used in this document:

Syntax	Description
[] (Square brackets)	An optional entry
{ } (Curly braces)	An entry that can be specified once or multiple times
(Vertical bar)	A list of choices out of which you can choose one
. . . (Horizontal ellipsis)	Other options that you can specify

Using VC SpyGlass Lint Product

The VC SpyGlass Lint product has the following usage features

- [VC SpyGlass Lint Parameters](#)
- [VC SpyGlass Lint Application Variables](#)
- [Product Reports](#)

VC SpyGlass Lint Parameters

This section provides detailed information on the VC SpyGlass Lint product Tag parameters.

You can set these parameters using the following syntax:

```
configure_lint_tag_parameter -tag <TAG-NAME> -parameter
<PARAM-NAME> -value <VALUE>
```

This section lists the Tag-specific parameters used in the VC SpyGlass Lint.

allowedOpPrecedence

Specifies the list of unary and binary operators for which the *ExprParen* tag will not report a violation.

Used by	<i>ExprParen</i>
Options	A list of unary and binary operators
Default value	" "

allowBuffer

(Optional) Specifies whether the *elementsAllowedPerScanChain* tag ignores buffers while traversing the scan chain.

By default, the `allowBuffer` parameter is set to `no` and the scan chain traversal stops at buffers. Set the value of the `allowBuffer` parameter to `yes` to allow buffers.

Used by	<i>elementsAllowedPerScanChain</i>
Options	yes, no
Default value	no

allowInverter

(Optional) Specifies whether the [elementsAllowedPerScanChain](#) tag ignores inverters while traversing the scan chain.

By default, the `allowInverter` parameter is set to `no` and the scan chain traversal stops at inverters. Set the value of the `allowInverter` parameter to `yes` to allow inverters.

Used by	elementsAllowedPerScanChain
Options	yes, no
Default value	no

allow_indirect_types

Specifies whether the [PortWithoutType-ML](#) tag reports a violation for signals that have indirect declared type definition.

By default, the `allow_indirect_types` parameter is set to `no` and the tag only allows cases where there is a direct mapping between the port net and its net type.

Set this parameter to `yes` to enable the tag to not report a violation for signals that have indirect declared type definition.

Used by	PortWithoutType-ML
Options	yes, no
Default value	no

allow_indirect_directions

Use this parameter to report violation for ports that are defined with indirect directions.

By default, the value of the parameter is `no`. In this case, the tag ignores ports that are defined with indirect directions.

Used by	PortWithoutDirection-ML

Options	yes, no
Default value	no

allow_tie_cell_list

Use this parameter to specify a list of perl regular expressions to specify list of Lib cells, which are required to be considered as a Tie cell.

Used by	Rule_26-1
Options	space-separated list of perl regular expressions
Default value	"EPTIELO.*,EPTIEHI.*,GPTIEHI.*,GPTIELO.*,TIEHI.*,TIELO.*"

allow_latch_cell

By default, the allow_latch_cell parameter is empty and not set. Set this parameter to comma-separated list of strings, to specify only those lib cell instances that shall be considered for violation to be flagged.

NOTE: Do not specify the ignore_latch_cell and allow_latch_cell parameters together. When used, the value of the allow_latch_cell parameter is discarded.

Used by	InferLatch
Options	comma-separated list of strings
Default value	' '

allow_module_list

Use this parameter to ignore modules whose name matches with the names specified using this parameter or starts with PD. Also supports regular expressions.

The default value is " ".

Used by	LINT_NONSTANDARD_LIB_USED
Options	comma-separated list of module names.
Default value	' '

allowedScanChainLength

(Optional) Specifies the maximum number of elements allowed in a scan chain for the [elementsAllowedPerScanChain](#) tag.

By default, the [elementsAllowedPerScanChain](#) tag flags scan chains of a single element. Set the value of `allowedScanChainLength` parameter to a positive integer number. Then, the [elementsAllowedPerScanChain](#) tag will flag scan chains that have more elements than the specified number.

Used by	elementsAllowedPerScanChain
Options	Positive integer value
Default value	1

ALLOWED_DRIVER_LIMIT

The default value of the parameter is 1024.

Set the value of the parameter to any integer value to specify the allowed driver limit.

Used by	MaxFaninLogic-ML
Options	positive integer value
Default value	1024

attach_violation_tag

Enables the *PragmaComments-ML* and *DuplicateCaseLabel-ML* tags to attach signature (tag) to the violation message to make the violation message unique.

The default value of the parameter is no. In this case, no tag is assigned to the violation message in the *PragmaComments-ML* and *DuplicateCaseLabel-ML* tags.

Set the value of the parameter to yes to attach signature (tag) to the violation message to make the violation message unique.

The tag is generated in an encrypted form and is hidden in spyglass textual reports, except spyglass_violations_report.xml.

When you waive violations through GUI, SpyGlass attaches signature to the waiver command (-signature {encrypted_word}).

You can generate custom waivers by referring signature in the spyglass_violations_report.xml file.

Used by	PragmaComments-ML , DuplicateCaseLabel-ML
Options	yes, no
Default value	no

apply_on_lib_cell_only

Use this parameter to enable the [UndrivenInTerm-ML](#) tag to check the LIB cells for undriven logic. The default value of the parameter is no. In this case, the UndrivenInTerm- ML tag does not checks only for the LIB cells for undriven logic.

Used by	UndrivenInTerm-ML
Options	yes, no
Default value	no

avoid_nba_always_latch

Specifies if the [SetBeforeRead-ML](#) tag reports violations for signals that are read in `always_latch` blocks, and set in non-blocking assignment before or after they are read.

By default, the value of the parameter is no and the tag reports violations for signals that are read in `always_latch` blocks, and set in

non-blocking assignment before or after they are read

Set the value of the parameter to `yes` to ignore violations for signals that are read in `always_latch` blocks, and set in non-blocking assignment before or after they are read.

Used by	SetBeforeRead-ML
Options	yes, no
Default value	no

avoid_seq_logic

Specifies whether the [W527](#) tag reports violation for combinational `always` blocks.

By default, the `avoid_seq_logic` parameter is set to `no`.

If you set this parameter to `yes`, the tag reports violation for combinational `always` blocks and does not report violation for sequential `always` blocks.

Used by	W527
Options	yes, no
Default value	no

avoid_synth_disable

Specifies whether the [W250](#) tag reports a violation for synthesizable disable constructs.

By default, the `avoid_synth_disable` parameter is set to `no`.

Set this parameter to `yes`, to enable the tag to not report violations for synthesizable disable constructs for the following cases:

- Tasks
- Blocks
 - ☐ Sequential blocks (whose start-end is defined by begin-end)

NOTE: For Parallel blocks (whose start-end is defined by fork-join), the tag does not waive violations as they are considered as non-synthesizable.

Used by	W250
Options	yes, no
Default value	no

bigmux_max

(Optional) Specifies the maximum number of levels in the nested if constructs beyond which the [DeepMux](#) tag should flag a message.

By default, the [DeepMux](#) tag flags nested if statements that are more than 4 levels deep. Use the `bigmux_max` tag parameter to specify a different integer number.

Used by	DeepMux
Options	Positive integer value
Default value	4

check_assign_pattern

Use this parameter to check assign pattern statements where the elements are assigned names or indexes.

Used by	W164a , W164b , W164a_a , W164a_b
Options	yes, no
Default value	no

case_option

Enables the [CheckKeywordsOfCaseStmt-ML](#) tag to check if unique/unique0/priority keyword is used with casez statement.

By default, the tag parameter is set to any.

Use this parameter to specify the keyword that must be used with `casez` statements.

Used by	CheckKeywordsOfCaseStmt-ML
Options	one of <code>unique</code> , <code>unique0</code> , <code>priority</code> , <code>any</code>
Default value	<code>any</code>

casez_option

Enables the [CheckKeywordsOfCaseStmt-ML](#) tag to check if `unique`/`unique0`/`priority` keyword is used with `casez` statement.

By default, the tag parameter is set to `any`.

Use this parameter to specify the keyword that must be used with `casez` statements.

Used by	CheckKeywordsOfCaseStmt-ML
Options	one of <code>unique</code> , <code>unique0</code> , <code>priority</code> , <code>any</code>
Default value	<code>any</code>

casez_option

Enables the [CheckKeywordsOfCaseStmt-ML](#) tag to check if `unique`/`unique0`/`priority` keyword is used with `casez` statement.

By default, the tag parameter is set to `any`.

Use this parameter to specify the keyword that must be used with `casez` statements.

Used by	CheckKeywordsOfCaseStmt-ML
Options	one of <code>unique</code> , <code>unique0</code> , <code>priority</code> , <code>any</code>
Default value	<code>any</code>

case_inside_option

Enables the [CheckKeywordsOfCaseStmt-ML](#) tag to check if `unique/unique0/priority` keyword is used with `case inside` statement.

By default, the tag parameter is set to `any`.

Use this parameter to specify the keyword that must be used with `case inside` statements.

Used by	CheckKeywordsOfCaseStmt-ML
Options	one of <code>unique,unique0,priority,any</code>
Default value	<code>any</code>

check_all_modules

Specifies whether the [CheckPortType-ML](#) and [OneModule-ML](#) tags should check for complex port types on all module levels.

By default, the `check_all_modules` parameter is set to `no` and the [CheckPortType-ML](#) rule checks for complex port types only at the top module interfaces.

Set the `check_all_modules` parameter to `yes` to enable the [CheckPortType-ML](#) rule to check on all the module levels.

Used by	CheckPortType-ML , OneModule-ML
Options	<code>yes, no</code>
Default value	<code>no</code>

check_unread

Enables the `W495` tag to report violation for inout ports that are unread and not set.

The default value of the parameter is `no`. In this case, the tag reports violation when inout port is read but not set.

Used by	W495
Options	yes, no
Default value	no

check_below_32

Reports violations (in addition to reporting violations to the default behaviour) if the:

- Default width of the sized parameters is less than or equal to 32 bits.
- Parameter is overridden by an unsized based number, where the width of the unsized based number is greater than the default width.

The Default value for the parameter is no.

Used by	UseParamWidthInOverriding-ML
Options	yes, no
Default value	no

check_bit_casting

By default, the [SelfDeterminedExpr-ML](#) tag does not check the self-determined expressions in the bit cast operation.

Set the value of the `check_bit_casting` parameter to `yes`, to check for bit casting and report violations when the size of the bit cast is not sufficient to retain the result of the expression.

Used by	SelfDeterminedExpr-ML
Options	yes, no
Default value	no

check_directive_type

Specifies whether the [UnsuppCompDir-ML](#) tag reports a violation for the use of ``ifdef`, ``resetall`, ``default_nettype` and ``unconnected_drive` compiler directives.

By default, the parameter set to "".

Set the value of the `check_directive_type` parameter to one or a comma/ space-separated list of compiler directives to enable the rule to report a violation for the use of ``ifdef`, ``resetall`, ``default_nettype` and ``unconnected_drive` compiler directives.

Used by	UnsuppCompDir-ML
Options	one or comma/ space - separated list of "ifdef", "resetall", "default_nettype", "unconnected_drive"
Default value	" "

check_escaped_names

Checks for escaped identifier names with non-escaped names.

By default, the value of the parameter is set to no, and tag does not check escaped identifier names with non-escaped names.

Set the value of the parameter to yes to check escaped identifier names with non-escaped names, and report violations for same cased names. For example, when the value of the parameter is set to yes, the `ConsCase` tag reports violation for the following:

```
A[0] , \A[0]  
A[0] , \a[0] - violation
```

Used by	ConsCase
Options	yes, no
Default value	no

check_enum_usage

Specifies whether the [TwoStateData-ML](#) tag should report a violation for 2-state enums where they are used as datatypes.

By default, the parameter is set to no.

Set the `check_enum_usage` parameter to yes to enable the tag to report a violation for 2-state enums where they are used as datatypes. The tag does not report a violation on enum datatype usages for parameters or localparameters. The `check_enum_usage` parameter is ineffective if the `check_four_state_assign` parameter is set to yes.

Used by	TwoStateData-ML
Options	yes, no
Default value	no

check_globally_defined

Enables the [DirectiveCheck-ML](#) tag to report a separate violation when the macro is defined in the project file.

By default, the value of the parameter is set to no and the tag reports a violation saying macro is not defined irrespective of if the macro is defined or not in the project file.

Set the value of the parameter to yes to report a violation saying macro is defined in the project file when it is defined in the project file.

Used by	DirectiveCheck-ML
Options	yes, no
Default value	no

check_include_files

Enables the tags to report violations if the macro is defined in an include file and is referred outside the module. The tag does not report violations if the file is included in the module scope.

Used by	STARC05-1.1.4.4
Options	yes, no
Default value	no

check_inst_connections

Enables the [ParamWidthMismatch-ML](#)-ML tag to report violations if:

- Natural width of the parameter connection is less than its LRM width and there is value truncation.
- Natural width of the parameter connection is greater than the defined width of the parameter.

By default, the value of the parameter is set to no.

Set the value of the parameter to yes to report violations for the above mentioned cases.

In addition, the default value of the parameter is not checked if it is overridden.

Used by	ParamWidthMismatch-ML
Options	yes, no
Default value	no

check_input_port

Use this parameter to disable violation on output port for the specified tag.

The default value of the parameter is no for the [UndrivenNUnloaded-ML](#) tag and yes for the [Rule_34](#) tag.

For [UndrivenNUnloaded-ML](#), set the value of the parameter to yes to disable violation on output ports.

For [Rule_34](#), set the value of the parameter to no to enable violation on output ports.

Used by	UndrivenNUnloaded-ML , Rule_34

Options	<i>yes, no</i>
Default value	no (UndrivenNUnloaded-ML) yes (Rule_34)

check_lrm_and_natural_width

Enables the [W164a](#) and [W164b](#) tags to calculate both LRM and natural width before reporting violation for the assignment statement. If any of these width matches with RHS, then no violation is reported for the assignment.

Used by	W164a , W164b
Options	<i>yes, no</i>
Default value	no

check_loop_breakers

Enables the [CombLoop](#) rule to check if the AND, OR, and 2-bit MUX instances in the path are mutually exclusive loops. If a loop is mutually exclusive, the no violation is reported for that loop.

Example:

```
module comb_loop (input sel);
    wire and2;
    wire whatever;
    wire mux = sel ? and2 : whatever;
    assign and2 = ~sel & mux;
endmodule
```

Used by	CombLoop ,
Options	<i>yes, no</i>
Default value	<i>no</i>

check_natural_width_for_static

Enables the [W164a](#) tag to not report violations for static LHS depending on the [check_static_value](#) parameter.

Used by	W164a
Options	<i>yes, no</i>
Default value	<i>no</i>

check_natural_width_of_div

Use this parameter to calculate the natural width of a division operator when its divisor is static, regardless of the value of the `nocheckoverflow` parameter. Natural width of such expression is also calculated according to the max value it can hold.

Used by	W164a , W164b and ImproperRangeIndex-ML
Options	yes, no
Default value	no

check_natural_width_of_modulo

Use this parameter to calculate the natural width of a modulo operator, regardless of the value of the `nocheckoverflow` parameter. Natural width of such expression is also calculated according to the max value it can hold.

Used by	W164a , W164b and ImproperRangeIndex-ML
Options	yes, no
Default value	no

check_natural_width_of_multiplication

Enables the [W164a](#) and [W164b](#) tags to calculate the width of multiplication expressions. Set the parameter to yes, to calculate the width of the multiplication expressions as per the natural width even if the `nocheckoverflow` parameter is set to yes.

Set the parameter to a comma/ space-separated list of tags to enable (the parameter) for the tags specified in the list.

Used by	W164a , W164b
Options	yes, no, comma-separated list of tags
Default value	no

check_non_static_index

Enables the [NegativeIndex-ML](#) tag to report violations for non-static bit expressions.

By default, the `check_non_static_index` parameter is set to no.

Set the value of this parameter to yes to enable the NegativeIndex-ML tag to report violations for non-static bit expressions.

The NegativeIndex-ML tag evaluates the minimum value for expression with arithmetic operators (- , + , * , /) and reports a violation if the minimum value is negative.

For non-static sub-expressions, the minimum value is considered as 0 and the maximum value is $2^{(\text{width})} - 1$.

In subtractions, when both the operands are non-static and the left operand width is greater than the right operand, the minimum value is considered as 0 if it is negative (to avoid noise violations).

The [NegativeIndex-ML](#) tag will unroll for loops and the for loop index is evaluated as static.

Used by	NegativeIndex-ML
Options	yes, no
Default value	no

check_only_missing_default

Enables the [STARC05-2.8.1.4](#) tag to report violations for case constructs only when the default clause is missing in sequential blocks.

By default, the value of the parameter is set to no. In this case, the tag reports a violation for all signals that are not assigned in all branches of the case construct.

Set the value of the parameter to yes to report violations for case constructs only when the default clause is missing in sequential blocks.

Used by	STARC05-2.8.1.4

Options	yes, no
Default value	no

check_only_top_module

Use this parameter to enable the [UnloadedInPort-ML](#) tag to report violation for only top-level port.

By default, the value of the parameter is no. In this case, the tag reports violation for all driven but unloaded ports of an input port.

Used by	UnloadedInPort-ML
Options	yes, no
Default value	no

check_parameter

Enables the [W362](#) tag to report violations if one of the operands is a parameter/localparam and the other operand is non-static, even if the value of the `check_static_value` parameter is set to no.

By default, the value of the parameter is set to no. In this case, the W362 tags does not report violation if one of the operands is a parameter/localparam and the other operand is non-static.

Set the value of the parameter to yes to report violation for such cases.

Used by	W362
Options	yes, no, comma separated list of tags
Default value	yes

check_static_natural_width

Enables the specified tag to check for natural width of a static expression even when the value of the [nocheckoverflow](#) parameter is set to yes.

Used By	W164a , W164b
Options	yes, no
Default Value	no

check_task_ports

Specifies whether the [STARC05-1.1.1.2](#) tag should report violations for the input and output of tasks.

By default, the value of the parameter is no.

Set the value of the parameter to yes to report violations for the input and output of tasks.

Used By	STARC05-1.1.1.2
Options	yes, no
Default Value	no

checkinalldu

Specifies whether the [NoInoutPort-ML](#) tag checks for inout ports present in all design units or at the hierarchy level specified by the [inoutlevel](#) tag parameter.

By default, the `checkinalldu` tag parameter is set to no and the `NoInoutPort-ML` tag checks for inout ports at the level specified by the `inoutlevel` tag parameter.

Set the `checkinalldu` tag parameter to yes, and `NoInoutPort-ML` tag checks for inout ports present in all design units.

Used by	NoInoutPort-ML
Options	yes, no
Default value	no

checkpwrpin

(Optional) Enables the [pwrpinConnToSeqOrIOCells](#) tag.

By default, the [pwrpinConnToSeqOrIOCells](#) tag is disabled.

Used by	pwrpinConnToSeqOrIOCells
Options	yes, no
Default value	no

checkRTLInst

Specifies whether the [UndrivenInTerm-ML](#) tag checks for undriven instances generated by SpyGlass for RTL assignment descriptions.

By default, the value of the `checkRTLInst` tag parameter is set to `no`, and the `UndrivenInTerm-ML` tag does not check for undriven instances generated by SpyGlass for RTL assignment descriptions.

Set the value of the `checkRTLInst` tag parameter to `yes` to check for such instances.

Used by	UndrivenInTerm-ML
Options	yes, no
Default value	no
Default Value in GuideWare2.0	yes

check_top_ports

Specifies whether the [InterfaceWithoutModport-ML](#) tag should report a violation when the interface typed ports are used without modports in the module definition.

By default, the value of the `check_top_ports` parameter is set to `no`.

Set the value of the `check_top_ports` parameter to `yes` to report a violation when the interface typed ports are used without modports in the module definition.

Used by	InterfaceWithoutModport-ML
Options	yes, no
Default value	no

checkv2kparamsonly

Specifies whether the specified tag should consider only the parameters declared in v2k style, that is, the parameters that are declared in the module header.

By default, the `checkv2kparamsonly` Tag parameter is set to `no`.

Set the `checkv2kparamsonly` Tag parameter is set to `yes`, and the specified tag considers only the parameters declared in v2k style, that is, the parameters that are declared in the module header.

Used by	ParamOverrideMismatch-ML , PragmaComments-ML
Options	yes, no
Default value	no

checkInstName

Enables the [NoGenLabel-ML](#) tag to report violation for generate block whose name matches the value specified using the parameter.

The default value of the parameter is `" "`.

Set the value of the parameter to a regular expression to enable the [NoGenLabel-ML](#) tag to report violation for generate block whose name matches the value specified using the parameter.

Used by	NoGenLabel-ML
Options	PERL Regular Expression
Default value	<code>" "</code>

clockgate_filename

The `clockgate_filename` parameter is used to specify the file that contains library names whose cell can be present in the clock tree. Following convention should be used to specify the libraries and the cells in the file specified using the `clockgate_filename` parameter:

```
Library: library_name1
cell_name1
cell_name2
...
Library: library_name2
cell_name1
...
```

Library names and cell names can be specified using shell wildcard pattern.

Used by	CGFromSpecifiedLib-ML , CGFromSpecifiedCell-ML
Options	The name of the file containing library names
Default value	null.txt

CLOCKNET

Use this parameter to read the clocks specified using the SGDC or SDC commands. Specify the value of the clock SGDC command as an input to this parameter (use space-separated values in case of multiple commands).

For example, for the following SGDC command:

```
clock -name top.reset top.mid1.rst
```

Specify the `CLOCKNET` parameter as shown below:

```
configure_lint_tag_parameter -tag
clockPinsConnectedToClkNets -parameter CLOCKNET -value {
top.clock top.mid1.rst2 } -goal <goal_name>
```

Used by	clockPinsConnectedToClkNets

Options	space-separated list of clock pin names
Default value	" "

clk_rst_net

Use this parameter to specify a space-separated list of clock/reset nets. The tag will not report a violation for the external port if its branch reaches to clock/reset net. The parameter also supports regex.

Used by	ExternalPortBranchCheck
Options	Space-separated list of strings/reg-exp
Default value	""

CONSIDER_SUPPLY_AS_DRIVER

Use this parameter to enable the [UnloadedNet-ML](#) rule to consider constant/supply as driver for a net.

The default value of the parameter is no.

Used by	UnloadedNet-ML
Options	yes, no
Default value	no

comment_acc_max_length

Specifies the maximum value for the accumulated comment length in a line where both RTL and comments are present. Enables the [LineLength](#) tag to report a violation if the comment length is greater than the specified maximum comment length.

Used by	LineLength
Options	yes, no
Default value	no

comblock_waiver_compat

Specifies whether the [Comblock](#) tag reports closest net name to fanout connection point and rearranges the loop order to get the same starting point for IP and SOC level.

By default, the `comblock_waiver_compat` tag parameter is set to `no`.

You can set the value of the `comblock_waiver_compat` tag parameter to `yes` to report closest net name to fanout connection point and rearrange the loop order to get the same starting point for IP and SOC level.

Used by	Comblock
Options	<code>yes</code> , <code>no</code>
Default value	<code>no</code>

concat_size_based_width

Enables the [ImproperRangeIndex-ML](#) tag to consider the specified width of sized-based numbers in non-static concatenation expressions without considering their value.

By default, the value of the parameter is `no`. In this case, the [ImproperRangeIndex-ML](#) tag ignores leading zeros while calculating the width of non-static concatenation expressions.

Set the value of the parameter to `yes` to consider the specified width of sized-based numbers in non-static concatenation expressions without considering their value. For example, consider the following code snippets:

■ `{1'b0 , c[3:0]}`

In this case, the width is 5, when the parameter is set to `yes`. (1 bit + 4 bit)
`{1'b0 , c[3:0]}`

In this case, the width is 4, when the parameter is set to `no`. (Maximum possible value for this expression is 15, therefore bit width is 4)

Used by	ImproperRangeIndex-ML

Options	yes, no
Default value	no

design_attributes

By default, the [STARC05-1.1.1.5](#) tag reports a violation for all definitions of the same names that differ in case, within the same scope.

You can specify a set of constructor types so that the tag considers only those names of selected construct types.

Used by	STARC05-1.1.1.5
Options	one or more from signal, port, parameter, instance, module, function, task, named-block
Default value	""

disable_rtl_deadcode_extended

Use this parameter to not report violations for dead codes.

By default, the value of the parameter is no. In this case, the tag reports violation for dead codes.

Set the value of the parameter to yes to not report violation for dead codes.

Used by	NeverExecutingForLoop-ML
Options	yes, no
Default value	no

disable_seq_clock_prop

Use the `disable_seq_clock_prop` parameter to prevent automatic propagation of clocks through flops.

VC SpyGlass Lint Parameters

Used by	CGFromSpecifiedLib-ML , CGFromSpecifiedCell-ML
Options	yes, no
Default value	no

dw_name

Specifies the DesignWare component naming convention as a regular expression for the [DisallowDWComp-ML](#) tag.

By default, the dw_name Tag parameter is set to the following regular expression:

```
DW.*mult.*|DW.*mac.*|DW.*prod.*|DW.*div.*|DW.*sqrt.*|DW.*square.*
```

You can set the dw_name Tag parameter to any valid regular expression.

Used by	DisallowDWComp-ML
Options	Valid regular expression
Default value	DW.*mult.* DW.*mac.* DW.*prod.* DW.*div.* DW.*sqrt.* DW.*square.*

enable_rtl_deadcode

Specifies whether to report different violations for dead codes in the RTL resulting from if/for/while and ternary operators that are ignored when the disable_rtl_deadcode parameter is enabled.

By default, this parameter is set to no.

Set this parameter to yes to report different violations for dead codes in the RTL resulting from if/for/while and ternary operators that are ignored when the disable_rtl_deadcode parameter is enabled.

Used By	W164a_a , W164a_b , W164b , W116 , W362 , W416 , W486 , STARCO5-2.10.3.2a
Options	yes, no, <rule-name>
Default Value	no

encoding_type

Specifies whether the [UseDefine](#) tag defines the encoding types for finite state machines (FSM).

The default value of the parameter is parameter, macro.

Set the value of the parameter to one or more from the comma-separated list of parameter, macro, enum tags, to define the encoding types for finite state machines (FSM). This parameter is only applicable to Verilog and supports the encoding parameter, macro, enum styles.

Used By	UseDefine
Options	one or more from the comma-separated list of parameter, macro, enum tags
Default Value	parameter, macro

excluded_check_port

Use this parameter to specify a space-separated list of ports names for which tag checking should not be done. The parameter also supports regex.

Used By	ExternalPortBranchCheck
Options	space-separated list of strings/reg-exp
Default Value	" "

flag_only_instance_ports

Enables the tag to report violation for module output ports that are connected to instance input pin.

Used By	W446
Options	yes, no
Default Value	no

fanout_factor

(Optional) Specifies the scale factor for raw fan-out calculated by the [MaxFanout](#) tag.

As the raw fan-out computed by VC SpyGlass Lint does not make allowance for optimization in synthesis, the raw fan-out value calculated by the *MaxFanout* tag needs to be scaled by a value typically less than 1, to derive a more accurate estimate.

As the latches or flip-flops would not be optimized away by a synthesis tool, this scaling factor is not applied on the portion of fan-out due to latches and flip-flops.

By default, the *MaxFanout* tag uses a scaling factor of 0.5. Use the `fanout_factor` parameter to specify your own scaling factor.

Used by	MaxFanout
Options	Floating number (typically less than 1)
Default value	0.5

fanoutmax

(Optional) Specifies the maximum number of fan-out elements beyond which the [MaxFanout](#) tag flags nets as large fan-out nets.

By default, the *MaxFanout* tag flags nets with fan-out of more than 20. Use the `fanoutmax` parameter to specify a different integer number.

Used by	MaxFanout
Options	Positive integer value
Default value	20

file_hdr_ignore_path

Enables the [FileHdr](#) tag to ignore violations for a specified list of file paths.

By default, the *FileHdr* tag will report for all files, which do not have a proper file header.

Used by	FileHdr
Options	Comma or space-separated list of perl regular expressions
Default value	" "

filter_mark_open

Use this parameter to not report violations on those port connections that have been explicitly left open in the RTL.

By default, the value of the parameter is no.

Used by	W287a , W287b
Options	yes, no
Default value	no

generate_connpoint_report

Specifies whether the [FlopEConst](#) and [FlopClockConstant](#) tags generate a report that contains the connection point information of the constant source.

By default, the `generate_connpoint_report` parameter is set to no and the above tags do not generate the report.

If you set this parameter to yes, the specifies tags generate a report that contains the connection point information of the constant source.

Used by	FlopEConst , FlopClockConstant
Options	yes, no
Default value	no

REPORT_D_FF_COUNT

The default value of the parameter is no. In this case the [InferFF](#) tag reports all inferred flops in the design.

VC SpyGlass Lint Parameters

Set the value of the parameter to yes to report only the number of D flip-flops in the design.

Set the value of the parameter to all to report all to report violations for all inferred flops as well as the count of D flipflops.

Used by	InferFF
Options	yes, no, all
Default value	no

REPORT_ALL_DRIVERS

The default value is no. Set the value of the parameter to yes to report all valid drivers.

Used by	MaxFaninLogic-ML
Options	yes, no
Default value	no

REPORT_ALL_POSEDGE_NEGEDGE_PAIRS

This parameter specifies that the [STARC05-1.4.3.1c](#) tag will report all pairs of violation between posedge and negedge modules. By default, the value of this parameter is 1. Set the value of the parameter to 0 to disable reporting violation between posedge and negedge modules.

Used by	STARC05-1.4.3.1c
Options	0, 1
Default value	1

REPORT_FILE_LINE_ON_SOURCE

This parameter specifies that the [STARC05-1.3.1.3](#), [STARC05-1.4.3.4](#), and

[W392](#) tags will report violation on the back reference on RTL data net of fanout flop. By default the value of this parameter is 0. Set the value of the parameter to 1 to report violations on back reference on RTL data net of fanout flop.

Used by	STARC05-1.3.1.3 , STARC05-1.4.3.4 , W392
Options	0, 1
Default value	0

report_gated_clock_name

This parameter specifies that the violation messages will contain the gated clock source signal name. The default value of this parameter is 1. Set the value of the parameter to 0 to disable reporting of the gated clock source signal name in the violation message.

Used by	LatchGatedClock
Options	0, 1
Default value	1

REPORT_ONLY_USER_NETS

This parameter specifies that the [STARC05-1.3.2.1b](#) and [GateResetAtTop](#) tags will not report violations when the reset source is not user defined RTL net. The default value of this parameter is 1. Set the value of the parameter to 0 to enable reporting violations.

Used by	STARC05-1.3.2.1b , GateResetAtTop
Options	0, 1
Default value	1

REPORT_UNDRIVEN_RESET

This parameter enables the [STARC05-1.3.1.3](#) tag to report violations for the undriven reset source. The default value is yes. Set the value of the parameter to no to disable detecting the undriven reset source.

Used by	STARC05-1.3.1.3
Options	yes, no
Default value	yes

REPORT_NONUSERMUX

This parameter enables the [STARC05-1.3.1.3](#) and [STARC05-1.4.3.4](#) tags to report violations when a non-user defined RTL mux as non-reset and non-clock usage. The default value is yes. Set the value of the parameter to no to disable reporting violations.

Used by	STARC05-1.3.1.3 , STARC05-1.4.3.4
Options	yes, no
Default value	yes

REPORT_UNDRIVEN_CLOCK

This parameter enables the [STARC05-1.4.3.4](#) tag to report violations for the undriven clock source. By default, the value of the parameter is yes. Set the value of the parameter to no to avoid reporting violations.

Used by	STARC05-1.4.3.4
Options	yes, no
Default value	yes

REPORT_UNUSEDFLOP_COUNT

This parameter enables the [UnUsedFlopOutput-ML](#) tag to report number of unused D flip-flops in the design.

By default, the value of the parameter is no. In this case, the tag reports a violation for unreachable sequential elements.

Set the value of the parameter to yes to report number of unused D flip-flops in the design, in addition to, the violation for unreachable sequential elements. Also, the [UnusedFlopOutput-ML](#) tag generates the UnusedFlopsRpt.rpt report, which lists all the unused flip-flops.

Used by	UnusedFlopOutput-ML
Options	yes, no
Default value	no

SKIP_MODULE

The default value is NULL. Set the value of the parameter to a comma-separated list of module names to specify the modules for which you want to skip checks.

Used by	MaxFaninLogic-ML
Options	Comma-separated list of module names
Default value	NULL

traverse_through_pad_pin

This parameter enables the [LatchGatedClock](#) tag to stop the fanin traversal through pin of pad cells while finding clock sources. By default, the value of the parameter is 0. Set the value of the parameter to 1 to stop the fanin traversal.

Used by	LatchGatedClock
Options	0,1
Default value	0

traverse_fanin_for_src_net

This parameter specifies that the [STARC05-1.3.1.6](#) and [W448](#) tags will not traverse in fanin of synchronous reset to detect flattened source reset. By default, the value of the parameter is 0. Set the value of the parameter to 1 to enable the fanin of synchronous reset to detect flattened source reset.

Used by	STARC05-1.3.1.6 , W448
Options	0,1
Default value	0

ignore_nets

(Optional) Specifies the nets to be ignored for the specified tag.

By default, the *MaxFanout* tag checks all nets in the design. However, you can ignore clock nets and reset nets.

To ignore nets for checking, specify the net names as full hierarchical names or just the names excluding the top-level design unit name as a comma- or space-separated list with the `ignore_nets` parameters.

You can also use regular expressions to specify the net names.

By default, the `ignore_nets` parameter takes hierarchical names excluding the top-level module name. If [infertop](#) parameter is set to *no*, specify the full hierarchical name and include the top-module name.

Used by	MaxFanout , UndrivenNUnloaded-ML
Options	A comma- or space-separated list of nets
Default value	Hierarchical names excluding the top-level module name

infertop

Enables specifying net names with the [ignore_nets](#) parameter without including the name of top-level design unit for net hierarchical names as used by the [MaxFanout](#) tag.

By default, the `infertop` parameter is set to *yes*. You may change the

default value by setting the `infertop` parameter to `no`.

If the `infertop` parameter is set to `yes`, the top-level design unit name should be omitted from the net names specified with the [ignore_nets](#) parameter.

If you want to specify the full hierarchical names including top module name, set the `infertop` parameter to `no`.

Used by	MaxFanout
Options	yes, no
Default value	yes

IGNORE_LATCH

The default value is NULL. Set the value of the parameters to a comma-separated list of latch names that you want to ignore during traversal.

Used by	MaxFaninLogic-ML
Options	Comma-separated list of latch names
Default value	NULL

instance_based_pin_names

Enables the [PortConnectedToMux-ML](#) tag to identify the ports of module, which need to be analyzed by the tag.

By default, the value of the parameter is " ".

Specify a comma-separated hierarchical instance-based pin list to identify the pins which needs to be analyzed by the tag.

The hierarchical separator can be either '.' or '/'.

Used by	PortConnectedToMux-ML
Options	Comma-separated hierarchical instance based pin list.
Default value	" "

instance_name_netlist

Specifies the regular expression pattern for instance names in a netlist design as checked by the [InstNameStyle-ML](#) tag.

By default, the `instance_name_netlist` tag parameter is set to the following:

```
/ (^ [a-zA-Z] \ $ ) | ( ^ [a-zA-Z] [a-zA-Z0-9_]* [a-zA-Z0-9] \ $ ) /
```

The default value of this parameter ensures the following:

- Name should begin with an alphabet.
- Name should not end with an underscore ('_') and contain only [a-zA-Z0-9_].

You can specify your own naming convention by specifying a different Perl regular expression pattern for instance names in a netlist design.

Used by	InstNameStyle-ML
Options	A Perl regular expression
Default value	<code>/ (^ [a-zA-Z] \ \$) (^ [a-zA-Z] [a-zA-Z0-9_]* [a-zA-Z0-9] \ \$) /</code>

ignore_nonstatic_counter

Enables the W116 tag to ignore violations ignore violations for non-static single-bit argument in additions and subtractions.

By default, the value of the parameter is set to no.

Set the value of the parameter to yes to ignore violations for non-static single-bit argument in additions and subtractions.

Used by	W116
Options	yes, no
Default value	no

instance_prefix

Specifies a regular expression pattern for instance name prefix as checked by the [InstNameStyle-ML](#) tag (VHDL).

By default, the `instance_prefix` tag parameter is set to the following:

`"$legal_prefix"`

Thus, the tag flags those instance names that do not follow this naming convention.

You can set the `instance_prefix` tag parameter to any valid Perl regular expression.

Used by	InstNameStyle-ML
Options	A Perl regular expression
Default value	<code>\$legal_prefix</code>

instance_suffix

Specifies a regular expression pattern for instance name suffix as checked by the [InstNameStyle-ML](#) tag (VHDL).

By default, the `instance_suffix` tag parameter is set to the following:

`"_[0-9]+"`

Thus, the tag reports the instance names that do not follow this naming convention.

You can set the `instance_suffix` tag parameter to any valid Perl regular expression.

Used by	InstNameStyle-ML
Options	A Perl regular expression
Default value	<code>"_[0-9]+"</code>

allow_types

By default, the [DisallowUnknownInCase-ML](#) tag reports violations for usage of 'x', 'z' and '?'. Set the value of the parameter to one or comma separated

list of 'x', 'z' and '?' to ignore reporting violations for usage of given types.

Used by	"DisallowUnknownInCase-ML"
Options	Name and gate count of black box modules
Default value	Empty string

report_only_libcell

Enables the [checkIOPinConnectedToNet](#) tag to report only lib-cells. The *checkIOPinConnectedToNet* tag also checks whether the input pins of a lib-cell is floating or not while traversing through buffers and inverters only. Also, if it's connected to a top port, it is considered as a driving net.

By default, the value of the parameter is no. In this case, the tag checks all the cell instances.

Set the value of the parameter to yes to report only lib-cells.

Used by	checkIOPinConnectedToNet
Options	yes, no
Default value	no

ignore_async_blocks

Specifies whether the [DiffDelayInNonBlock-ML](#) tag should report a violation for asynchronous blocks.

By default, the value of this parameter is set to no and the tag reports a violation for asynchronous blocks within the whole always block.

Set the value of this parameter to yes to not report a violation for asynchronous blocks.

Used by	DiffDelayInNonBlock-ML
Options	yes, no
Default value	no

ignore_auto_function

Use this parameter to ignore rule checking for automatic functions.

Used by	STARC05-2.9.1.2e
Options	yes, no
Default value	no

IGNORE_SAME_SUPPLY_SOURCE

Use this parameter to report only one violation per directly connected supply source. Here, directly means there should not be any comb logic between supply and flop/latch.

By default, the value of the parameter is set to yes.

Set the value of the parameter to no to enable the affected rules to report all the violations directly connected to the supply source.

NOTE: *The default value of the parameter is yes. However, under the SG_VC_Compat switch, the default value is no.*

Used by	FlopClockConstant , FlopDataConstant , LatchDataConstant , FlopSRConst , FlopEConst , DisabledAnd , DisabledOr , MuxSelConst , TristateConst , LatchEnableConstant
Options	yes, no
Default value	no

IGNORE_SAME_DRIVERS

Use this parameter to ignore drivers of same type.

By default, the value of the parameter is set to no. In this case, the [STARC05-2.5.1.5a](#) tag considers drivers of same type for rule checking.

Used by	STARC05-2.5.1.5a

Options	yes, no
Default value	no

ignore_blocking_hanging_flop

Specifies whether the [STARC05-2.3.2.1](#) and [sim_race03](#) tags should ignore hanging flip-flops.

By default this parameter is set to `no`. Set this parameter to `yes` to enable the tag to ignore hanging flip-flops.

Used by	STARC05-2.3.2.1 , sim_race03
Options	yes, no
Default value	no

ignore_case_compare_op

Enables the STARC05-2.10.1.4b tag to ignore violations for case equality comparisons (`==`, `!=`).

By default, the value of the `ignore_case_compare_op` parameter is set to `no` and the tag reports all types of signal comparisons with values containing X (unknown) or Z (high impedance).

Set this parameter to `yes` to enable the tag to not report violations for case equality comparisons (`==`, `!=`).

Used by	STARC05-2.10.1.4b
Options	yes, no
Default value	no

ignore_clk_generator

Use this parameter to ignore violations for clock generator instances.

By default, the value of the parameter is `no`. In this case, the rule reports violations for the clock generator instances.

Used by	STARCO5-1.4.3.4
Options	yes, no
Default value	no

ignore_if_case_statement

Use this parameter to not report violation for signals, which are inside if/else or case statement.

By default, the value of the parameter is set to no.

Used by	W415a
Options	yes, no
Default value	no

ignore_chain_cell

Specifies whether the [DisabledAnd](#) and [DisabledOr](#) Tags report violations for cascading chain of cells.

By default, the `ignore_chain_cell` Tag parameter is set to no and the above Tags report violations for the cascading chain of cells.

If you set this parameter to yes, these Tags do not report violations for cascading cells.

Used by	DisabledAnd , DisabledOr
Options	yes, no, comma or space separated list of Tags
Default value	no

ignore_chain_cell

Specifies whether the [DisabledAnd](#) and [DisabledOr](#) Tags report violations for cascading chain of cells.

By default, the `ignore_chain_cell` Tag parameter is set to no and the above Tags report violations for the cascading chain of cells.

If you set this parameter to `yes`, these Tags do not report violations for cascading cells.

Used by	DisabledAnd , DisabledOr
Options	yes, no, comma or space separated list of Tags
Default value	no

ignore_const_zero_only

Specifies if the [STARCO5-1.3.1.7](#) tag should ignore violations if the set or reset pin is connected to only constant zero. It will not ignore violation when pin is connected to constant one.

By default, this parameter is set to `yes`

Set the parameter to `yes` to enable the tag to not ignore a violation.

Used by	STARCO5-1.3.1.7
Options	yes, no
Default value	yes

ignore_hanging_flop

Specifies whether the specified tags do not report violations if the output of the flip-flop/latch is hanging.

By default, the `ignore_hanging_flop` Tag parameter is set to `no` and the mentioned tags report violations if the output of the flip-flop/latch is hanging.

Set the value of the parameter to `yes` to ignore violations for the flip-flop/latch whose output is hanging.

Used by	FlopEConst , FlopSRConst , FlopClockConstant , ClockEnableRace , SetResetConverge-ML , IntReset , Rule_16
Options	yes, no, comma-separated list of Tags
Default value	no

ignore_integer

Use this parameter to ignore violations on integer data type.

By default, the value of the parameter is no. In this case, the W224 tag reports violation for integer data type.

Used by	W224
Options	yes, no
Default value	no

ignore_latch_cell

By default, the ignore_latch_cell parameter is empty and not set . Set this parameter to comma-separated list of string, to specify lib cell instances that need to be ignored.

NOTE: Do not specify the ignore_latch_cell and allow_latch_cell parameters together. When used, the value of the allow_latch_cell parameter is discarded.

Used by	InferLatch
Options	comma-separated list of strings
Default value	' '

ignore_libcell_inst

Enables the [NonOverridenParams-ML](#) tag to ignore violations for library cell instances.

The default value is no. Set the value of the parameter to yes to ignore violations for master library cell instances.

Used by	NonOverridenParams-ML
Options	yes, no
Default value	no

ignore_libcell_traversal

By default, the [FlopSRConst](#) Tag considers the flip-flops under library cell. Set the value of the parameter to yes to ignore the flip-flops under library cell.

Used by	FlopSRConst
Options	yes, no
Default value	no

ignore_out_term

Specifies whether the [UnloadedOutTerm-ML](#) tag should report a violation for specified terminals.

By default, the value of the parameter is set to "" and the tag reports a violation for all the output terminals of an instance that does not have a load but has a driver.

Set the value of this parameter to a comma or space-separated list of PERL regular expression to enable the rule to not report a violation for the specified output terminals.

Used by	UnloadedOutTerm-ML
Options	yes, no
Default value	no

ignore_parameter_type

Use this parameter avoid reporting violations on two state data types used as parameter types.

By default, the value of the parameter is set to no. In this case, the tag will report violation for two state data types used as parameter types.

Used by	TwoStateData-ML
Options	yes, no
Default value	no

ignore_dim_port_types

Specifies whether the [CheckPortType-ML](#) tag should report violations for multi-dimensional ports and zero-range ports.

By default, the value of this parameter is set to no.

Set the value of this parameter to yes to not report violations for multi-dimensional ports and zero-range ports.

Used by	CheckPortType-ML
Options	yes, no
Default value	no

ignore_sub_modport

Enables the [InterfaceWithoutModport-ML](#) tag to disable violations when the modport is explicitly specified in sub module definition.

By default, the value of the parameter is set to no.

Set the value of the parameter to yes to disable violations when the modport is explicitly specified in sub module definition.

Used by	InterfaceWithoutModport-ML
Options	yes, no
Default value	no

ignore_same_case_item

Enables the [STARC05-2.8.1.3](#) tag to ignore violations for the overlapping

case labels.

By default, the value of the parameter is no and the tag reports violations for such cases.

Set the value of the parameter to yes to ignore violations for the same case items.

Used by	STARC05-2.8.1.3
Options	no, yes
Default value	no

ignore_system_tasks

Enables the [STARC05-2.1.2.4](#) tag to not report LRM allowed system tasks.

By default, the value of the parameter is yes.

Set the value of the parameter to no to ignore LRM allowed system tasks.

Used by	STARC05-2.1.2.4
Options	no, yes
Default value	no

ignore_user_defined_tasks

Enables the STARC05-2.1.2.4 tag to ignore violations for the user-defined tasks.

By default, the value of the parameter is set to no.

Set the value of the parameter to yes to ignore violations for the user-defined tasks.

Used by	STARC05-2.1.2.4
Options	no, yes
Default value	no

ignore_unique_and_priority

Enables the [STARC05-2.8.1.4](#) tag to ignore violations for the unique, unique0, and priority cases.

By default, the value of the parameter is set to no.

Set the value of the parameter to yes to ignore violations for the unique, unique0, and priority cases.

Used by	STARC05-2.8.1.4
Options	no, yes
Default value	no

report_case_type

Enables the [ShiftRegister-ML](#) tag to report a violation for casez, casex, case_inside, unique, unique0 and priority parameter values.

By default, the tag parameter is set to "" .

Use this parameter to report a violation for casez, casex, case_inside, unique, unique0 and priority parameter values.

Used by	ShiftRegister-ML
Options	any of casez, casex case_inside, unique, unique0 and priority
Default value	""

ignore_concat_expr

Specifies whether the [W156](#) tag should ignore violations for the reverse connections which is an expression that concat operation is involved.

By default, the value of the ignore_concat_expr is set to no.

Set the value of this parameter to `yes` to not report violations for the reverse connections which is an expression that concat operation is involved.

Used by	W156
Options	yes, no
Default value	no

ignore_parameter_assignment

Specifies whether the [W164a](#), [W164b](#), [W164a_a](#), and [W164a_b](#) tags should ignore violations for parameter and local parameter assignments.

By default, the `ignore_parameter_assignment` parameter is set to `no`.

Set this parameter to `yes` or comma/space-separated list of rule names to enable the tags to not report any violation for parameter and local parameter assignments.

Used by	W164a , W164b , W164a_a , and W164a_b
Options	yes, no, comma or space-separated list of rule names

ignore_vector_bitselect

Specifies whether the [STARC05-2.8.5.3](#) tag should report a violation for case expressions.

By default, the `ignore_vector_bitselect` parameter is set to `no`.

Set the value of this parameter to `yes` to not report violations for the following cases:

- Case expression is 1 bit constant (for example, `1'b1`)
- Case item expression is bit select of a vector signal (for example, `state[0]`)

NOTE: *Index should be constant.*

Used by	STARC05-2.8.5.3
Options	no, yes
Default value	no

ignore_wildcard_operators

Use this parameter to not report [W467](#) tag violations reported for RHS side of wild operators, ==? and !=?, as well as inside operator, that is, LHS 'inside' RHS.

By default, the value of the parameter is set to no.

Used by	W467
Options	no, yes
Default value	no

MAX_DRIVER_COUNT

Specifies the threshold value to limit the extraction of drivers in fan-in cone.

The default value is 200.

Set the value of the parameter to an integer value to limit the extraction of drivers in fan-in cone to a specified value.

Used by	W415 , STARC05-2.5.1.4 , STARC05-2.5.1.5a , checkMultiple-Drivers , W323
Options	any positive integer value
Default value	200

report_blocking_only

Use this parameter to enable the [EventControlInRHS-ML](#) tag to report only for the blocking assignments.

Used by	EventControlInRHS-ML
Options	yes, no
Default value	no

ignore_parameters

Specifies whether the [W224](#) and [W576](#) tags should ignore violations for parameters.

By default, the *ignore_parameters* parameter is set to no.

Set this parameter to yes to enable the rule to not report any violation for parameters.

Used by	W224 , W576
Options	yes, no
Default value	no

REPORT_CONST_CONNECT

Specifies whether the [DisabledOr](#), [FlopDataConstant](#), and [FlopEConst](#) tags should report constant source details in the violation message.

By default, the value of the parameter is set to no.

Set the value of the parameter to yes to enable the tags to report constant source details in the violation message.

Used by	DisabledOr , FlopDataConstant , and FlopEConst
Options	yes, no
Default value	no

report_internal_netname

This parameter disables the tag to display the hierarchal names of the

violating internal nets. The default value of this parameter is no. Set this parameter to yes to enable the tag to display the hierarchal names of the violating internal nets.

Used by	STARCO5-3.3.6.2
Options	yes, no
Default value	no

report_hanging_cell

(Optional) Specifies whether the [DisabledAnd](#), [DisabledOr](#), [FlopDataConstant](#), [LatchEnableConstant](#), and [LatchDataConstant](#) Tags should flag for instances whose output is hanging.

By default, the value of the `report_hanging_cell` Tag parameter is set to no, and the specified Tags do not flag instances with hanging output.

Set the value of the `report_hanging_cell` Tag parameter to yes to report such instances in the design.

This behavior is same for both library and internally generated instances.

Used by	DisabledAnd , DisabledOr , FlopDataConstant , LatchEnableConstant , LatchDataConstant
Options	yes, no
Default value	no

report_local_name

Specifies whether the [STARCO5-1.3.1.6](#) and [W448](#) tags should report hierarchical or non hierarchical reset signal name in the violation message.

By default, the value of the parameter is set to yes. It will report non hierarchical reset signal name in violation message.

Set the value of the parameter to no to report hierarchical signal name.

Used by	STARCO5-1.3.1.6 , W448
Options	yes, no
Default value	yes

report_inferred_cell

(Optional) Specifies whether the [FlopClockConstant](#), [FlopEConst](#), and [FlopSRConst](#) Tags should flag only inferred flip-flops.

By default, the `report_inferred_cell` Tag parameter is set to no, and the specified Tags flag all the flip-flops (instantiated and inferred flip-flops) in the design. Instantiated flip-flops include instances from lib library cells as well as Verilog libraries with cells encapsulated using ``celldefine`.

Set the value of the `report_inferred_cell` Tag parameter to yes to enable the specified Tags to flag only for the inferred flip-flops.

Used by	FlopClockConstant , FlopEConst , FlopSRConst
Options	yes, no
Default value	no

report_regoutputs_ports

Use this parameter to specify comma-separated list of ports on which [RegOutputs](#) tag needs to be run. The parameter also supports regex.

Used by	RegOutputs
Options	{comma separated port list}
Default value	{ }

strict

(Optional) Specifies the strict Tag-checking mode for specified Tags.

By default, the value of the strict Tag parameter is set to no.

You can also set the value of the `strict` Tag parameter to a comma or space separated list of Tags. In such a case, strict checking is performed only for the Tags specified in the list.

VC SpyGlass Lint Parameters

Used by	TristateConst , LatchGatedClock , LogNMux , W19 , W71 , W116 , W156 , W164a , W164b , W226 , W263 , W287a , W323 , W337 , W342 , W343 , W362 , W415 , W443 , W444 , W456a , W481a , W484 , W490 , W494 , W495 , SigAsgnDelay-ML , UseBusWidth-ML , RedundantLogicalOp-ML , ConstantInput-ML , TristatePort-ML , SelfDeterminedExpr-ML , NoGenLabel-ML , CheckParamSensList-ML , SignedUnsignedExpr-ML , UseSVCasting-ML CombLoop , OneStmtLine , BufClock , ArrayIndex , NotInSens , InferLatch , STARC05-1.1.4.7 , STARC05-1.2.1.3 , STARC05-2.1.6.1v , STARC05-2.1.6.2 , STARC05-2.2.2.2a , STARC05-2.2.2.3a , STARC05-2.5.1.2 , STARC05-2.4.1.4 , STARC05-2.5.1.5a , STARC05-2.8.1.4 , STARC05-2.10.3.2b , STARC05-2.10.3.2b_sa , STARC05-2.9.1.2a , STARC05-2.10.4.6 , STARC05-3.1.5.2 , STARC05-3.2.2.1 , STARC05-3.5.6.7 , STARC05-1.1.2.1a , STARC05-2.1.3.3 , STARC05-3.1.3.5 , STARC05-3.1.4.4 , Rule_3 , Rule_24 , Rule_28
Options	yes, no, comma/space-separated list of Tags
Default value	no

supplyHigh

(Optional) Specifies the name of the file containing supply high net names.

The file must be a text file with each supply high name net name (hierarchical name) in a separate line. You can use newlines or spaces only for indenting purposes.

Used by	checkPinConnectedToSupply
Options	Name of the file containing supply high net names
Default value	_null_

supplyLow

(Optional) Specifies the name of the file containing supply low net names. The file must be a text file with each supply low name net name (hierarchical name) in a separate line. You can use newlines or spaces only for indenting purposes.

The `supplyLow` Tag parameter is used by the following Tags:

Used by	checkPinConnectedToSupply
Options	Name of the file containing supply low net names
Default value	<code>_null_</code>

supplyNetFile

Specifies the name of the file containing supply net (both `supply0` and `supply1` nets) names.

The file must be a text file with each line of the following format:

```
<supplytype> <net-name-list>
```

Where, *<supplytype>* is `supply0` or `supply1` and *<net-name-list>* is a space-separated list of simple supply net names (hierarchical names) of the particular type. Do not use hierarchical names.

You can use newlines or spaces only for indenting purposes.

Used by	FloatingInputs , FlopDataUndriven
Options	Name of files containing supply net names
Default value	<code>_null_</code>

case_condition_count

Enables the [ShiftRegister-ML](#) tag to check the minimum number of case conditions to be present to report a violation.

By default, the tag parameter is set to 16.

Use this parameter to check the minimum number of case conditions to be

present to report a violation.

Used by	ShiftRegister-ML
Options	interger
Default value	16

consider_sgdc_clock

Specifies whether the [LatchGatedClock](#) Tag checks only for the clocks that are specified in the sgdc file.

By default, the value of the `consider_sgdc_clock` parameter is set to `no`.

Set the value of this parameter to `yes` to check for the clocks specified in the sgdc file.

Used by	LatchGatedClock
Options	yes, no
Default value	no

checkunusedlatch

Specifies whether the [LatchGatedClock](#) Tag checks for latches whose output is not used.

By default, the value of the `checkunusedlatch` parameter is set to `no` and the [LatchGatedClock](#) Tag does not consider the latches whose output is not used.

Used by	LatchGatedClock
Options	yes, no
Default value	no

effort_level

Specifies the effort level that VC SpyGlass Lint should make for checking the specified tag. For some cases VC SpyGlass Lint may take excessive time, if `effort_level` is set to a high value. If this parameter is set to a smaller value, then the Tag can miss out some violations.

By default, the `effort_level` parameter is set to 100 iterations. However, you can set the value of the parameter to any positive integer value.

For CombLoop parameter:

Specifies the number of times the *CombLoop* Tag should traverse a net to detect combinational loop. The value of the `effort_level` tag parameter specifies the effort that VC SpyGlass Lint makes while finding the offending issue. There may be cases where VC SpyGlass Lint may take excessive time if the effort level is set high through this parameter.

By default, the *CombLoop* Tag passes through a particular net 100 times to detect combinational loop. You can modify this number as per your requirement.

Used by	LatchFeedback , CombLoop , STARC05-1.2.1.3 , STARC05-1.3.2.2 , STARC05-1.6.2.2 , Rule_24 , Rule_28 , SameNetConnectionCheck , GlitchGenCircuitCheck
Options	<positive integer value>
Default value	100

ignoreCellName

For LatchFeedback/LatchGatedClock:

Specifies the naming pattern of the clocks gated through lib cell instances for which the violations should be ignored.

By default, the `ignoreCellName` parameter is not set and the Tag reports violations for all latches driven by gated or internally generated clocks.

Set this parameter to a PERL regular expression to specify the clocks gated through lib cell instances for which the violations should be ignored.

For W336:

Specifies the modules that should be ignored by the W336 Tag. By default, the *ignoreCellName* parameter is not set.

Set this parameter to a comma separated list of PERL regular expressions containing names of the modules that should be ignored by the W336 Tag.

For STARC05-1.4.3.1a/ STARC05-1.4.3.1b/BufClock

Specifies the cell names to be ignored through which clocks are inverted.

By default, the *ignoreCellName* parameter is not set and the Tags give violation for all gated clocks.

Set this parameter to a PERL regular expression to specify the cell names that need to be ignored, through which the clocks are inverted.

Used by	LatchGatedClock , W336 , STARC05-1.4.3.1a , STARC05-1.4.3.1b , STARC05-1.3.2.1a , BufClock
Options	PERL regular expressions
Default value	""

ignoreModName

For STARC05-1.4.3.1a

Set this parameter to a PERL regular expression to specify the module names to be ignored through which clocks are inverted.

For STARC05-1.4.3.1b

Set this parameter to a PERL regular expression to specify the clocks gated through module instances that should be ignored.

Used by	STARC05-1.4.3.1a , STARC05-1.4.3.1b
Options	PERL regular expressions
Default value	""

latch_chain_max_length

Specifies the number of latches in the feedback path.

By default, the `latch_chain_max_length` parameter is set to 2.

The [LatchFeedback](#) Tag finds the feedback path going from combinatorial logic and the latches. Set this parameter to a positive integer value to specify/control the number of latches in the feedback path.

Used by	LatchFeedback
Options	Positive integer
Default value	2

module_based_pin_names

Enables the [PortConnectedToMux-ML](#) to identify the ports of module, which needs to be analyzed by the tag.

By default, the value of the parameter is set to " ".

Specify a comma-separated module-based pin list to identify the ports of a module, which needs to be analyzed by the tag.

Used by	PortConnectedToMux-ML
Options	Comma-separated module based pin list
Default value	" "

overlappingLatchLoops

Enables detection of overlapping loops.

By default, the value of this parameter is set to `yes` and the [LatchFeedback](#) Tag detects overlapping loops.

Set the value of this parameter to `no` to disable detection of overlapping loops.

Violations of overlapping loops are reported, even if part of the overlapping loop does not contain a latch.

Used by	LatchFeedback
Options	yes, no
Default value	yes

check_complete_sensitivity_list

Checks for the implicit or complete always block.

By default, the value of the `check_complete_sensitivity_list` parameter is `yes` and the [sim_race01](#) Tag reports violation for implicit or complete always block.

When the value of the parameter is set to `no`, the Tag will not report any race for implicit or complete always block with respect to the other always block.

Used by	sim_race01
Options	yes, no
Default value	yes

check_seq_assign

Specifies whether the [sim_race01](#) Tag reports the race condition for a signal that is assigned in the sequential block bus signals.

By default, the value of the `check_seq_assign` parameter is `no` and the [sim_race01](#) Tag reports the race condition for a signal that is assigned in the sequential block bus signals.

Set this parameter to `yes`, to not report such cases.

Used by	sim_race01
Options	yes, no
Default value	no

effort_level_sim_race

Specifies the amount of effort to be put by the [sim_race01](#) Tag while detecting the race condition.

By default, the value of this parameter is 0. Set this parameter to an integer value in case of high run time to specify the amount of effort to be applied while detecting the race condition.

Used by	sim_race01
Options	Positive integer value
Default value	0

handle_large_bus

The `handle_large_bus` parameter specifies whether to check large arrays. By default, this parameter is set to no and the respective tags do not process large arrays.

Set this parameter to yes to process large arrays (greater than 50,000) and report violation if not used correctly. If you set this parameter to `<Tag-name>`, the `handle_large_bus` parameter performs the check for that particular tag only.

Used by	sim_race01 , W111 , W120 , W123 , W240 , W241 , W494 , W495 , W528 , UnusedFunctionInput-ML , ImproperRangeIndex-ML , W499
Options	yes, no
Default value	no

ignore_typedefs

Ignores all type definitions from tag checking.

By default, the value of the `ignore_typedefs` parameter is set to no. In this case, tags mentioned in the table below consider type definitions for tag checking.

Set the value of the parameter to yes to ignore all type definitions.

Used by	W341
Options	yes, no
Default value	no

waiver_compat

Enables the waivers in the specified tags to work correctly even if the line numbers of the RTL get changed.

By default, this parameter is set to `no` and the violations are not waived when:

- Tag message has line number string
- Waivers are applied based upon the message string (having line number), and during subsequent runs
- RTL has changed and hence the line number

If you set the value of this parameter to `yes` or `<Tag-name>`, it ensures that the Tags do not generate the line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.

Used by	sim_race01 , sim_race02 , W121 , W143 , W398 , W392 , W402b , W415a , W546 , ConsCase , ClockPhase , STARCO5-1.1.1.3 , STARCO5-2.4.1.1.3 , LINT_MULTIASSIGN_BLOCKING_SIG
Options	no, yes, comma or space separated list of Tags
Default value	no

cell_check_for_race

Use this parameter to report a race condition between clock pins and enable pins of custom cells as well.

The following delimiters are used by this parameter:

There are 3 delimiter characters in the parameter. They are,

- **:** (**colon**): to deviate cell name and pair of pin names which includes clock pin and enable pin names

- **, (comma)**: to deviate clock and enable pin names
- **; (semi-colon)**: to deviate entries of valid cells

Used by	ClockEnableRace
Options	Comma-separated cell names specified in the following format: <code>'<cell name1>:<pin1>,<pin2>;<cell name2>:<pin1>,<pin2>'</code>
Default value	" "

checkcomblogic

(Optional) Checks for combinational cells between clock and enable path. This Tag parameter is used by the [ClockEnableRace](#) Tag.

If the combination logic is not constrained by the [seq_atpg](#) constraint or power ground simulation, the path will be considered transparent. If the [seq_atpg](#) constraint or VDD/VSS is set, and if the path is not blocked, then the path will be treated as continuous. In case of a latch, if it is transparent, the path will be treated as continuous, otherwise discontinuous.

By default, the value of the `checkcomblogic` Tag parameter is set to `no`, and the [ClockEnableRace](#) Tag does not check for the combinational cells between clock and enable path.

Used by	ClockEnableRace
Options	yes, no
Default value	no

check_mux_select_pins

(Optional) Reports MUXes that contain a large number of select pins

By default, the [LogNMux](#) Tag reports a violation when:

$\log(\text{number of input pins in a LogN MUX}) > \text{Limit specified by the } \text{logmux\_max} \text{ parameter}$

If you set this parameter to `yes`, the Tag also reports a violation when:
Number of select input pins in a LogN MUX > Limit specified by the `logmux_max` parameter.

Used by	LogNMux
Options	yes, no
Default value	no

consider_cont_assign

Specifies whether the [STARC05-1.6.2.2](#) rule should consider the continuous statement or feedthrough module while checking for snake path.

By default, this parameter is set to `no` and the rule does not consider continuous statement or feedthrough module while checking for snake path.

Set this parameter to `yes` to enable the rule to consider such cases.

Used by	STARC05-1.6.2.2
Options	yes, no
Default value	no

delaymax

(Optional) Specifies the maximum number of logic level beyond which the [LogicDepth](#) Tag should flag a message.

By default, the [LogicDepth](#) Tag flags paths that have more than 40 logic levels. Set the `delaymax` Tag parameter to a positive integer number to change the default maximum limit.

Used by	LogicDepth
Options	Positive integer value
Default value	40

delaymax_err

(Optional) Specifies the maximum number of logic levels beyond which the [LogicDepth](#) Tag should flag an error message.

The value specified in the delaymax_err Tag parameter should be greater than the value specified in the [delaymax](#) Tag parameter. If the value of delaymax_err parameter is less than delaymax parameter, the value of delaymax Tag parameter will be considered. However, if a value is specified for delaymax_err parameter but no value is specified for delaymax parameter, the default value of delaymax parameter will be considered.

By default, the value of the delaymax_err Tag parameter is set to 200, and the [LogicDepth](#) Tag reports an error message for all logic levels greater than 200 and a warning message for all logic levels ranging between 40 (default value of delaymax parameter) to 200 (default value of delaymax_err parameter).

You can specify different positive integer values to the delaymax and delaymax_err Tag parameters to change the above default setting. You can specify the ranges in the following manner:

```
configure_lint_tag_parameter -tag LogicDepth -parameter
delaymax -value A -parameter delaymax_err -value B
```

Where,

A and B are positive integer values.

The above setting signifies the following:

- No violation will be reported for logic levels between 0 to A
- A warning message will be reported for logic levels from A to B
- An error message will be reported for logic levels greater than B

Used by	LogicDepth
Options	Positive integer value
Default value	200

delaymax_floptoflop

(Optional) Specifies the maximum number of logic levels between flip-flop output pin to flip-flop data pin beyond which the [LogicDepth](#) Tag flags a message.

By default, the `delaymax_floptoflop` parameter is set to -1 and its value is same as that specified in the [delaymax](#) parameter.

You can set the `delaymax_floptoflop` Tag parameter to a positive integer number to change the default maximum number of logic levels from one flip-flop to another flip-flop.

Used by	LogicDepth
Options	Positive integer value
Default value	-1

delaymax_floptomem

(Optional) Specifies the maximum number of logic level beyond which the [LogicDepth](#) Tag should flag a message while checking paths from a flip-flop output pin or a primary input pin to a memory element input pin.

By default, the `delaymax_floptomem` parameter is set to -1 and its value is same as that specified in the [delaymax](#) parameter.

You can set the `delaymax_floptomem` Tag parameter to a positive integer number to change the default maximum limit for the corresponding paths.

Used by	LogicDepth
Options	Positive integer value
Default value	-1

delaymax_floptooutput

(Optional) Specifies the maximum number of logic levels between flip-flop output pin to primary output pin beyond which the [LogicDepth](#) Tag flags a message.

By default, the `delaymax_floptoooutput` parameter is set to -1 and its value is same as that specified in the [delaymax](#) parameter.

You can set the `delaymax_floptoooutput` Tag parameter to a positive integer number to change the default maximum number of logic levels between flip-flop output pin to primary output pin.

Used by	LogicDepth
Options	Positive integer value
Default value	-1

delaymax_inputtoflop

(Optional) Specifies the maximum number of logic levels between primary input pin to flip-flop data pin beyond which the [LogicDepth](#) Tag flags a message.

By default, the `delaymax_inputtoflop` parameter is set to -1 and its value is same as that specified in the [delaymax](#) parameter.

You can set the `delaymax_inputtoflop` Tag parameter to a positive integer number to change the default maximum number of logic levels between primary input pin to flip-flop data pin.

Used by	LogicDepth
Options	Positive integer value
Default value	-1

delaymax_inputtooutput

(Optional) Specifies the maximum number of logic levels between primary input pin to primary output pin beyond which the [LogicDepth](#) Tag flags a message.

By default, the `delaymax_inputtooutput` parameter is set to -1 and its value is same as that specified in the [delaymax](#) parameter.

You can set the `delaymax_inputtooutput` Tag parameter to a positive

integer number to change the default maximum number of logic levels between primary input pin to primary output pin.

Used by	LogicDepth
Options	Positive integer value
Default value	-1

delaymax_memtoflop

(Optional) Specifies the maximum number of logic level beyond which the [LogicDepth](#) Tag should flag a message while checking paths from a memory element output pin to a flip-flop input pin or a primary output pin.

By default, the `delaymax_memtoflop` parameter is set to -1 and its value is same as that specified in the [delaymax](#) parameter.

You can set the `delaymax_memtoflop` Tag parameter to a positive integer number to change the default maximum limit for the corresponding paths.

Used by	LogicDepth
Options	Positive integer value
Default value	-1

delaymax_memtomem

(Optional) Specifies the maximum number of logic level beyond which the [LogicDepth](#) Tag should flag a message while checking paths from a memory element output pin to a memory element input pin.

By default, the `delaymax_memtomem` parameter is set to -1 and its value is same as that specified in the [delaymax](#) parameter.

You can set the `delaymax_memtomem` Tag parameter to a positive integer number to change the default maximum limit for the corresponding paths.

Used by	LogicDepth
Options	Positive integer value
Default value	-1

ignore_icg_cell

Specifies whether the [ClockEnableRace](#) Tag should report race through an Integrated Clock Gating (ICG) cell.

By default, the `ignore_icg_cell` parameter is not set and the [ClockEnableRace](#) Tag reports the race through ICG cell.

Set this parameter to the ICG cell name(s) to not report race through those ICG cells.

This parameter supports comma-separated list of ICG cell names. This parameter also supports regular expression.

Used by	ClockEnableRace
Options	comma-separated list of ICG cell names
Default value	none

ldto and ldfrom

(Optional) Specify the clock names for originating and terminating flip-flops for the [LogicDepth](#) Tag while reporting logic depths.

By default, the [LogicDepth](#) Tag reports logic depth for paths between all combinations of flip-flops, primary ports, and memory elements. You can customize the [LogicDepth](#) Tag checking by specifying the paths to be checked using the `ldfrom` and `ldto` Tag parameters as follows:

VC SpyGlass Lint Parameters

Set Tag Parameters...		To check...
ldfrom	ldto	
Originating clock	Terminating clock	All logic paths that start from the output pins of flip-flops triggered by the originating clock and end in data pins of flip-flops triggered by the terminating clock
Originating clock	Set to *	All logic paths starting from the output pins of flip-flops triggered by the originating clock and ending in the data pin of a flip-flop triggered by any clock or a primary output
Set to *	Terminating clock	All logic paths starting from the output pin of any flip-flop or a primary input pin and ending at the data pins of flip-flops triggered by the terminating clock

You can specify multiple clocks with the `ldfrom` and `ldto` Tag parameters.

It is possible that the clocks for some end points (output ports or data pin of flip-flops) are not defined and therefore cannot be ascertained. To report delay in such flip-flops, use the `?` character that represents unknown clocks. Therefore, when you specify the `?` character as a clock name in the `ldfrom` or `ldto` Tag parameters, delay from or delay up to a flip-flop of the unknown clock is also calculated and reported.

By definition, the primary inputs and primary outputs fall in the unknown (?) clock domain. Therefore, using the `?` character in the `ldfrom` or `ldto` Tag parameters also reports delays in paths starting from primary inputs and paths ending in primary outputs respectively.

You need to specify complete hierarchical clock net name with the `ldfrom` and `ldto` Tag parameters as in the following example:

```
configure_lint_tag_parameter -tag LogicDepth -parameter
ldfrom -value top.clk1 -parameter ldto -value ?
```

Used by	LogicDepth

Options	Net name, ?
Default value	*

logmux_max

(Optional) Specifies the maximum number of select bits beyond which the *LogNMux* Tag reports MUX constructs.

By default, the *LogNMux* Tag reports MUX constructs that have more than three select lines. Use the `logmux_max` parameter to specify a different integer number.

Used by	<i>LogNMux</i>
Options	Positive integer value
Default value	3

report_common_source

Use this parameter to report violations when the clock and enable pin of the flip-flop is connected to a common source. Traversal will consider the paths irrespective of whether flop is found in the fan-in of clock or enable pin.

Used by	<i>ClockEnableRace</i>
Options	
Default value	3

show_connected_net

Enables the tags to show the immediate net connected to a flip-flop/mux.

By default, the `show_connected_net` parameter is set to no.

Set this parameter to yes to enable the tag to show immediate net connected to a flip-flop/mux.

Used by	STARC05-1.4.3.4 , STARC05-1.3.1.3
Options	yes, no
Default value	yes

show_viol_count

Specifies the violation limit for each start point in the [LogicDepth](#) Tag.

The `show_viol_count` Tag parameter reports $\langle N \rangle$ violations per start point, if the value of the parameter is set to $\langle N \rangle$. For example, if the value of the `show_viol_count` is set to 2, then 2 violations are reported per start point.

The default value of the `show_viol_count` parameter is set to -2, that is, the `LogicDepth` Tag reports only the longest path exceeding the limit for a start point.

For more information on the `LogicDepth` Tag behavior with negative values of the `show_viol_count` parameter, refer to the [LogicDepth](#) Tag.

Used by	LogicDepth
Options	Integer value
Default value	-2

allow_clk_in_condition

Specifies whether the [W122](#) Tag traverses a combinational block present inside a sequential block.

By default, the `allow_clk_in_condition` parameter is set to `no`.

If you set this parameter to `yes`, the Tag does not traverse a combinational block present inside a sequential block and does not report a violation in such cases.

Used by	W122

Options	yes, no
Default value	no

NOTE: *The `allow_clk_in_condition` parameter is supported only for Verilog functionality.*

allviol

Specifies whether all or limited number of tag messages should be reported.

By default, the `allviol` parameter is unset and the specified tag reports only a maximum of 500 messages per design and a maximum of 20 messages per module.

You can set the value of the `allviol` parameter to `Yes` (to report all messages) or to `No` (to report limited messages). You can also set the value of this parameter to a comma or space-separated list of tags.

Used by	W257 , W314 (Verilog), DisabledAnd , DisabledOr , FloatingInputs , FlopClockConstant , FlopClockUndriven , FlopDataConstant , FlopDataUndriven , FlopEConst , FlopSRConst , LatchDataConstant , LatchEnableConstant , TristateConst , checkPinConnectedToSupply , BitDataType-ML , InlineComment-ML , MultiOpInModule-ML , NoArray-ML , RedundantLogicalOp-ML , UseBusWidth-ML , UseSVCasting-ML , NoFeedThrus-ML , SetResetConverge-ML , SigHierName , PortComment , SignalComment , VariableComment , Indent , HardConst , NoTopGates , NoGates , CombLoop , TypeComment , PortGrpComment , ConstantComment , STARC05-1.1.2.6a , STARC05-1.1.2.6b , STARC05-1.1.3.3e , STARC05-1.2.1.3 , STARC05-1.6.2.2 , STARC05-2.3.1.3 , STARC05-3.1.4.2 , Rule_16 , Rule_24 , Rule_28 , LINT_LOOP_THRU_FLOP_WITHOUT_SETRESET
Options	yes, no, comma or space-separated list of tags
Default value	no

assume_driver_load

Use the `assume_driver_load` parameter in replacement of `checkDriverInModule` parameter.

By default, the `assume_driver_load` parameter is not set and the [W415](#), [Rule_3](#), and [Rule_32](#) tags do not report a violation, if a net connected to output port of an instance is not driven inside that instance.

The `assume_driver_load` parameter performs the tag checking for the tags based on the following parameter values:

- **load:** Checks for loaded nets only and reports a violation, if net is multiple driven but there is no load.
- **driver:** Checks for driven nets only and reports violation, if net is multiple driven but there is no driver on it.
- **both or yes:** Checks both driver and load and reports a violation, if net is multiple driven but there is no valid driver or load.

Used by	W415 , Rule_3 , Rule_32
Options	no, yes, both, driver, load
Default value	no
Default Value in GuideWare2.0	yes

avoid_port_signal

Specifies whether the [W498](#) tag reports a violation for port signals.

By default, the `avoid_port_signal` parameter is set to no.

Set this parameter to `yes` to enable the tag to not report a violation for port signals.

Used by	W498
Options	yes, no
Default value	no

casesize

Specifies the maximum permissible width of case selector expression and the maximum number of case clauses allowed as checked by the [W453](#) Tag.

By default, the W453 Tag flags case constructs where the case selector expression is wider than 16 bits and the number of case clauses is more than 20.

Specify the casesize Tag parameter as follows:

```
configure_lint_tag_parameter -tag W453 -parameter casesize
-value '8-16'
```

Please note that the two values specified by the casesize Tag parameter are separated by a hyphen.

Used by	W453
Options	<character-string>
Default value	"16-20"

checkblocking

Specifies to check for blocking assignment statements only while checking with the [W415a](#) and [LINT_MULTIASSIGN_BLOCKING_SIG](#) tag.

By default, the checkblocking Tag parameter is not set and the W415a Tag checks for all kinds of multiple assignment within the same scope.

You can set the value of the checkblocking Tag parameter to yes (to enable the mode) or to no (to disable the mode).

Used by	W415a , LINT_MULTIASSIGN_BLOCKING_SIG
Options	yes, no
Default value	no

checkfullbus

The following table describes the impact of this parameter on the VC SpyGlass tags:

Tag	Tag behavior	
	When the <code>checkfullbus</code> parameter is set to <code>no</code>	When the <code>checkfullbus</code> parameter is set to <code>yes</code>
W240	Reports a violation if any bit of the input port is unread	Reports a violation only when the entire input port is unread
W120	Reports a violation if any bit of a variable is defined or declared but not used in the current scope of a module or an architecture	Reports a violation only when all bits of a variable are completely unused in the current scope of a module or an architecture
W528	Reports a violation if any bit of a variable is set but not read in the current scope of a module or an architecture	Reports a violation only when all bits of a variable are completely unread in the current scope of a module or an architecture
UnUsedFunctionInput-ML	Reports a violation if any bit of a function input is declared but not read in the current scope of a module	Reports a violation only when all bits of a function input are completely unread in the current scope of a module
UndrivenInTerm-ML	Reports a violation for vector signals if any bit of vector signals is undriven but loaded	Does not report a violation for vector signals if at least one bit of vector terminal is driven
Used by	W240 , W120 , W528 , UnUsedFunctionInput-ML , UndrivenInTerm-ML	
Options	yes, no	
Default value	no	

checkfullrecord

Specifies whether the [W456a](#), [W123](#), [W528](#), [W120](#), and [W240](#) Tags should check for all the elements of a VHDL record.

By default, the `checkfullrecord` parameter is set to `no`.

Set this parameter to `yes` to not report violation for a record:

- W456a: if all elements of the record are not read.
- W123: if at least one element of the record is read and set.
- W528: if at least one element of the record is set and read.
- W120: if at least one element of the record is used.
- W240: if at least one element of the record is read.

Used by	W456a , W123 , W528 , W120 , W240
Options	yes, no, comma separated list of Tags
Default value	no

checkfullstruct

Specifies if the specified tag should report a violation for structs.

By default, this parameter is set to no.

Set the checkfullstruct parameter to yes to not report a violation for structs if at least one element of the struct is read.

Used by	W240 , W528
Options	yes, no
Default value	no

check_bbox_driver

Specifies if the [W415](#), [Rule_3](#), or [Rule_32](#) tags should consider the black-box as a driver.

By default, this parameter is set to no and the [W415](#), [Rule_3](#), and [Rule_32](#) tags does not consider black-box as a driver. Set the check_bbox_driver parameter to yes to consider the black-box as a valid driver.

Used by	W415 , Rule_3 , Rule_32
Options	yes, no
Default value	no

check_case_type

Specifies the case types on which user want to perform Tag checking.

By default, the check_case_type parameter is set to all and W551 Tag reports violation for all three case types, that is, full_case, unique case and priority case.

Set the value of the parameter to one or more of the following values as shown in the table:

Value	Description
priority	The W551 Tag checks only for priority case.
unique	The W551 Tag checks only for unique case.
full_case	The W551 Tag checks only for full_case case.
all	The W551 Tag checks for priority, unique, and full_case cases.

You can assign more than one of the above values to the parameter and Tag checking is performed on all case types passed.

Consider the following example:

```
configure_lint_tag_parameter -tag 4551 -parameter
check_case_type -value 'priority,unique'
```

In the above example, the W551 Tag detects the presence of default clause in priority cases as well as unique cases.

Used by	W551
Options	priority, unique, full_case, all
Default value	all

check_complete_design

Specifies whether the [W391](#) and [W392](#) should check within the complete design.

By default, `check_complete_design` parameter is set to `no` and the Tags [W391](#) and [W392](#) Tags checks within the design unit.

You can set the value of `check_complete_design` parameter to `yes` to check within the complete design for violations.

Used by	W391 , W392
Options	yes, no, comma separated list of Tags
Default value	no

check_concat_max_width

Specifies the width to be considered by the [W164a](#) and [W164b](#) Verilog Tags after adding zero concatenated bits.

If the RHS expression is concatenated with zero bits, by default, no violation is reported when the width of the LHS expression is present between the width of the RHS expression without considering zero concatenated bits and the width of the RHS after adding zero concatenated bits.

When you set the `check_concat_max_width` parameter to `yes`, the RHS width is considered as the width after adding zero concatenated bits. That is, the violation is reported if the LHS width does not match the RHS width after adding zero concatenated bits.

Used by	W164a , W164b
Options	yes, no, comma separated list of Tags
Default value	no

checkconstassign

Specifies whether the [W415](#), [Rule_3](#), and [Rule_32](#) tags should flag cases where the same constant value is assigned multiple times.

By default, the `checkconstassign` tag parameter is set to `no`, and the

tags ignore cases where a net is driven multiple times by the same constant value or supply nets.

Set the value of the `checkconstassign` tag parameter to `yes` to flag cases where a net is driven multiple times by the same constant value or supply nets.

For example, the `W415` tag does not report violation in the following case if the `checkconstassign` tag parameter is set to `no`:

```
assign out = 1'b0;
..
assign out = 1'b0;
```

Used by	W415 , Rule_3 , Rule_32
Options	yes, no
Default value	no

check_const_selector

Specifies if the [W171](#) and [W226](#) Tags check for the case selector.

By default, the `check_const_selector` parameter is set to `no`.

If you set this parameter to `yes`:

- The `W171` Tag checks whether case selector is constant or not. The `W171` Tag does not report any violation if case selector is a constant and case label belongs to one-hot operation, although case label is a variable.
- The `W226` Tag checks whether all case labels are one-hot. If so, the Tag does not report a violation even if the case selector is a constant.

Used by	W171 , W226
Options	yes, no
Default value	no

check_counter_assignment

Reports a violation for the counter type of assignments.

By default, the value of this parameter is set to `no` and the Tag does not check counter type assignments. Set the `check_counter_assignment` parameter to `yes` to report violation for counter type of assignments.

The following settings can be performed to report the counter type of assignments:

- Set the `check_counter_assignment` parameters to `yes`.
- **Verilog RTL:** Set the `check_static_value` parameter to `yes` to enable Tag checking on static expressions and expressions that have a static part.

An assignment expression is considered as counter type assignment if the following conditions are met:

- ☐ 1 is added to a variable/signal on the RHS of the assignment
- ☐ The width of the variable on the RHS matches the width of the LHS expression

For example, the following assignments are considered as counter type assignments:

```
reg [5:0]a, b;
reg [1:0]c, d;
always @(*)
begin
    a = a+1;
    b = a+1;
    a = 1+a;
    c[1] = c[1] + 1;
    c[1:0] = 1 + d[1:0];
end
```

The following assignments are **not considered** as counter type assignments (as explained with the in-line comments):

```
reg [5:0]a, b;
reg [1:0]c, d;
always @(*)
```

```
begin
  a = a+b;           //1 is not added to a variable
  b[2:0]= a+1;       //width of the variable on RHS does
not match width of LHS
  c[1:0]= (c[1]*2)+1; //addition of 1 is with an
expression instead of variable
  d[1:0]= 2 + d[1:0]; //2 is added instead of 1
end
```

- **VHDL RTL:** Set the [strict](#) parameter to yes to enable Tag checking on static expressions and expressions that have a static part. The `check_counter_assignment` parameter is enabled only if the [nocheckoverflow](#) parameter is set to no.

NOTE: The `check_counter_assignment` parameter is applicable for the W164a (Verilog and VHDL) and W164b (VHDL only) Tags.

Used by	W164a , W164b , W116
Options	yes, no
Default value	no

checkDriverInModule

The `checkDriverInModule` has been deprecated and will be removed in the next major release. Use the [assume_driver_load](#) parameter in replacement of the `checkDriverInModule` parameter.

checknonblocking

Specifies to check for non-blocking assignment statements only while Tag checking with the [W415a](#) Tag.

By default, the `checknonblocking` Tag parameter is not set and the W415a Tag checks for all kinds of multiple assignments within the same scope. It does not distinguish between blocking and nonblocking assignment statements.

You can set the value of the `checknonblocking Tag` parameter to `yes` (to enable the mode) or to `no` (to disable the mode).

Used by	W415a
Options	yes, no
Default value	no

check_genvar

Specifies whether the [W116](#) Tag should report violations for genvar variables.

By default, the `check_genvar` parameter is set to `no` and the W116 does not report a violation when the operands are genvar.

When this parameter is set to `yes`, the W116 Tag reports violation for unequal length operands in the bit-wise logical, arithmetic, and ternary operators, although operands are genvar. This parameter is applicable for **Verilog** only.

Used by	W116
Options	yes, no
Default value	no

check_implicit_senselist

Specifies whether the [W502](#) or [STARC05-2.6.2.2](#) tags reports violation for `always @*` and `always_comb`.

By default, the specified tag does not report violation for `always @*` and `always_comb`.

Set the value of the `check_implicit_senselist` parameter to `yes` to report violation for such cases.

Used by	W502 , STARC05-2.6.2.2
Options	yes, no
Default value	no

check_initialization_assignment

Specifies whether the [W415a](#) and [LINT_MULTIASSIGN_BLOCKING_SIG](#) tags report a violation when an entire vector is initialized with an initialization value and a subset of the vector is overwritten.

By default, the Tags report a violation when an entire vector is initialized with an initialization value and a subset of the vector is overwritten. If you set this parameter to `yes`, the Tag does not report a violation for such cases.

Used by	W415a , LINT_MULTIASSIGN_BLOCKING_SIG
Options	yes, no
Default value	no

check_latch

Specifies whether the [W392](#) Tag check for the non-edge triggered cases.

By default, this parameter is set to `no` and the [W392](#) Tag reports the edge triggered cases only.

Set the `check_latch` parameter to `yes` to enable the [W392](#) Tag to report the non-edge triggered cases also.

Used by	W392
Options	yes, no
Default value	no

checkOperatorOverload

Specifies whether the [W116](#) Tag reports inconsistent bit-width mismatch and the [W164a](#) and [W164b](#) Tags evaluate width of the expression without considering overloaded operators.

This parameter is applicable for **VHDL** Tags only.

- For the [W116](#) Tag:
By default, the `checkOperatorOverload` parameter is set to `yes` and the [W116](#) Tag does not report inconsistent bit-width mismatch for the overloaded operators from non-IEEE packages. Set the value of this parameter to `no` to report violations in such cases.
- For the [W164a](#) and [W164b](#) Tags:
By default, the `checkOperatorOverload` parameter is set to `yes` and the [W164a](#) and [W164b](#) Tags evaluate the width of the expression considering the overloaded operator. Set this parameter to `no` to evaluate width of the expression without considering overloaded operators.

Used by	W116 , W164a , W164b
Options	yes, no, comma separated list of Tags
Default value	yes

check_next_state

Enables the specified tag to validate the defined naming convention in the next state of FSM.

By default, the `check_next_state` parameter is set to `"_ns"`.
Set the value of this parameter to comma separated PERL regular expression patterns to define the prefix or suffix (naming convention) for the next state of FSM, which is used by the specified tag to validate the next state names of FSM.

Used by	FSMNextStateName-ML
Options	Comma separated PERL regular expression pattern
Default	"_ns"

check_pattern

Specifies whether the specified tag should validate suffix or prefix naming convention in the next state of FSM.

By default, the *check_pattern* parameter is set to `suffix` and the *specified* tag validates the suffix naming convention in the next state of FSM.

Set the value of the *check_pattern* parameter as `prefix` to enable the *specified* tag to validate the prefix naming convention in the next state of FSM.

If some value other than prefix or suffix is specified, then tag check suffix in next state, as per the default value.

Used by	FSMNextStateName-ML
Options	prefix, suffix
Default	suffix

CHECKSIBLING

The default value of the parameter is yes. In this case, the specified tags report violations for the sibling hierarchies also.

Set the value of the parameter to no ignore violations for the sibling hierarchies.

Used by	RegInputOutput-ML , UndrivenNUnloaded-ML
Options	yes, no
Default	yes

check_param_association

Specifies whether the [W146](#), [STARC05-3.2.3.1](#), or [PragmaComments-ML](#) tags reports violation for parameters in instantiation.

By default, the value of the parameter is set to `no` and the *specified* tags

do not report violation for parameter instances.
Set the value of the parameter to yes to report such cases.

Used by	W146 , STARC05-3.2.3.1 , PragmaComments-ML
Options	yes, no
Default value	no

consider_lrm_for_div

Specifies if the [W164a](#) and [W164b](#) tags should consider the width of the division to be the maximum width of the operands.
By default, the `consider_lrm_for_div` parameter is set to no.
Set the `consider_lrm_for_div` parameter to yes to enable the tag to calculate the width of the division to be the maximum width of the operands (LRM width) when `nocheckoverflow` parameter is set to no.

Example:

```
a[2:0] = b[2:0] / c[3:0]; // without parameter on, no violation as width of RHS is 3// but with parameter on,width of RHS is 4 and hence violation.
```

Used by	W164a , W164b
Options	yes, no
Default value	no

consider_sub_as_add

Specifies if the [W416](#), [W164a](#) and [W164b](#) tags should calculate the width of $a-b/b-1$ as $a+b/b+1$ respectively.
By default, when the `nocheckoverflow` parameter is set to no, the [W164a](#) and [W164b](#) tags consider the width of expression like $b-1$ as the width of b when there is no underflow due to -ve operator. However, this behavior is limited to a simple - ve operation, where this is a part of arithmetic expression other than $+/-$.

Set the `consider_sub_as_add` parameter to `yes` to enable the tags to calculate the width of `a-b / b-1` as `a+b/ b+1` respectively.

Used by	W416 , W164a , W164b
Options	yes, no
Default value	no

check_shifted_only

This parameter has been deprecated and its functionality is covered by the [handle_shift_op](#) parameter. The following table describes the corresponding values of both the parameters for your reference:

check_shifted_only Parameter Values	Corresponding handle_shift_op Parameter Values	Description
yes, both, <Tag-Name>	shift_both or <Tag-Name>	Shifted width is considered for the both, left and right shift expressions
left	shift_left	Shifted width is considered only for the left shift expressions
right	shift_right	Shifted width is considered for the right shift expressions only
no	no	Default behavior

check_shifted_width

For W486 tag:

Specifies whether to consider the natural width of the left operand for a left shift expression.

By default, this parameter is set to `no` and for a **left shift expression**, the [W486](#) Tag evaluates the default width as 32 bits if left operand is constant integer and right operand is non static.

Set this parameter to `yes` to consider the natural width of the left operand in such a case.

For STARC05-2.1.3.1 tag:

The default value of the parameter is `no`. In this case, the non-shifted width of the function argument is compared with the function input width to filter the violation.

When you set the value of the parameter to `yes`, the non-shifted width of the function argument is not compared with the function input width to filter the violation.

Used By	W486 , STARC05-2.1.3.1
Options	yes, no, comma separated list of Tags
Default Value	no

check_sign_extend

Checks for width mismatch due to sign extension in signed comparisons.

NOTE: *The `check_sign_extend` parameter is valid for the Verilog functionality of the W362 Tag.*

By default, this parameter is set to `no` and the W362 Tag does not check for width mismatch due to sign extension in signed comparisons.

Set this parameter to `yes` or W362 to check for width mismatch due to sign extension in signed comparisons.

Used By	W362
Options	yes, no, W362
Default Value	no

check_static_value

Enables the specified tags to report a violation for static expressions and non-static expressions that contain static expressions.

By default, this parameter is set to `no`. In this case, the specified Tags do not report a violation for static expressions and non-static expressions that

contain static expressions.

In addition, the parameter enables the specified tags to report a violation for static expressions and non-static expressions that contain static expressions.

Other possible values are `only_const` and `only_expr`.

The following table describes the options supported by the `check_static_value` parameter:

Parameter Value	Tag Behavior
<code>only_const</code>	Tags report violations for cases that involve static expressions. For example: <pre>configure_lint_tag_parameter -tag 164a -parameter check_static_value -value only_const</pre> In the above example, all the affected Tags report violations for cases that involve static expressions. Specify the value followed by the Tag names to enable the specified Tags to report violations for this parameter value. For example: <pre>configure_lint_tag_parameter -tag 164b -parameter check_static_value -value only_const,W164a,W116</pre> In the above example, the <i>W164a</i> and <i>W116</i> Tags report violations for cases that involve static expressions.
<code>only_expr</code>	Tags report violations for non-static expressions that contain a static part. Specify the value followed by the Tag names to enable the specified Tags to report violations for this parameter value. For example: <pre>configure_lint_tag_parameter -tag STARC05-2.10.3.2b check_static_value -parameter check_static_value -value only_expr</pre> In the above example, the <i>STARC05-2.10.3.2b</i> Tag reports a violation for non-static expressions that contain a static part.
<code>only_static</code>	Tag will not report violation for expressions, which dose not have any static part.
<code>yes</code>	All the affected Tags report violations for all cases of width mismatch, involving static expressions and non-static expressions that contain a static part.

Parameter Value	Tag Behavior
<Tag_list>	Specified Tags report violations for all cases of width mismatch, involving static expressions and non-static expressions that contain a static part.
no	None of the Tags report violations for static expressions or non-static expressions that contain static expressions.
Setting different values for Tags	This parameter can be set to different value for different Tags, For example: <pre>configure_lint_tag_parameter -tag STARC05-2.10.6.1 -parameter check_static_value -value only_expr -tag STARC05-2.10.3.2b -parameter check_static_value -value only_const</pre> This sets the parameter to: • <code>only_expr</code> for the W116 and W362 Tags • <code>only_const</code> for the W164b and STARC05-2.10.6.1 tags • <code>yes</code> for the W484, W164a, and STARC05-2.10.3.2b tags
Used by	W164a , W164b , W116 , W362 , W484 , W416 , STARC05-2.10.3.1 , STARC05-2.10.3.2a , STARC05-2.10.3.2b , STARC05-2.10.3.2b , STARC05-2.10.6.1
Options	yes, no, only_const, only_expr, only_static, comma-separated list of Tags
Default value	no

Consider the following example:

```
module TEST();
wire [7:0] sig1, sig2;

assign sig1 = 8'hFF + 2'b10; //CASE A
assign sig1 = sig2 + 8'hFF;  //CASE B

endmodule
```

Assume that you want to specify the parameter only for the [W164a](#) Tag. In the above example, if the `check_static_value` parameter is set to `yes` or

Tag name, the [W164a](#) Tag reports a violation in both CASE A and CASE B.

checksynreset

Specifies whether the [W392](#) Tag flags or ignores synchronous resets.

By default, the `checksynreset` Tag parameter is set to `yes` and the [W392](#) Tag considers synchronous resets for Tag checking.

You can set the value of the `checksynreset` Tag parameter to `yes` (to consider synchronous resets) or to `no` (to ignore synchronous resets).

Used by	W392
Options	yes, no
Default value	yes

check_sequential

Specifies whether the [W71](#) Tag checks sequential block for missing default.

By default, the `check_sequential` Tag parameter is set to `no` and the [W71](#) Tag does not check for missing default in case construct inside sequential block.

You can set the value of the `check_sequential` Tag parameter to `yes` (to see violations of sequential block) or to `no` (to ignore violations of sequential block).

Used by	W71
Options	yes, no
Default value	no

check_temporary_flop

Specifies whether the [W336](#) Tag reports a violation for temporary flip-flops.

By default, the `check_temporary_flop` parameter is set to `no` and the [W336](#) Tag does not report violations for temporary flip-flops.

Set the value of the `check_temporary_flop` parameter to `yes` to report temporary flip-flops.

Used by	W336
Options	yes, no
Default value	no

check_unsign_overflow

Specifies whether the [W164a](#), [W164b](#), and [STARC05-2.10.3.2b](#) tags suppress overflow in unsigned signals due to sign extension.

By default, this parameter is set to `no`. This indicates the tag suppresses the overflow when sign extension is used for unsigned signals in addition or subtraction operation.

If you set this parameter to `yes` or `<tag-name>`, the Tag does not suppress the overflow when signextension is used for unsigned signals in addition or subtraction operation.

NOTE: *Irrespective of the value of this parameter, the tag suppresses the overflow when sign extension is used for signed signals in addition or subtraction operation.*

Used by	W164a , W164b , STARC05-2.10.3.2b
Options	yes, no, <code><tag-name></code>
Default value	no

CHECK_Z_ASSIGN

Specifies whether the tristate checks will report violations on Z assignment cases in [TristatePort-ML](#), [W438](#), [W541](#), [TristateSig-ML](#), and [UseMuxBusses](#) tags.

By default, this parameter is set to 0. This indicates that the tags will not report violations on Z assignment cases.

If you set this parameter to 1, the tags will report violations on Z assignment cases for the specified tags.

Used by	TristatePort-ML , W438 , W541 , TristateSig-ML , and UseMuxBusses
Options	0, 1
Default value	0

concat_width_nf

Enables the W164a Tag to use new algorithm to calculate the width of concatenations ignoring the self-determined nature of concatenation items.

By default, this parameter is set to no. Set this parameter to yes to enable the Tag to calculate the width of concatenations ignoring the self-determined nature of the concatenation items.

Used by	W164a
Options	yes, no
Default value	no

considerInoutAsOutput

Enables the [W122](#) Tag to consider the inout port as output port. Therefore, the signal, which is passed to that port is considered as set.

By default, the value of the *considerInoutAsOutput* parameter is no. In this case, the [W122](#) Tag considers the inout port as both input and output. Therefore, the signal that is passed to the inout port is considered as both read and set.

NOTE: *This parameter is applicable only for VHDL.*

Set the value of the parameter to yes to consider the inout port as output port.

Used by	W122
Options	yes, no
Default value	no

control_sig_detection_nf

Defines the control signal detection to be done by the specified tags.

By default, the `control_sig_detection_nf` parameter is set to `no`.

If you set this parameter to `yes`, the Tag treats a signal to be a control signal only if it is used in a conditional operator or control binary operator within the scope of currently processing statement. The Tag does not consider the signal to be a control signal even if it is used as a control signal in other places of the module (that is, the statements other than the current statement).

Used by	W164a , STARC05-2.10.3.2b, STARC05-2.10.6.1
Options	yes, no
Default value	no

datapath_or_control

Specifies whether the [W164a](#) Tag should check the specified type of signals.

By default, the `datapath_or_control` parameter is set to `no` and the Tag reports violations for all type of signals.

Set this parameter to `datapath`, `control`, or `all` to specify the type of signals to be checked.

The following are the details of the possible values of this parameter:

- **no**: This is the default value. Existing violation message are reported.
- **all**: The [W164a](#) Tag checks for all type of signals (both datapath and control signals) at the LHS of assignment, which is the same as the existing behavior. The violation messages mention the type of the signals on the LHS of the assignment (datapath or control).
- **datapath**: Only datapath signals are checked and violation messages mention the type of the signals (datapath signal) at the LHS of the assignment.

- **control:** Only control signals are checked and violation messages mention the type of signals (control signal) at the LHS of the assignment.

Used by	W164a
Options	<i>datapath, control, all, no</i>
Default value	no

do_not_run_W71

Specifies whether to run [W71](#) (Verilog) Tag or not. This parameter is provided to block duplicate violation in case the [W69](#) and [W71](#) Tags are run in mixed mode (Verilog, VHDL).

By default, the `do_not_run_W71` Tag parameter is set to `no` and [W71](#) (Verilog and VHDL) Tag is run.

You can set the value of the `do_not_run_W71` parameter to `yes` to block [W71](#) (Verilog) Tag.

Used by	W71
Options	yes, no
Default value	no

define_module_list

Enables the [PortConnectToFixedVal-ML](#) tag to check for module instances, which are specified by this parameter.

By default, the value of the parameter is `' '`. In this case, the tag checks for all the module instances.

Set the value of the parameter to a comma-separate list of modules to enable the [PortConnectToFixedVal-ML](#) tag to check for only the specified module instances.

NOTE: The `define_module_list` parameter supports comma-separated regular expression list.

Used by	PortConnectToFixedVal-ML
Options	comma-separated list of module names. Also support regular expressions
Default value	' '

DISABLE_CLOCK_SORTING

The `DISABLE_CLOCK_SORTING` parameter stops sorting of clocks from CDC API.

NOTE: *The `DISABLE_CLOCK_SORTING` parameter is valid for both Verilog and VHDL.*

By default, this parameter is set to 0 and sorting of clocks from the CDC API is enabled.

Set this parameter to 1 to disable sorting of clocks from CDC API. By default, sorting of clocks is enabled.

Used By	W391 , STARC05-1.4.3.1
Options	0 and 1
Default Value	0

disable_rtl_deadcode

The `disable_rtl_deadcode` parameter specifies whether to report a violation for disabled code in loops and conditional (`if` condition, ternary operator) statements.

NOTE: *The `disable_rtl_deadcode` parameter is valid for both Verilog and VHDL.*

By default, this parameter is set to `no` and the respective Tags report a violation for disabled code in loops and conditional (`if` condition, ternary operator) statements.

Set this parameter to `yes` to disable violations for disabled code in loops and conditional (`if` condition, ternary operator) statements.

Used By	W116 , W164a , W164b , W362 , W484 , W486 , W456a , W416 , STARC05-2.10.3.2a , STARC05-2.10.3.2b , STARC05-2.10.3.2c , STARC05-2.10.6.1
Options	yes, no, <i><Tag-name></i>
Default Value	no

disable_signal_usage_report

Use this parameter to disable the generation of data for the specific Tag for the SignalUsageReport report.

By default, the value of this parameter is no. In this case, all Tag related data is reported in the SignalUsageReport report.

Used By	W241 , W528
Options	yes, no, <i><Tag-name></i>
Default Value	no

dump_array_bits

Specified whether the [W123](#) Tag should report the multi-dimensional signals/bits information.

By default, the dump_array_bits parameter is set to no.

Set this parameter to yes, to enable the W123 Tag to include multi-dimensional signals/bits information in the violation message.

This parameter is applicable to **VHDL** only.

Used By	W123
Options	yes, no
Default Value	no

effort_simloop_level

Specifies the amount of efforts put by the [sim_loop01](#) tag in detecting a simulation loop

Use this parameter to restrict the traversal through a particular scalar net while detecting a simulation loop. By default, a net is traversed 200 times during loop detection. When you set this parameter to a higher value, the tag might require higher run-time.

Used by	sim_loop01
Options	Positive integer value
Default value	200

fast

Specifies to suppress the synthesis of the source RTL description. If this parameter is set, the following Tags are not run:

W18	W287a	W287b	W323	W336	W391	W392
W396	W401	W402	W402b	W415	W438	W448
W541	STARCO5-2.3.6.1					

By default, the `fast` Tag parameter is not set. Therefore, the source RTL description is synthesized and the above tags are run. You can set the value of the `fast` tag parameter to `Yes` to (suppress synthesis) or to `No` (to synthesize the source RTL description).

For the [W414](#) and [W428](#) Tags, the `fast` Tag parameter is not set by default and the synthesis version of these Tags is run. Set the value of the `fast` parameter to `Yes` to suppress the synthesis and execute the RTL version of the [W414](#) and [W428](#) Tags.

Used by	W18 , W287a , W287b , W323 , W336 , W391 , W392 , W396 , W401 , W402 , W402b , W415 , W438 , W448 , W541 , STARC05-2.10.6.1
Options	yes, no
Default value	no

flag_reset_type

Specifies if the [RegisterInfo-ML](#) tag reports violations for sync-reset/ async-reset/ none-reset type registers in the design.

By default, the `flag_reset_type` parameter is set to "" and does not report violations regarding report data.

Set the parameter to `sync/ async/ none` to report violations if the combination has the following:

- `sync` - Reports registers on flops/latches with synchronous resets.
- `async` - reports registers on flops/latches with asynchronous resets.
- `none` - reports registers on flops/latches without synchronous or asynchronous resets.

Used by	RegisterInfo-ML
Options	sync, async, none
Default value	" "

flag_unconnected_out_flops

By default, parameter is set to no, and the [STARC05-3.3.1.4b](#) tag does not report violation when flop output is unconnected.

If parameter is set to yes, the [STARC05-3.3.1.4b](#) tag will flag violation when flop output is unconnected.

Used by	STARC05-3.3.1.4b
Options	yes, no
Default value	no

force_handle_shift_op

Enables the [W164a](#) Tag to honor the `handle_shift_op` parameter even if the `nocheckoverflow` parameter is set.

By default, the `force_handle_shift_op` parameter is set to `no`.

Set this parameter to `yes` to enable the `W164a` Tag to honor [handle_shift_op](#) parameter value even if the [nocheckoverflow](#) parameter is set to `yes`.

Used By	W164a
Options	yes, no
Default Value	no

handle_case_select

Enables you to specify the maximum width of case selector that the supported tag should process.

You can specify any integer value as an input to the parameter. By default, the parameter is set to 64. Set the value to -1 to enable the supporting Tags to process case statements with any case selector width.

If the width of a case selector is greater than the value specified for the `handle_case_select` parameter, the Tags ignore the case statement and reports a violation.

Used by	W69 , W71 , SetBeforeRead-ML , STARC05-2.8.1.4
Options	Any integer value
Default value	64

handle_equivalent_drivers

Enables the [W415](#) and [Rule_32](#) tags to report violations when a net is simultaneously driven by two identical instances of two input gates of type OR, NOR, AND, or, NAND and both instances are driven by same input signals.

By default, the value of the parameter is set to no. In this case, the Tag will not report violation for above scenario.

Set the value of the parameter to yes to report violations when a net is simultaneously driven by two identical instances of basic two input gates of type OR/NOR/AND/NAND and both instances are driven by same input signals.

For example, consider the following assignment statements:

```
assign a = b & c;
```

```
assign a = b & c;
```

The W415 Tag will report violation for above assignments when the value of the *handle_equivalent_drivers* parameter is set to yes.

Used By	W415 , Rule_32
Options	yes, no
Default Value	no

handle_large_expr

Specifies whether the [W484](#) and STARC05-2.10.6.1 tags should handle large expressions.

By default, if the RHS expression is a binary expression and each term is a concatenation expression padded with zero bits, the Tag checks for overflow while considering zero padded bits. That is, violation is not reported if the actual width of the expression can be accommodated in zero padded bits.

When the *handle_large_expr* parameter is set to yes, the specified tag limits this checking for expressions containing up to 25 terms and for larger expressions, the LRM width is considered.

Used By	W484 , STARC05-2.10.6.1
Options	yes, no
Default Value	no

handle_lrm_param_in_shift

Specifies whether the [W164a](#) Tag should honor [use_lrm_width](#) parameter in the width calculations of shift expressions.

By default, the `handle_lrm_param_in_shift` parameter is set to `no`.

Set this parameter to `yes` to enable the [W164a](#) Tag to honor the [use_lrm_width](#) parameter in the width calculations of shift expressions.

Used By	W164a
Options	yes, no
Default Value	no

handle_shift_op

Specifies whether to consider shifted or non-shifted width of a shift expression.

By default the `handle_shift_op` parameter is set to `no`. In this case, both the shifted and non-shifted widths of a left or right shift expression are compared and no violation is reported if any of these widths matches the LHS expression. But, the Tag does not calculate shifted width, if the RHS of the shift expression is non-static.

NOTE: *This parameter is effective when the `nocheckoverflow` parameter is set to `no`.*

The `handle_shift_op` parameter can be set to `shift_left`, `shift_right`, `shift_both`, `no_shift`, `no`, `no_shift_forced`, or comma separated list of Tag names, to compare shifted and non-shifted widths of a left or right shift expression.

The behavior of these options is as follows:

- **shift_left**: Shifted width is considered for those left-shift expressions that have the RHS of shift expression as either static or non-static. For example:

```
configure_lint_tag_parameter -tag W110 -parameter  
handle_shift_op -value shift_left
```

In this case, the parameter will be set to `shift_left` for all the Tags that use this parameter.

Or

```
configure_lint_tag_parameter -tag W164a, W110 -parameter  
handle_shift_op -value shift_left
```

In this case, the parameter will be set to `shift_left` for the specified Tags only.

- **shift_right**: Shifted width is considered for those right-shift expressions that have the RHS of shift expression as static. For example:

```
configure_lint_tag_parameter -tag W110, W164a -parameter  
handle_shift_op -value shift_right
```

Or

```
configure_lint_tag_parameter -tag W110, W164a -parameter  
handle_shift_op -value shift_left
```

- **shift_both**: Shifted width is considered for both left and right shift expressions, as stated above for parameter values `shift_left` and `shift_right`. For example:

```
configure_lint_tag_parameter -tag W164a -parameter  
handle_shift_op -value shift_both
```

Or

```
configure_lint_tag_parameter -tag W164b,W110 -parameter  
handle_shift_op -value shift_both
```

- **Comma separated list of Tags**: This sets the value of the parameter to `shift_both` for the specified Tags. For example:

```
configure_lint_tag_parameter -tag W164b,W110 -parameter  
handle_shift_op -value shift_both
```

- **no_shift**: Non-shifted width is considered for both left and right shift expressions, that is, the width of the LHS of shift expression is considered. This is applicable only for those shift expressions that have the LHS as non static and the RHS as static. For example:

```
configure_lint_tag_parameter -tag W16b -parameter  
handle_shift_op -value no_shift
```

Or

```
configure_lint_tag_parameter -tag W164b,W164a -parameter  
handle_shift_op -value no_shift
```

- **no**: Default behavior.
- **no_shift_forced**: Checks for the non-shifted width of the shift operator regardless of the staticness of the RHS and LHS expression of the shifted operation.
- This parameter can also be set to different values for different Tags, for example:

```
configure_lint_tag_parameter -tag W110,W164b -parameter  
handle_shift_op -value shift_left -tag W164a -parameter  
handle_shift_op -value shift_both
```

This sets the parameter to `shift_left` for the W110 and W164b Tags and `shift_both` for the W164a Tag.

Used By	W164a , W164b , W110
Options	shift_left, shift_right, shift_both, no_shift, no, no_shift_forced, Tag name
Default Value	no

handle_static_caselabels

Specifies whether the [W263](#) Tag should ignore static case labels having less width than the case selector.

By default, the `handle_static_caselabels` parameter is set to `no`.

Set this parameter to `yes` to ignore violations for static case labels, which are of less width than the width of case selector when case labels are

parameters.

Used By	W263
Options	yes, no
Default Value	no

handle_zero_padding

Performs leading zero expansion and truncation of RHS of an assignment. This is performed only if the RHS of the assignment is static. This parameter also considers the width specified by the width specifier of sized-base numbers when handling zero padding.

Used By	W110 , W164a , W164b , W362 , W416
Options	yes, no
Default Value	no

hier_path_length_limit

Specifies the maximum limit of length of hierarchical path name.
By default, the `hier_path_length_limit` parameter is set to 1024.
Set this parameter to a positive integer value to define the maximum limit of the length of a hierarchical path name.

Used by	HierarchicalPathLengthCheck
Options	Positive integer number
Default value	1024

ignore_auto_function_return

Allows the [W499](#) Tag to ignore name of automatic functions to be set in all branches of function body. That is, if this parameter is set while the function is automatic, and all other outputs are assigned in all branches, then no violation is reported.

By default, the value of this parameter is no. In this case, the [W499](#) Tag reports the automatic function name when the return-by-function-name is not specified in all the branches in the function body.

Used By	W499
Options	yes, no
Default Value	no

ignore_below_line

Specifies whether the [AlwaysComment](#), [PortComment](#), [SignalComment](#), and [VariableComment](#) tags should ignore the comment below the RTL construct.

By default, the `ignore_below_line` tag parameter is set to `no`.

Set the value of the parameter to `yes` to ignore the comment following (next line) the RTL construct.

Used by	AlwaysComment , PortComment , SignalComment , Variable-Comment
Options	yes
Default value	no

ignore_cell_list

Specifies whether the [InstName](#) tag should report violations for the specified cell instances.

By default, the `ignore_cell_list` parameter is set to `""`.

Set this parameter to a comma-separated list of PERL regular expressions to provide a list of cell names and enable the rule to not report violations for the cell instances.

Used by	InstName
Options	comma-separated list of PERL regular expressions
Default value	""

ignore_bitwiseor_assignment

Specifies whether the [W415a](#) and [LINT_MULTIASSIGN_BLOCKING_SIG](#) tags report violations for a bitwise or assignment inside the for loop that is assigned multiple times on the same line.

By default, the value of the `ignore_bitwiseor_assignment` is set to `no` and the Tags report a violations for a bitwise or assignment inside the for loop that is assigned multiple times on the same line.

Set the value of the parameter to `yes` to not to report violations in such cases.

Used by	W415a , LINT_MULTIASSIGN_BLOCKING_SIG
Options	yes, no
Default value	no

ignore_nonstatic_leftshift

Specifies whether the [W164a](#) and [W164b](#) tags should ignore assignments where the RHS is a left shift operator.

By default, the `ignore_nonstatic_leftshift` parameter is set to `no`.

Set this parameter to `yes` enable the tags to ignore assignments where the RHS is a left shift operator and both the operands are non-static.

Used by	W164a , W164b
Options	yes, no
Default value	no

ignore_cond_having_identifier

Enables the specified tag to ignore the constants defined by parameters, localparams or constant integers.

By default, the value of the `ignore_cond_having_identifier` is set to `no` and the specified tag considers constants defined by parameters, localparams or constant integers, for Tag checking.

Set the value of this parameter to `yes` to not report violation for such constants.

Used by	W159, AlwaysFalseTrueCond-ML
Options	yes, no
Default value	no

IGNORE_CONDITIONAL_EXPRESSIONS

Use this parameter to ignore conditional expressions with logical operators. The default value of the parameter is `no`.

Used by	W576
Options	yes, no
Default value	no

ignore_counter_with_same_width

Enables the [W164a](#) tag to not report violation for cases where any constant value added/subtracted from different signal on both LHS and RHS with same width is treated as a counter.

Used by	W164a
Options	yes, no, only_by_one
Default value	no

ignore_equivalent_drivers

Ignores violation for a net, which is:

- Driven by two identical instances of basic two input gates of type OR/NOR/AND/NAND.

- both instances are driven by same input signals.

By default, the value of the parameter is yes. In this case, the Tag ignores violation for such nets.

Set the value of the parameter to no to report violation for such nets as stated above.

For example, the W415 Tag does not report violation for the following scenario when the value of the `ignore_equivalent_drivers` parameter is set to yes.

```
assign a = b & c;
```

```
assign a = b & c;
```

Used by	W415
Options	yes, no
Default value	yes

ignore_forloop_indexes

Specifies whether the [W116](#) Tag should ignore the `for` expressions that contain index variables of `for` loops.

By default, the value of the `ignore_forloop_indexes` is set to no and the [W116](#) reports the `for` expressions that contain index variables of `for` loops.

Set the value of this parameter to yes to not report violation for such cases.

Used by	W116
Options	yes, no
Default value	no

ignore_genvar

Specifies whether the [W121](#) Tag should ignore violations for variable shadowed by `genvar` variables.

By default, the value of the `ignore_genvar` parameter is set to `no` and the *W121* Tag reports violations for variables shadowed by `genvar` variables.

Set the value of this parameter to `yes` to not report violation for such cases.

Used by	W121
Options	yes, no
Default value	no

ignore_generatefor_index

Specifies whether the *W226* Tag should ignore for `case` statements inside `generte-for` blocks if the case selector is a loop variable of `generate-for` loop.

By default, the value of the `ignore_generatefor_index` parameter is set to `no`.

Set the value of this parameter to `yes` to ignore violations for `case` statements inside `generte-for` blocks, if the case selector is a loop variable of `generate-for` loop.

Used by	W226
Options	yes, no
Default value	no

Example

ignore_greybox_drivers

Enables the *W415* and *Rule_32* tags to do not consider greybox as a valid driver,

By default, the value of the parameter is `no`. Default value is `no`. If the value of the parameter is set to `yes`, the *W415* Tag will not consider greybox as a valid driver.

Used by	W415, Rule_32
Options	yes, no
Default value	no

ignore_hier_scope_var

Specifies whether the [W123](#) Tag should ignore user-defined data types.

By default, the value of the `ignore_hier_scope_var` is set to `no` and the [W123](#) Tag reports all user-defined data types.

Set the value of this parameter to `yes` to not report any violation for user-defined data types. This parameter is applicable for Verilog only.

Used by	W123
Options	yes, no
Default value	no

ignore_in_ports

Specifies whether the [W111](#) considers input and inout ports for Tag checking.

By default, the value of the `ignore_in_ports` is set to `no` and the [W111](#) Tag considers input and inout ports for Tag checking.

Set the value of the parameter to `yes` to ignore input and inout ports for Tag checking.

Used by	W111
Options	yes, no
Default value	no

ignore_inout

Specifies whether the specified tag should ignore the inout ports/terminals.

By default, the value of the `ignore_inout` is set to `no` and the W210 Tag considers inout ports for Tag checking.

Set the value of this parameter to `yes` to not report any violation for the inout terminals.

Used by	W210 , UndrivenInTerm-ML , RegInput-ML , STARC05-2.5.1.5a
Options	yes, no
Default value	no

ignore_integer_const_variables

Use this parameter to report violations for constant case labels of integer type.

Used by	W263
Options	yes, no
Default value	no

ignore_internal_signal

Use this parameter to ignore violation for internal signals.

By default, the value of the parameter is set to `no`. In this case, the [STARC05-2.3.6.1](#) tag reports violations for internal signals.

Used by	STARC05-2.3.6.1
Options	yes, no
Default value	no

ignore_flops

Set the `ignore_flops` parameter to ignore violations where flip-flop is connected to the tri-state logic.

By default, the value of this parameter is set to `no`. In this case, the tag

reports violation for flip-flops, which are connected to the tri-state logic.

Set the value of the parameter to `yes` to ignore violations where flip-flop is connected to the tri-state logic.

Used by	STARCO5-2.5.1.2
Options	yes, no
Default value	no

ignore_fullcase_default

Specifies if the [NoAssignX-ML](#) tag should report a violation for default clauses of a fully covered case.

By default, the `ignore_fullcase_default` parameter is set to `no`.

Set this parameter to `yes` to not report a violation for default clauses of a fully covered case when the `check_xassign_casedefault` parameter is set to `yes`.

Used by	NoAssignX-ML
Options	yes, no
Default value	no

ignore_local_variables

Specifies whether the [W336](#) Tag should ignore variables defined inside the sequential blocks.

By default, the value of the `ignore_local_variables` parameter is set to `no` and the [W336](#) Tag reports violations for variables (left hand side of the assignment) defined inside the sequential block.

Set the value of this parameter to `yes` to ignore violations for such cases.

Used by	W336

Options	yes, no
Default value	no

ignoreModuleInstance

Specifies whether the [W123](#) Tag flags or ignores the unset variables that are only used in instance port mapping.

By default, the `ignoreModuleInstance` Tag parameter is set to `no` and the `W123` Tag flags the unset variables that are also used in instance port mapping.

Set the value of the `ignoreModuleInstance` Tag parameter to `yes` to ignore the unset variables that are only used in instance port mapping.

Used by	W123
Options	yes, no
Default value	no

ignore_nonBlockCondition

Specifies whether the [W415a](#) Tag reports or ignores non-blocking assignments to a variable inside a conditional construct, if the variable is also assigned in a non-blocking manner before the conditional construct.

By default, the parameter is not set and the Tags report conditions.

You can set the value of the `ignore_nonBlockCondition` parameter to `yes` (to ignore such conditions) or to `no` (to flag such conditions).

Used by	W415a
Options	yes, no
Default value	no

ignore_macro_to_nonmacro

Enables the `W121` Tag to ignore name matching for macro to non-macro

names. However, the W121 Tag still checks for the macro to macro name matching.

Used by	W121
Options	yes, no
Default value	no

ignore_multi_assign_in_forloop

Specifies whether the [W415a](#) and [LINT_MULTIASSIGN_BLOCKING_SIG](#) tags checks inside a for loop for multiple assignments on the same line.

When this parameter is set to *Yes*, the Tags do not report a violation for a signal in the for loop that is assigned multiple times on the same line.

By default, the value of the `ignore_multi_assign_in_forloop` parameter is set to `no` and the Tags report a violation for multiple assignments on the same line.

Consider the following example:

```
reg a, b, c;
  for(i = 0; i < 2; i = i + 1)
    begin
      a = a + 1;
    end
```

In the above example, the Tags, in default mode, reports a violation for signal `a` assigned in the for loop. But, when the `ignore_multi_assign_in_forloop` parameter is set to *Yes*, no violation is reported for multiple assignments on the same line in the for loop.

Used by	W415a , LINT_MULTIASSIGN_BLOCKING_SIG
Options	yes, no
Default value	no

ignore_param_type

If set to yes, STARC05-2.1.3.1 tag will ignore the specified type for parameter and take natural width if value of parameter is a constant integer.

Used by	STARC05-2.1.3.1
Options	yes, no, comma-separated list of rules
Default value	no

ignore_pli_tasks_and_functions

Specifies the PLI tasks and functions that should be ignored by the [W213](#) Tag.

By default, the `ignore_pli_tasks_and_functions` parameter is not set.

Set this parameter to a comma-separated list of PLI tasks or functions that should be ignored by the [W213](#) Tag.

Used by	W213
Options	Comma-separated list of PLI task or function names
Default value	""

IGNORE_SYNC_RESET_FOR_ASYNC_RESET_FLOP

Use this parameter to ignore sync set or reset for flops which have async set or reset.

The default value of the parameter is yes.

Used by	W392
Options	yes, no
Default value	yes

ignore_reinitialization

Specifies whether the [W415a](#) or [LINT_MULTIASSIGN_BLOCKING_SIG](#) tags should ignore violations for re-initialization assignments inside the `for` loops.

By default, the `ignore_reinitialization` parameter is not set and the Tags report violations for re-initialization when the [check_initialization_assignment](#) parameter is set to `yes`. Set this parameter to `yes` to ignore violations for re-initialization assignments inside the `for` loops.

Used by	W415a , LINT_MULTIASSIGN_BLOCKING_SIG
Options	yes, no
Default value	no

ignore_scope_names

Specifies whether the [W121](#) Tag should ignore Tag checking on scope names.

By default, the `ignore_scope_names` parameter is not set and the [W121](#) Tag checks for scope names.

Set this parameter to `yes` to ignore Tag checking on scope names.

Used by	W121
Options	yes, no
Default value	no

IGNORE_WITHIN_RANGE_LABELS

Ignores the [W263](#) violation, if case label value lies within the range of case select value.

By default, the value of the parameter is `no`. In this case, the [W263](#) tag reports violation if the case label value lies within the range of case select value.

Set the value of the parameter to `yes` to ignore such violations.

Used by	W263
Options	yes, no
Default value	no

ignoreSeqProcess

Specifies whether the [W122](#) Tag checks inside the sequential process block.

By default, the value of the `ignoreSeqProcess` parameter is set to `no` and the [W122](#) Tag ignores the signals read inside clock condition, but reports violation for the signals read inside other conditions in sequential block.

You can set the value of the parameter to `yes` to ignore all signals read inside sequential process block.

Used by	W122
Options	yes, no
Default value	no

latch_reset

Specifies a list of primary reset port names for the [GatedReset](#), [W392aL](#), [W392bL](#), and [LinkagePort](#) tags.

Used by	GatedReset , W392aL , W392bL , LinkagePort
Options	Name of the reset pin for latches in the design
Default value	Empty string

limit_task_function_scope

Use this parameter to ignore Tag checking if a signal is declared with function/task scope as well as with module scope.

Used by	W121
Options	yes, no
Default value	no

modport_compat

Specifies whether the [W497](#) and [W498](#) tag should evaluate mod ports that are used as hier scope variable in a port connection.

By default, the `modport_compat` parameter is set to no.

Set the value of the parameter to yes to enable the tag to not evaluate mod ports that are used as hier scope variables in a port connection.

Used by	W497 , W498
Options	yes, no
Default value	no

net_count

By default, the value of the parameter is 10 and the specified tag reports only up to 10 nets in a loop in violation.

Specify a positive integer value to this parameter to specify the number of nets that should be printed in violation message.

Used by	CombLoop , STARC05-1.2.1.3
Options	Positive integer value
Default value	10

nocheckoverflow

Specifies whether the related Tags check the bit-width as per LRM.

By default, the `nocheckoverflow` Tag parameter is set to no, and the affected Tags do not check the bit-width as per LRM.

You can set the value of *nocheckoverflow* Tag parameter to *yes* to check the bit-width as per LRM. You can also specify the Tag name as an argument to the *nocheckoverflow* Tag parameter to enable LRM-based checking for that particular Tag.

Used by	W116 , W164a , W164b , W263 , W486 , W110 , W263 , W362 , W416 AsgnOverflow-ML , ImproperRangeIndex-ML , LINT_IMPROPER_RANGE_INDEX STARC05-2.1.3.1 , STARC05-2.2.3.1 , STARC05-2.8.1.6 , STARC05-2.10.3.1 , STARC05-2.10.3.2a , STARC05-2.10.3.2b , STARC05-2.10.3.2c , STARC05-3.2.3.2
Options	yes, no, Tag name
Default value	no

not_used_signal

Specifies a comma-separated name list of signals to be ignored by the [W528](#) Tag.

By default, the value of this parameter is *nil*, which means that Tag checking by the Tag will be done for all the signals.

You can specify full names or regular expressions. The following example matches the signal named *data_signal* and all signal names starting with *bus_*:

```
"data_signal,bus_*
```

Used by	W528
Options	comma-separated list of signals
Default value	nil

only_apply_on_leaf_cell

Use this parameter to enable [UnloadedOutTerm-ML](#) tag to check for LIB cells only.

By default, the value of the parameter is *no*. In this case, the tag does not

check only for LIB cells. It checks for all unloaded but driven output terminal of an instance.

Used by	UnloadedOutTerm-ML
Options	yes, no
Default value	no

process_complete_condop

Enables the Tag checking on both the operands of the condop assignment.

By default, the value of this parameter is no.

Set the value of the parameter to yes to enable the Tag checking on both operands of the condop assignment.

NOTE: *The process_complete_condop parameter is valid even when the value of the nocheckoverflow parameter is set to yes.*

Used by	W164a , W164b
Options	yes, no
Default value	nil

report_all_connections

Specifies whether the [W156](#) tag should report a violation for all reverse order buses in the same module.

By default, the report_all_messages parameter is set to no.

Set this parameter to yes to report a violation for all reverse order buses in the same module.

Used by	W156
Options	yes, no
Default value	no

report_all_hier

Enables the tag to report violation for lower-level hierarchies as well. The default value is no. In this case, the [ConstantInput-ML](#) tag reports violation for only higher-level hierarchies.

Used by	ConstantInput-ML
Options	yes, no
Default value	no

report_all_messages

Specifies whether the [W122](#) and [W502](#) Tags report the bit index information. By default, the `report_all_messages` parameter is set to no.

Set this parameter to yes or Tag name(s) to report the bit information in the violation messages of the [W122](#), [W456](#), and [W502](#) Tags.

NOTE: *If this parameter is set, the violation messages are also modified and old waivers do not work. See the specific Tag messages for details.*

Used by	W122 , W502
Options	yes, no, comma separated list of Tags
Default value	no

report_all_unpacked

Enables the [UnpackedStructUsed-ML](#) tag to report violations for the following scenarios, when the value of the `report_all_unpacked` parameter is set to yes:

- Unpacked typedef array definitions.
- Unpacked members within structs.
- Unpacked struct, and array definitions within packages.
- Declarations of unpacked structs and arrays

By default, the value of the parameter is set to no. In this case, the tag does not report violation for the above cases.

Used by	UnpackedStructUsed-ML
Options	yes, no
Default value	yes

report_at_constant

Enables the [LatchDataConstant](#) tag to report violations for file name and line number where the data net of a latch is connected to a constant.

By default, the value of the parameter is set to no. In this case, [LatchDataConstant](#) tag reports a violation for file name and line number where the latch is specified in the code.

Set the value of the parameter to yes to report violations for file name and line number where the data net of a latch is connected to a constant.

Used by	LatchDataConstant
Options	yes, no
Default value	no

report_blackbox_inst

Specifies whether the [W110](#) Tag reports violation for port width mismatch for black box instances.

By default, the value of this parameter is set to no. In this case, the W110 Tag does not report violation for port width mismatch for black box instances.

Set the value of the parameter to yes to report violation for port width mismatch for black box instances.

Used by	W110

Options	yes, no
Default value	no

report_clock_reset_loops

Specifies whether the [CombLoop](#) tag should report violations for loops in clock and reset tree.

By default, the *report_clock_reset_loops* tag parameter is set to no.

Set the *report_clock_reset_loops* tag parameter to yes to report violations only for loops in clock and reset tree.

Used by	CombLoop
Options	yes, no
Default value	no

report_declarations

Enables the [NegativeIndex-ML](#) tag to report violations for declarations with negative indexes.

By default, the value of the parameter is yes. In this case, the tag does not report violations for declarations with negative indexes.

Set the value of the parameter to yes to report violations for such cases.

Used by	NegativeIndex-ML
Options	yes, no
Default value	no

report_global_param

Enables the [W175](#) tag to report violations for unused global parameter declarations.

By default, the value of the parameter is no. In this case, the tag does not report violation for unused global parameter.

Set the value of the parameter to yes to report violation for unused global parameter.

Used by	W175
Options	yes, no
Default value	no

report_hierarchy

Specifies whether the [W287b](#) tag should report RTL module name and hierarchy in the violation message.

By default, the parameter is set to no.

Set the parameter value to yes to report rtl module name and hierarchy in the violation message.

Used by	W287b
Options	yes, no
Default value	no

REPORT_LOOP_VAR_EXPR

By default parameter is set to no. When parameter is set to yes, the [ImproperRangeIndex-ML](#) tag reports violations for index expressions with loop variable.

When both IGNOREINTEGER and REPORT_LOOP_VAR_EXPR parameters are set to yes, the [ImproperRangeIndex-ML](#) tag reports index expressions with integer loop variables, and doesn't reports index expressions with integer variables, which are not used as loop variables.

Used by	ImproperRangeIndex-ML
Options	yes, no
Default value	no

report_less_severity

By default, the value of the parameter is set to no. In this case, all violations will be reported with Warning severity.

Set the value of the parameter to yes to report parameters, constant and macro conditions with Info severity. Others are reported with Warning severity.

Used by	AlwaysFalseTrueCond-ML
Options	yes, no
Default value	yes

report_nonblocking_in_error

Specifies whether the [W164a](#), [W164b](#), [W164a_a](#), and [W164a_b](#) tags should report violations (with Error severity) for nonblocking assignments.

By default, the `report_nonblocking_in_error` parameter is set to no.

Set the value of the parameter to yes to enable the tags to report violations (with Error severity) for nonblocking assignments.

Used By	W164a , W164b , W164a_a , W164a_b
Options	yes, no
Default Value	no

report_one_hier

Specifies whether the [STARCO5-2.8.2.1](#) tag reports only one violation for multiple hierarchies.

By default, the `report_one_hier` parameter is set to no.

Set the value of the `report_one_hier` parameter to yes to report only one violation for multiple hierarchies.

Used By	STARC05-2.8.2.1
Options	yes, no
Default Value	no

report_only_bitwise

Specifies whether the [W116](#) tag should report violations only for bit wise operations.

By default, the `report_only_bitwise` parameter is set to no.

Set the value of the parameter to yes to report violations only for bit wise operations in the W116 tag.

Used by	W116
Options	yes, no
Default value	yes

report_only_overflow

Specifies whether the [W504](#) Tag reports violations only for integers or constant integers where the width of port expression is greater than the width of port.

By default, the `report_only_overflow` parameter is set to no.

If you set this parameter to yes, the [W504](#) Tag reports violations only for integers or constant integers where the width of port expression is greater than the width of port.

For example:

```
module top ();
  blk1 u_blk1 (.a(3)); //Violation with report_only_overflow
  blk2 u_blk2 (.b(3)); //No violation with report_only_overflow
endmodule
module blk1 (input a);
  ...
endmodule
```

```
endmodule
module blk2 (input [2:0] b);
...
endmodule
```

However, there are some special cases while calculating width of port expression:

- If port expression contains shift operator, the width of expression will be width of LHS operand. For example:

```
(32 >> 2) - width of 32
```

- If port expression contains arithmetic operators (+, -, *, /, %), the width of expression will be maximum width out of LHS and RHS operands. For example:

```
(4+16) - width of 16
```

Used By	W504
Options	yes, no
Default Value	no

report_cast

Used to specify whether the newly introduced message should be reported instead of the previous one.

The W372 Tag specifically reports for '\$cast' PLI tasks or functions if this parameter is set to yes.

Used by	W372
Options	yes, no
Default value	no

report_if_blocks_only

Specifies whether the [W193](#) Tag reports the use of semicolon with the if, else, and else-if statements.

By default, the value of the `report_if_blocks_only` parameter is set to `no`.

Set the value of this parameter to `yes` to report violations only when a semicolon is used with the `if`, `else`, and `else-if` statements.

Used by	W193
Options	yes, no
Default value	no

report_inter_nba

Specifies whether the [W280](#) Tag reports the inter-nonblocking assignment delay.

By default, the value of this parameter is set to `no`. In this case, the [W280](#) Tag does not report inter-nonblocking assignment delay.

Set the value of the parameter to `yes` to report violation for such cases.

Used by	W280
Options	yes, no
Default value	no

report_port_net

Specifies if the [W154](#) tag reports nets/ports declared without a type.

By default, the `report_port_net` parameter is set to `no`.

If you set this parameter to `yes`, the [W154](#) tag reports nets/ports declared without a type.

Used By	W154
Options	yes, no
Default Value	no

report_single_hierarchy

Enables the [SelfDeterminedExpr-ML](#) tag to block/allow same violations for multiple hierarchies.

By default, the value of the parameter is no. In this case, the tag reports same violations for multiple hierarchies.

Set the value of the parameter to yes to disable reporting of same violations for multiple hierarchies.

Used by	SelfDeterminedExpr-ML
Options	yes, no
Default value	no

report_single_violation

Use this parameter to report one violation when signal name, instance and hierarchy are the same.

The default value of the parameter is no.

Used by	W287a
Options	yes, no
Default value	no

report_sync_rst_only

Enables the [STARC05-1.1.5.2b](#) tag to report synchronous reset names.

By default, the value of the report_sync_rst_only parameter is set to no. In this case, the tag does not report for synchronous reset names.

Set the value of the parameter to yes to enable the tag to process synchronous reset name only to validate if the names are matching the format specified using the starc2005_reset_string parameter.

Used by	STARC05-1.1.5.2b

Options	yes, no
Default value	no

report_to_integer

Enables the [ValueSizeOverflow-ML](#) to report violation for cases where `to_integer` function is parsed with an argument whose bit width is larger than 32.

Used by	ValueSizeOverflow-ML
Options	yes, no
Default value	no

reportLibLatch

Specifies whether latches inferred from both .sglib and .lib libraries are reported, as checked by the specified tag.

By default, the `reportLibLatch` tag parameter is set to `no`, and the tags specified in the table below do not report library latches.

Set the `reportLibLatch` Tag parameter to `yes` to report library latches.

Set the value of the parameter to `only` to report violations only for the latches inferred through lib cells.

Used by	W18 , InferLatch , STARC05-2.2.1.2
Options	yes, no, only
Default value	no

reportconstassign

The `reportconstassign` parameter specifies whether the [W116](#) Tag checks

for constants whose width is less than the operand.

NOTE: *The `reportconstassign` parameter is valid for Verilog only.*

By default, the `reportconstassign` parameter is set to `no` and the `W116` Tag does not report a violation for constants whose width is less than the other operand.

If you set this parameter to `yes`, the `W116` Tag reports a violation in such cases.

Used By	W116
Options	yes, no
Default Value	no

reportundrivenout

Specifies whether the [RegOutputs](#) Tag should flag unconnected/undriven output ports.

By default, the `reportundrivenout` Tag parameter is set to `yes` and the `RegOutputs` Tag flags unconnected/undriven output ports.

You can set the `reportundrivenout` Tag parameter to `no` to allow unconnected/undriven output ports in the design.

Used By	RegOutputs
Options	yes, no
Default Value	yes

reset_gen_du

Enables the [LINT_RSTGEN_AVOID_SEQSTMT](#) tag to identify reset generator modules.

Also, supports regular expressions.

Used By	LINT_RSTGEN_AVOID_SEQSTMT

Options	Comma separated list of reset-generator-module names
Default Value	" "

reset_gen_module

Enables you to specify module or cell names for which the [W402b](#) tag will not report violation if the reset signal is gated by a instance of a given module.

Used By	W402b
Options	yes, no
Default Value	yes

report_semicolon

Specifies if the [W193](#) and [W164a](#) tags reports the use of extra semicolon.

By default, the `report_semicolon` parameter is set to `no`.

If you set this parameter to `yes`, the *tags* report the use of extra semicolon.

Used By	W193 , W164a
Options	yes, no
Default Value	no

report_struct_name_only

By default, Tag [W123](#) gives the violation messages for each violation bit of the struct variables.

Set the value of the parameter to `yes` to enable the W123 Tag to report one violation for a struct variable.

NOTE: *this parameter is applicable only for Verilog.*

Used by	W123
Options	yes, no
Default value	no

reportsimilarassgn

Specifies whether the [W415a](#) and [LINT_MULTIASSIGN_BLOCKING_SIG](#) tags should check assignments in which RHS value is the same.

By default, the `reportsimilarassgn` parameter is set to `no` and the Tags do not report violation if the RHS value in an assignment is the same as the RHS value in the previous assignment.

Set the `reportsimilarassgn` Tag parameter to `yes` to report such cases.

Used by	W415a , LINT_MULTIASSIGN_BLOCKING_SIG
Options	yes, no
Default value	no

REPORT_ONLY_SVCONSTRUCT

By default, the value of the parameter is `NULL`. In this case, the tag will report violations for all the system Verilog constructs. Use the `REPORT_ONLY_SVCONSTRUCT` parameter to specify comma-separated system Verilog constructs for which you want to report violation.

Consider the following example:

```
configure_lint_tag_parameter -tag {SVConstruct-ML} -
parameter REPORT_ONLY_SVCONSTRUCT -value {pure, 'typedef' }
```

Used by	SVConstruct-ML
Options	<code>NULL</code> , comma-separated system Verilog constructs
Default value	<code>NULL</code>

RESETNET

Use this parameter to read the resets specified using the SGDC or SDC commands. Specify the value of the reset SGDC command as an input to this parameter (use space-separated values in case of multiple commands).

For example, for the following SGDC command:

```
reset -name top.reset top.mid1.rst
```

Specify the RESETNET parameter as shown below:

```
configure_lint_tag_parameter -tag
resetPinConnectedToResetNet -parameter RESETNET -value {
top.reset top.mid1.rst2 } -goal <goal_name>
```

Used by	resetPinConnectedToResetNet
Options	space-separated list of reset pin names
Default value	" "

runcapa

(Optional) Specifies whether the [CAPA-ML](#) tag should run.

By default, the runcap tag parameter is set to no and the CAPA-ML tag is disabled.

Set the runcap tag parameter to yes to enable the CAPA-ML tag.

Used by	CAPA-ML
Options	yes, no
Default value	no

set_message_severity

Sets the severity of violation messages reported by the [W210](#) Tag.

By default, the set_message_severity Tag parameter is set to no and the W210 Tag reports all unconnected ports with message severity as

Warning.

Set the `set_message_severity` parameter to `yes` if you want to report unconnected input or inout ports with messages severity as **Error**.

Used by	W210
Options	yes, no
Default value	no

SETNET

Use this parameter to read the sets specified using the SGDC or SDC commands. Specify the value of the set SGDC command as an input to this parameter (use space-separated values in case of multiple commands).

For example, for the following SGDC command:

```
set -name top.set top.mid1.rst
```

Specify the SETNET parameter as shown below:

```
configure_lint_tag_parameter -tag SETPINCONNECTEDTOSETNET
-parameter SETNET -value { top.reset top.mid1.rst2 } -goal
<goal_name>
```

Used by	setPinConnectedToSetNet
Options	space-separated list of set pin names
Default value	" "

sign_extend_func_names

The `sign_extend_func_names` parameter lists the function names that are used for sign extension. This parameter enables the respective Tags to recognize VHDL sign extension functions.

NOTE: *The `sign_extend_func_names` parameter is valid for VHDL only.*

The respective Tags calculate the width of extend functions as per the `const` extension argument specified in the argument list.

Used By	W116 , W164a
Options	comma-separated list of function names
Default Value	EXTEND

simplesense

Enables the [W122](#) or NotInSens tag to report the signals that are read in an `always` construct and being set in blocking assignment statement but are not present in the sensitivity list.

By default, the `simplesense` Tag parameter is not set. Therefore, if a signal is assigned some value through a blocking assignment and is later read in same `always` construct, the `W122` or [NotInSens](#) tag ignores that signal provided its bit index is not a variable.

You can set the value of the `simplesense` tag parameter to `yes` or `1` (to enable the mode) or to `no` or `0` (to disable the mode).

Used by	W122 , NotInSens
Options	yes, no
Default value	no

traverse_function

Specifies whether the inputs should be considered as read when being read inside a function.

By default, the `traverse_function` parameter is set to `no` and the [W123](#) Tag considers the inputs as unread when being read inside the function block.

Set this parameter to `yes` to enable the `W123` Tag to consider such inputs as read. This parameter is applicable to **VHDL** only.

Used by	W123
Options	yes, no
Default value	no

treat_concat_assign_separately

Use this parameter to report violation for each bucket assignment in unpacked array separately. For a packed array, a violation is reported for the whole array.

Used by	W164a , W164b
Options	yes, no
Default value	no

treat_latch_as_combinational

Specifies to treat combinational block inferring latch as combinational block while Tag checking with the [W336](#) and [W414](#) Tags.

By default, the `treat_latch_as_combinational` Tag parameter is not set and the [W336](#) and [W414](#) Tags treat combinational block inferring latch as sequential block.

You can set the value of the `treat_latch_as_combinational` Tag parameter to `yes` (to enable the mode) or to `no` (to disable the mode).

Used by	W336 , W414
Options	yes, no, comma separated list of Tags
Default value	no

thresholdScanChainLength

(Optional) Specify the maximum number of elements to be reported in the violation message for a scan chain (checked by the [elementsAllowedPerScanChain](#) tag) to control the memory or time requirements for long scan chains during the SpyGlass run.

If the length of a scan chain is greater than the number specified by the `thresholdScanChainLength` parameter, then only the first `thresholdScanChainLength` elements of the scan chain are reported. Make sure that the value of the `thresholdScanChainLength` parameter is reasonably large, for

example, 10000. This ensures that the memory or time requirement is optimal and you can check most scan chains in the design.

In any case, the value of the `thresholdScanChainLength` parameter must be more than the value of the `allowedScanChainLength` parameter.

Used by	<i>elementsAllowedPerScanChain</i>
Options	Positive integer number
Default value	10000

use_carry_bit

Enables the [*W116*](#) and [*W164a*](#) (Verilog) Tags to get the width after considering the carry bit of addition.

By default, the `use_carry_bit` parameter is set to `no` and the width is taken as maximum of the two operands for a binary expression having plus and minus operators.

Set this parameter to `yes` or `<Tag-name>` to get width after considering the carry bit of addition. No violation is reported, even using this parameter, for sub-expressions of a binary expression if all terms have the same width and all operators are either plus or minus.

For W164a tag: By default, when the `nocheckoverflow` parameter is set `no`, the width of addition and subtraction is considered as the width of the maximum value that the expression can hold. Set the `use_carry_bit` parameter (with default value as `W164a`) to `no` to calculate the width as the LRM, that is the maximum width of the operand.

Example:

```
a[2:0] = b[2:0] + c[2:0];
```

By default, the `W164a` tag reports violation because RHS width is 4. However, when you set the value of the parameter to `off W164a`, width of RHS is 3 and the tag does not report any violation.

Used by	<i>W116, W164a</i>
Options	yes, no, Tag name
Default value	no

use_lrm_width

Specifies whether the related Tags should consider the LRM width, which is 32 bits, for unsized based numbers and integer constants.

By default, the *use_lrm_width* parameter is set to *no*, and the natural width of integer constants is considered, which is $\log_2(N) + 1$.

Set this parameter to *yes* to consider the LRM width, which is 32 bits, for unsized based numbers and integer constants.

Set this parameter to a Tag name so that the *use_lrm_width* parameter is applicable to the specified Tags only. For the other Tags, which are not specified with this parameter, VC SpyGlass Lint considers the natural width for integer value.

NOTE: For new width related changes, refer to the New Width Flow Application Note.

Used by	Lint: <i>W164a</i> , <i>W164b</i> , <i>W110</i> , <i>W116</i> , <i>W263</i> , <i>W362</i> , <i>W486</i> , <i>W416</i> STARC2005: <i>STARC05-2.1.3.1</i> , <i>STARC05-2.8.1.6</i> , <i>STARC05-2.10.3.1</i> , <i>STARC05-2.10.3.2a</i> , <i>STARC05-2.10.3.2b</i> , <i>STARC05-2.10.3.2c</i> , <i>STARC05-3.2.3.2</i> , <i>STARC05-2.10.6.5</i>
Options	yes, no, comma-separated Tag list
Default value	no

use_natural_width

Use this parameter to calculate the width using the natural width.

Used by	<i>W224</i>
Options	yes, no
Default value	no

verilint_compat

Specifies to run the Tags that are normally not required as they check for syntax errors (that are already reported by VC SpyGlass Lint) or are redundant.

VC SpyGlass Lint Parameters

By default, the `verilint_compat` Tag parameter is not set and the following Tags are not run:

W159	W163	W259	W313	W316	W326
W348	W464	W474	W475	W476	W477
W493	W546				

You can set the value of the `verilint_compat` Tag parameter to `yes` or `1` (to enable the mode) or to `no` or `0` (to disable the mode).

Used by	W159 , W163 , W259 , W313 , W316 , W326 , W348 , W464 , W474 , W475 , W476 , W477 , W493 , W546
Options	yes, no
Default value	no

W416_vhdl_only

Specifies whether the [W416](#) tag should use Verilog or VHDL version.

By default, the value of the parameter is set to `no`.

Set the value of the parameter to `yes` to enable the [W416](#) tag to use only the VHDL version and check VHDL parts of the design.

Used by	W416
Options	yes, no
Default value	no

allow_combo_cloud

Enables the [RptNegEdgeFF-ML](#) Tag to traverse through combinational logic to check whether the clock signal is inverted.

By default, the value of the parameter is `no`. In this case, the [RptNegEdgeFF-ML](#) Tag traverses through buffers and inverters in the clock path to check whether the clock signal is inverted.

Set the value of the parameter to `yes` to enable the [RptNegEdgeFF-ML](#) Tag to

traverse through combinational logic to check whether the clock signal is inverted.

Used by	RptNegEdgeFF-ML
Options	yes, no
Default value	no

allow_combo_logic_base

Specifies whether the affected tags should skip all VC SpyGlass Lint-generated combinational elements in the path.

By default, the `allow_combo_logic_base` parameter is set to `no`. This indicates that the Tag skips only VC SpyGlass Lint-generated buffers and inverters in the path.

Set this parameter to `yes` to skip all VC SpyGlass Lint-generated combinational elements in the path, starting from port of an instance, to check whether the port is connected to a constant or not.

Used by	ClockPhase , ConstantInput-ML , NoFeedThrus-ML , STARC05-1.3.1.3 , STARC05-1.3.2.2 , STARC05-1.4.3.4 , STARC05-1.2.1.1a , W391 , W392 , STARC05-3.3.6.2
Options	yes, no, comma/space-separated list of tags
Default value	yes for STARC05-1.3.2.2 tag, no for remaining tags

avoid_generated_supply

Ignores supply nets that are internally generated and does not report violation for them.

By default, the value of the parameter is `no`. In this case, `ForbiddenPadConn-ML` Tag reports violations if any power supply net is connected to pad pin of cells.

Set the value of the parameter to `yes` to ignore supply nets that are internally generated.

Used by	ConstantInput-ML
Options	yes, no
Default value	no

buf_count

(Optional) Specifies the maximum number of buffers allowed between the input/output port and register input/output for the specified tags.

By default, the `buf_count` Tag parameter is set to -1 and these Tags allow any number of buffers.

You can set the `buf_count` Tag parameter to any positive integer number.

Used by	ScopedVarUsedBeforeDefine-ML , RegInput-ML , RegInputOutput-ML , STARC05-1.6.2.1
Options	Positive integer value
Default value	-1

capa_checkwithinblock

(Optional) Specifies whether the [CAPA-ML](#) Tag should check within a block for duplicate code sections.

By default, the value of the `capa_checkwithinblock` Tag parameter is set to `no` and no Tag checking is done within a block.

Set the value of the `capa_checkwithinblock` Tag parameter to `yes` to enable Tag checking within a block.

Used by	CAPA-ML
Options	yes, no
Default value	no

capa_max_line_count

Enables the [CAPA-ML](#) tag to specify the maximum line number count of the modules/architectures that should be considered while rule checking.

By default, the value of the `capa_max_line_count` tag parameter is set to 0 and the rule checks all the available modules/architecture. And if this parameter is enabled, the tag will count the actual line count of modules/architectures by skipping empty lines and the lines that have only comments.

Set the value of the `capa_max_line_count` tag parameter to a positive integer to specify the maximum line number count of the modules/architectures that should be considered while tag checking. If the line count of a module/ architecture is greater than the given value, then the tag will not check that module/ architecture.

Used by	CAPA-ML
Options	positive integer
Default value	0

capa_scope

(Optional) Specifies the scope at which the [CAPA-ML](#) Tag checking is done to detect duplicate sections of code.

By default, the `capa_scope` Tag parameter is set to `module` and the CAPA-ML Tag checks in modules (Verilog) and architectures (VHDL).

Set the `capa_scope` Tag parameter to `file`, and the CAPA-ML Tag checks in files.

Used by	CAPA-ML
Options	module, file
Default value	module

capa_similarity

(Optional) Specifies the minimum percentAge match between two sections of code to detect a copy-paste operation by the [CAPA-ML](#) Tag.

By default, the `capa_similarity` Tag parameter is set to 1.0 and it checks for 100% match between two sections of code.

You can set the `capa_similarity` Tag parameter to a different floating point value to specify a different match number.

Used by	CAPA-ML
Options	Floating point number
Default value	1.0

capa_size

(Optional) Specifies the minimum number of lines to be matched by the [CAPA-ML](#) Tag parameter to detect duplicate sections of code.

By default, the `capa_size` Tag parameter is set to 5 and the CAPA-ML Tag checks 5 lines of code.

You can specify a different number with the `capa_size` Tag parameter to have the CAPA-ML Tag check for a different number of lines of code.

Used by	CAPA-ML
Options	Positive integer value
Default value	5

casedepth

(Optional) Specifies the maximum allowed case construct nesting depth for the [NestedCaseStmt-ML](#) Tag.

By default, the `casedepth` Tag parameter is set to 1 and the NestedCaseStmt-ML Tag flags nested case constructs with more than one level of nesting.

You can specify a different number with the `casedepth` Tag parameter to have the NestedCaseStmt-ML Tag flag for a different nesting level.

categorize_violations

Enables the [NoFeedThrus-ML](#) Tag to categorize violations in Error and Warning severity labels.

By default, the `categorize_violations` Tag parameter is set to `no`.

When you set this parameter to `yes`, the violations are reported with Error and Warning severity labels.

- The violations for the feedthrough paths that are passing through buffers and inverters are labeled as Error.
- The violations for the feedthrough paths passing through other combination logics are labeled as Warning.

Used by	NoFeedThrus-ML
Options	yes, no
Default value	no

checkallblocks

(Optional) Specifies whether the [SelfAssignment-ML](#) Tag checks assignment statements in both combinational and sequential always constructs.

By default, the `checkallblocks` Tag parameter is set to `no` and the `SelfAssignment-ML` Tag checks assignment statements in combinational always constructs only.

Set the `checkallblocks` Tag parameter to `yes`, and the `SelfAssignment-ML` Tag checks assignment statements in both combinational and sequential always constructs.

Used by	SelfAssignment-ML
Options	yes, no
Default value	no

check_all

Specifies whether the *SignedUnsignedExpr-ML* Tag should check for all data types.

By default, the `check_all` parameter is set to `no` and the *SignedUnsignedExpr-ML* Tag does not check for all of the data types.

Set the `check_all` parameter to `yes` to enable the *SignedUnsignedExpr-ML* Tag to check for all of the data types.

Used by	<i>SignedUnsignedExpr-ML</i>
Options	yes, no
Default value	no

check_all_pragmas

Reports violations for unrecognized pragmas, which are semicolon-separated or space-separated in a single line comment.

The Default value for the parameter is `no`. In this case, the *UnrecSynthDir-ML* Tag does not report violations for unrecognized pragmas, which are semicolon-separated or space-separated in a single line comment. Set the value of the parameter to `yes` to report violations for such pragmas.

Used by	<i>UnrecSynthDir-ML</i>
Options	yes, no
Default value	no

check_blackboxes

By default, the *ConstantInput-ML* Tag does not report violations for black box instances, which are instantiated at the top-level.

Set the value of the `check_blackboxes` parameter to `yes` to report violations for black boxed instances.

Used by	<i>ConstantInput-ML</i>
Options	yes, no
Default value	no

checkcontassign

Specifies whether the [SelfAssignment-ML](#) Tag should ignore procedural blocks.

By default, the `checkcontassign` parameter is set to `no` and the [SelfAssignment-ML](#) Tag checks for procedural blocks.

If you set the `checkcontassign` parameter to `yes`, the [SelfAssignment-ML](#) Tag ignores all types of procedural blocks (`always` and `initial`). Also, the Tag reports violations for continuous assignment statements if the LHS of expression is of `net` type.

NOTE: *If the 'checkallblocks' and 'checkcontassign' parameters are set at the same time, the 'checkcontassign' parameter will be ignored.*

Used by	SelfAssignment-ML
Options	yes, no
Default value	no

check_dup_label

(Optional) Specifies whether the [NoGenLabel-ML](#) Tag reports any duplicate generate labels

By default, the Tag does not report any duplicate generate labels. Set the `check_dup_label` parameter to `yes` to report duplicate generate labels.

The `check_dup_label` parameter is affected by the values of the [genlabeltype](#) and [strict](#) parameters. For instance:

- If the [genlabeltype](#) parameter is set to `generate-if` and `generate`, the Tag reports duplicate labels of the `generate-if` and `generate` blocks, which match the labels of either the `generate-if` or `generate` block. In this case, the Tag does not report a label of the `generate-if` block that matches to a label of the `generate-for` block.
- By default, the Tag reports a violation only for duplicate labels of those `generate` blocks that contain instantiation or declaration. When the

strict parameter is set to *yes*, the Tag reports violations for duplicate labels of all generate blocks.

Used by	<i>NoGenLabel-ML</i>
Options	yes, no
Default value	no

check_enum_decl

Specifies whether the *NoAssignX-ML* tag reports violations for enum declarations.

By default, the parameter is set to *no* and the tag does not report violations for enum declarations where x values are assigned to enum elements.

Set the *check_enum_decl* parameter to *yes* to report violations for enum declarations where x values are assigned to enum elements.

Used by	<i>NoAssignX-ML</i>
Options	yes, no
Default value	no

check_gate_primitives

Specifies if the *ConstantInput-ML* Tag reports gate instances.

By default, the *check_gate_primitives* parameter is set to *no*.

Set this parameter to *yes* to report violations for gate instances.

Used by	<i>ConstantInput-ML</i>
Options	yes, no
Default value	no

checkInHierarchy

Specifies whether the Tags using this parameter should traverse down the hierarchy to search for valid driver/load.

By default, the `checkInHierarchy` Tag parameter is set to `yes` and the specified Tags traverse hierarchically inside to search for valid driver/load.

Set the `checkInHierarchy` Tag parameter to `no`, and the specified Tags do not check inside the hierarchy.

Used by	HangingInstInput-ML , UndrivenOutPort-ML , UndrivenInTerm-ML , UndrivenNet-ML , UndrivenNUnloaded-ML , UnloadedInPort-ML , UnloadedNet-ML , UndrivenOutTermNLoaded-ML , UnloadedOutTerm-ML , Rule_34 , Rule_35 , UndrivenOutTermNLoaded-ML
Options	- no for Rule_34, Rule_35 - yes for UndrivenInTerm-ML, UnloadedNet-ML, UndrivenNUnloaded-ML, UndrivenOutPort-ML, UndrivenOutTermNLoaded-ML, UnloadedInPort-ML, UnloadedOutTerm-ML, HangingInstInput-ML, UndrivenOutTermNLoaded-ML
Default value	yes

check_conversion_function

Enables the [ValueSizeOverflow-ML](#) Tag to check for TO_SIGNED and TO_UNSIGNED conversion functions.

By default, the [ValueSizeOverflow-ML](#) Tag checks for a predefined set of IEEE conversion functions as described in the [ValueSizeOverflow-ML](#) Description.

To enable the Tag to also check for TO_SIGNED and TO_UNSIGNED conversion functions, set the value of the parameter to comma-separated list of these IEEE conversion functions.

Used by	ValueSizeOverflow-ML
Options	<comma-separated list of IEEE conversion functions: TO_SIGNED and TO_UNSIGNED>
Default value	" "

check_default_value

Specifies whether the [UndrivenInTerm-ML](#) Tag checks for the input ports of the VHDL entity or components, which have a default value and have the port unconnected in the port-map.

By default, the Tag does not reports violation for input port of VHDL entity or components, which have a default value and the port is left unconnected.

Set the value of the `check_default_value` parameter to `no` to enable the [UndrivenInTerm-ML](#) Tag to report violations for such input ports.

Used by	UndrivenInTerm-ML
Options	yes, no
Default value	yes

check_complex_operators

Specifies whether the [MultiOpInModule-ML](#) Tag checks for complex operators in the design.

By default, the `check_complex_operators` Tag parameter is set to `yes` and the [MultiOpInModule-ML](#) Tag checks for complex operators.

Set the `check_complex_operators` Tag parameter to `no`, and the [MultiOpInModule-ML](#) Tag checks for base operators.

Used by	MultiOpInModule-ML
Options	yes, no
Default value	yes

check_conditional_assignment

Specifies whether the [STARC05-2.7.1.3a](#) and [IfWithoutElse-ML](#) tags reports a violation for the signals that are assigned inside both `case` block and `if` condition. By default, the value of the parameter is set to `no` and the *specified* tag does not report violation for the signals that are assigned inside both `case` block and `if` condition.

You can set the value of the parameter to `yes` to report such signals.

Used by	STARC05-2.7.1.3a , IfWithoutElse-ML
Options	yes, no
Default value	no

check_default_state

Specifies whether the [LINT_FSM_UNREACHABLE_STATES](#) tag should check the default state.

By default, the `states_to_check` parameter is set to `IGNORE_DEFAULT`, the `LINT_FSM_UNREACHABLE_STATES` tag ignores the default state.

If you set to this parameter to `CHECK_DEFAULT`, the tag checks and reports the default state, which cannot be reached from the initial state.

Used by	LINT_FSM_UNREACHABLE_STATES
Options	<code>IGNORE_DEFAULT</code> , <code>CHECK_DEFAULT</code>
Default	<code>IGNORE_DEFAULT</code>

check_names_space

Enables the [AutomaticFuncTask-ML](#) tag to specify the names space to be checked. You can provide one or more of the following name space tags to the `check_names_space` parameter:

- **module** - To report functions in module scope.
- **package** - To report functions in package scope.
- **interface** - To report functions in interface scope.
- **global** - To report functions in global scope.

By default, the `check_names_space` parameter is set to `module`, `interface`, `package` and the *AutomaticFuncTask-ML* tag reports violations only for functions in module scope, package scope and interface scope.

Used by	<i>AutomaticFuncTask-ML</i>
Options	Comma-separated list of name space tags
Default	module, interface, package

check_nested_decl

Specifies whether the *NoGenLabel-ML* tag should report violations if the parent `generate` blocks do not have a name.

By default, the `check_nested_decl` parameter is set to `no`.

Set the value of this parameter to `yes` to consider instances and declarations in nested `generate` blocks as part of parent `generate` blocks and report violations if the parent `generate` blocks do not have a name.

Used by	<i>NoGenLabel-ML</i>
Options	yes, no
Default	no

check_op_in_flop

Specifies whether the *MultiOpInModule-ML* Tag checks in sequential blocks inferring flip-flops.

By default, the `check_op_in_flop` Tag parameter is set to `no` and the *MultiOpInModule-ML* Tag does not check in sequential blocks inferring flip-flops.

Set the `check_op_in_flop` Tag parameter to `yes`, and the `MultiOpInModule-ML` Tag checks in sequential blocks inferring flip-flops.

Used by	MultiOpInModule-ML
Options	yes, no
Default value	no

check_output_ports

Specifies whether the [SetBeforeRead-ML](#) Tag checks for output ports.

By default, the `check_output_ports` Tag parameter is set to `no` and the [SetBeforeRead-ML](#) Tag does not check for output ports.

Set the value of the `check_output_ports` parameter to `yes` to enable the [SetBeforeRead-ML](#) Tag to check for output ports of modules and functions/tasks along with the internal signals.

Used by	SetBeforeRead-ML
Options	yes, no
Default value	no

checkTopDu

Specifies whether the Tag checking should be performed at the top modules only.

By default, this parameter is set to `no` and the `NoFeedThrus-ML` Tag reports violation for feedthrough connections present in every module and the `NoExprInPort-ML` reports violation for logic expressions which are present in port connections in the instances inside every module.

The following is the behavior of Tags when this parameter is set:

- [NoFeedThrus-ML](#): If you set the parameter value to `yes` or `NoFeedThrus-ML`, then the `NoFeedThrus-ML` Tag reports violation for feedthrough connection present in the top module only.
- [NoExprInPort-ML](#): If you set the parameter value to `yes` or `NoExprInPort-ML`, then `NoExprInPort-ML` Tag reports violation for

logic expressions, which are present in port connections of the instances inside the top module only.

- [Rule_34](#), [Rule_35](#): By default, the tags check for all DU's but if option `-checkTopDu` is set to tag name then it will check only top level nets and terminals and while detecting load, tag dives down the hierarchy till it detects the input of a combination/sequential/black-box. Also, in case `-checkTopDu` is set to tag name, then to detect a driver it traverses back to find a driver through hierarchy.

Used by	NoFeedThrus-ML , NoExprInPort-ML , Rule_34 , Rule_35
Options	yes, no, comma separated Tag names
Default value	no

checkGateInst

Specifies whether the `NoExprInPort-ML` Tag reports violations for Verilog primitives.

By default, this parameter is set to `no` and the `NoExprInPort-ML` Tag reports violations for logic expressions which are present as port connections of user module instances. However, when instance is a Verilog primitive, the `NoExprInPort-ML` Tag skips reporting violation for logic expressions that are present in port connections.

Set this parameter `yes` to include instances of Verilog primitives also for reporting violations by the `NoExprInPort-ML`.

Used by	NoExprInPort-ML
Options	yes, no
Default value	no

check_relational_operators

Specifies whether the [LINT_SV_STRING_USED_IN_DESIGN](#) tag should report violation only if string is used in a relational operation.

By default, the `check_relational_operators` parameter is set to `no`.

If you set to this parameter to *yes* , the *LINT_SV_STRING_USED_IN_DESIGN* tag reports violation only if a string is used in a relational operation.

Used by	LINT_SV_STRING_USED_IN_DESIGN
Options	yes, no
Default	no

check_ifdef_define_include

Enables the [DirectiveCheck-ML](#) Tag to report ``define`, ``ifdef` , and ``include` constructs.

By default, the `check_ifdef_define_include` Tag parameter is set to `no`.

Set this Tag parameter to *yes*, to enable the *DirectiveCheck-ML* Tag to report for ``define`, ``ifdef` , and ``include` constructs used in the design.

The messages are enhanced as follows:

Message 1:

``ifdef` construct used

Message 2:

``define` construct used to define macro 'define macro'

Message 3:

``include` construct used to include 'include file name'

Used by	DirectiveCheck-ML
Options	yes, no
Default value	no

check_in_top

Specifies whether or not the constructs declared in the top-level hierarchy should be reported by the [LongHierName-ML](#) tag.

By default, the value of the `check_in_top` tag parameter is set to `yes` and the [LongHierName-ML](#) tag reports constructs declared in the top-level and higher-level hierarchies.

Set the value of this parameter to `no` to stop the violations for constructs declared in the top-hierarchy. In this case, only the constructs declared at least two levels down the hierarchy are reported by the [LongHierName-ML](#) tag.

Used by	LongHierName-ML
Options	no, yes
Default value	yes

check_lhs_rhs_assign_width

Specifies whether the [SelfDeterminedExpr-ML](#) Tag should not report violation for assignments where the LHS width is greater than the RHS width.

By default, the value of the `check_lhs_rhs_assign_width` parameter is set to `no`.

Set the value of this parameter to `yes` to not report violation for assignments where the LHS width is greater than the RHS width.

Used by	SelfDeterminedExpr-ML
Options	yes, no
Default	no

checktaskarg

Specifies whether the [UnusedFunctionInput-ML](#) Tag checks the input ports of tasks.

By default, the `checktaskarg` parameter is set to `no`.

Set this parameter to `yes` to enable the `UnusedFunctionInput-ML` Tag to check the input ports of tasks.

Used by	<i>UnusedFunctionInput-ML</i>
Options	yes, no
Default value	no

check_type

Specifies whether all or specific types of constructs should be reported by the [*LongHierName-ML*](#) tag.

By default, the value of the `check_type` parameter is set to `all` and the `LongHierName-ML` tag reports violations for all types of following constructs:

`net, port, instance, parameter, variable, task, function, integer, real`

Set the value of the `check_type` parameter to a comma-separated or space-separated list of one or more of the above construct types to specify the construct types to be reported by the `LongHierName-ML` tag.

Used by	<i>LongHierName-ML</i>
Options	Comma-separated or space-separated list of the following construct types: <code>net, port, instance, parameter, variable, task, function, integer, real, or all</code>
Default value	all

check_unused_typedef

Specifies whether the [*InvalidTypedefName-ML*](#) tag checks the unused typedef names.

By default, the `check_unused_typedef` parameter is set to `no`.

Set this parameter to `yes` to enable the `InvalidTypedefName-ML` tag to

report a violation for unused typedef names.

Used by	InvalidTypedefName-ML
Options	yes, no
Default value	no

check_xassign_casedefault

Checks for default clause of case statement, which has static value containing X on the RHS of the assignment.

By default, the value of the `check_xassign_casedefault` parameter is set to `no` and the [NoAssignX-ML](#) Tag does not report any violation for the default clause of case statement.

Set the value of the parameter to `yes` to report violation for such cases.

Used by	NoAssignX-ML
Options	yes, no
Default value	no

checkregisterdu

Specifies a comma-separated list of module names so that the [RegInput-ML](#) tag checks ports of the specified modules only.

If you do not specify this parameter or you do not specify an empty value to this parameter, the RegInput-ML rule-checking is done for ports of each and every user-defined top modules in a design.

Used by	RegInput-ML
Options	A comma-separated list of module names
Default value	NULL

check_register_type

Specifies if a port connected to a flop, latch, or memory object types should be considered as registered by the [RegInput-ML](#) tag.

By default, the [RegInput-ML](#) tag considers a port as registered if the port is connected to a flop.

Used by	RegInput-ML
Options	A comma separated list of object types such as, flop, latch, and memory
Default value	flop

check_unloaded_ports

Specifies if the [RegInput-ML](#) tag reports violation for unloaded/unconnected input/inout ports. By default, the [RegInput-ML](#) tag does not report these violations.

Used by	RegInput-ML
Options	yes, no
Default value	no

chkTopModule

(Optional) Specifies whether the [RegInputOutput-ML](#) Tag checks for all levels of hierarchy or only top-level and block-level design units.

By default, the `chkTopModule` Tag parameter is set to `no` and the [RegInputOutput-ML](#) Tag checks for all levels of hierarchy.

Set the `chkTopModule` Tag parameter to `yes`, and the [RegInputOutput-ML](#) Tag checks for the top-level design units and those design units where VC SpyGlass Lint block pragmas have been specified as follows:

```
--VHDL:
-- VC SpyGlass Lint block
architecture ...
```

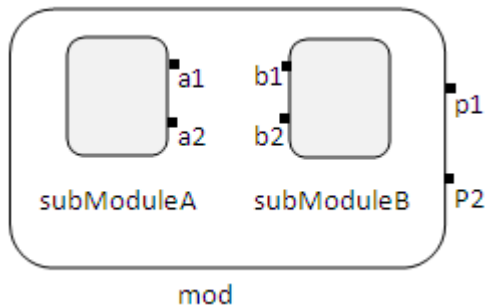
```
//Verilog:
```

VC SpyGlass Lint Parameters

```
// VC SpyGlass Lint block
module ...
```

If the `chkTopModule` Tag parameter is set to `yes` and no design units have `block` pragma specified, the `RegInputOutput-ML` Tag checks for top-level design units only.

For example, consider the following figure:



The module, `mod`, contains two sub-modules: `subModuleA` and `subModuleB`. The following table specifies the modules and the ports assigned to them:

Module Name	Module Type	Port Name
Mod	Top-level Module	p1 and p2
SubModuleA	Sub module	a1 and a2
SubModuleB	Sub module	b1 and b2

Now, if you set the `chkTopModule` Tag parameter to `no`, the `RegInputOutput-ML` Tag checks for ports of all levels of hierarchy. Hence, in this case, the Tag checks for all the ports, `p1`, `p2`, `a1`, `a2`, `b1`, and `b2`. However, if you set the value of the `chkTopModule` Tag parameter to `yes`, the Tag checks only for the top-level ports, `p1` and `p2`.

Used by	RegInputOutput-ML
Options	yes, no

Default value	no
Default Value in GuideWare2.0	yes

check_var_decl_init

Specifies whether the [SimSynthMismatchInVarInit-ML](#) tag reports a violation for simulation - synthesis mismatch.

By default, the value of the parameter is set to `no`.

Set the value of the parameter to `yes` to enable the tag to perform the following:

- Avoid checking for initialization inside initial block of any data type such as bit, byte, int, reg and logic data type.
- Report a violation for initialization of reg, logic, bit, byte, short real, real, int, and other data type except the wire data types declared in the module scope.
- Report a violation in any named block where a reg, logic, bit, byte, integer, and int signal is initialized except the wire data type.
- Report a violation if there is any initialization done on all kind of data type such as reg, logic, integer, int or real data type inside generate block. The tag checks all types of generate blocks as follows:
 - ☐ generate for
 - ☐ generate case
 - ☐ generate if

Used by	SimSynthMismatchInVarInit-ML
Options	yes, no
Default value	no

comment_string

Specifies comma-separated list of strings to be searched in comment lines of the design for the specified tag.

By default, the `comment_string` Tag parameter is not set.

You can set the value of the `comment_string` Tag parameter to any character string.

Used by	FindStringsInComment-ML , <code>LineLength</code>
Options	Comma-separated list of strings
Default value	" "

consider_inout

Specifies whether the specified tag should ignore modules with inout port. By default, the `consider_inout` parameter is set to no. If you set this parameter yes, the `FuncWithIntReturn-ML` tag considers inout port as output port and ignores violation for the modules that have inout ports.

Used by	FuncWithIntReturn-ML , NoOutputPort-ML
Options	Comma-separated list of strings
Default value	" "

depth_ml

Specifies the hierarchy level up for the [RegInputOutput-ML](#) and [ScopedVarUsedBeforeDefine-ML](#) Tag.

By default, the `depth_ml` Tag parameter is set to -1 and the `RegInputOutput-ML` and `RegInput-ML` Tag checks at all hierarchy levels.

You can specify a different number with the `depth_ml` Tag parameter to have the specified Tag check at the specified level of hierarchy. For example, in VC SpyGlass Lint, the hierarchy of top modules is considered as 1. So, if you specify the value of the `depth_ml` Tag parameter as 1, then the specified Tag checks only top modules.

Used by	RegInput-ML , RegInputOutput-ML , ScopedVarUsedBeforeDefine-ML
Options	Integer value
Default value	-1

dccompat

Performs Tag checking according to the Design Compiler conventions.

By default, the `dccompat` parameter is set to `no` and the Tag checking is not performed according to Design Compiler conventions.

Set the value of the parameter to `yes` to consider unsized integer constants as signed and perform Tag checking according to the Design Compiler conventions.

Used by	SignedUnsignedExpr-ML
Options	yes, no
Default value	no

define_file_name

Specifies whether the [LINT_UNDEFINED_MACRO_USED](#) tag should ignore modules with inout port.

By default, the `define_file_name` parameter is not set.

You can set this parameter to a PERL regular expression to specify the file names. The [LINT_UNDEFINED_MACRO_USED](#) tag reports violations for those macros which are used in `ifdef/ifndef` but defined in a file which does not match with the regular expression specified through this parameter.

Used by	LINT_UNDEFINED_MACRO_USED
Options	PERL regular expression pattern
Default	" "

DEFINE_SAME_NET_START_POINT

Use this parameter to specify the name connected to the input port or output pin name of a cell or cell name.

If this parameter is empty then the rule will check for all the nets of the design.

Use this parameter to specify net names as described below:

```
configure_lint_tag_parameter -tag "SameNetConnectionCheck" -
parameter DEFINE_SAME_NET_START_POINT -value { top.net1
top.net2 } -goal test_goal
```

Used by	SameNetConnectionCheck , GlitchGenCircuitCheck
Options	space-separate list of strings
Default	" "

delta_delay_mode

Enables the [MaxDelay-ML](#) tag to identify the type of delays for which the tag should report violations.

By default, the value of the parameter is set to `sequential` and the tag reports a violation for only sequential delays.

Set the value of the parameter as follows:

- `combinational` - To report a violation for combinational delays.
- `all` - To report a violation for all delays.

Used by	MaxDelay-ML
Options	combinational, all
Default value	sequential

disable_vhdl_keywords

Specifies whether the [STARCO5-1.1.1.3](#) tag should report a violation for VHDL keywords.

By default, this parameter is set to no.

Set this parameter to yes to ignore VHDL keywords.

Used by	STARCO5-1.1.1.3
Options	yes, no
Default value	no

dump_ifdef_data

Enables extraction of information regarding ifdef, undef, and include define directives in a design as used by [DirectiveCheck-ML](#) Tag.

By default, the dump_ifdef_data Tag parameter is set to no and the DirectiveCheck-ML Tag does not generate information for ifdef, undef, and include define directives.

Set the dump_ifdef_data Tag parameter to yes, and the DirectiveCheck-ML Tag will generate messages for ifdef, undef, and include define directives.

Used by	DirectiveCheck-ML
Options	yes, no
Default value	no

directivecheck

Enables the [DirectiveCheck-ML](#) tag to report violations for the following compiler directives:

- ``ifdef`
- ``undef`
- ``else`
- ``include`
- ``ifndef`
- ``elsif`
- ``define`

By default, the `directivecheck` parameter is set to "" and does not report violations for the above mentioned compiler directives.

Set the `directivecheck` tag parameter to a comma separated list of directives to report violations for the above mentioned compiler directives.

NOTE: *If the `directivecheck` parameter is enabled, then it is prioritized (over other parameters) and other parameters are neglected.*

Used by	DirectiveCheck-ML
Options	comma-separated list of compiler directives
Default value	""

enum_typedef_regex

Use this parameter to report violations for all the used enum typedefs whose names do not match with any of the Perl regular expression patterns specified through this parameter.

The default value of the parameter is `'.*_e'`.

Used by	InvalidVariedTypedefNameStyle-ML
Options	Comma-separated perl regular expression list of accepted typedef names
Default value	<code>'.*_e'</code>

excluded_cell_list

Specifies the list of excluded cell names.

By default, the `excluded_cell_list` parameter is not set.

Set this parameter to a comma separated list of excluded cell names. When a net is connected to these cells, it is not considered as valid.

Used by	ExternalPortBranchCheck
Options	Comma separated list of cells
Default value	""

other_typedef_regex

Use this parameter to report violations for all the used typedefs, other than enum, struct, and union, whose names do not match with any of the Perl regular expression patterns specified through this parameter.

The default value of the parameter is `'.*_t'`.

Used by	InvalidVariedTypedefNameStyle-ML
Options	Comma-separated perl regular expression list of accepted typedef names
Default value	<code>'.*_t'</code>

struct_typedef_regex

Use this parameter to report violations for all the used struct typedefs whose names do not match with any of the Perl regular expression patterns specified through this parameter.

The default value of the parameter is `'.*_s'`.

Used by	InvalidVariedTypedefNameStyle-ML
Options	Comma-separated perl regular expression list of accepted typedef names
Default value	<code>'.*_s'</code>

union_typedef_regex

Use this parameter to report violations for all the used union typedefs whose names do not match with any of the Perl regular expression patterns specified through this parameter.

The default value of the parameter is `'.*_u'`.

Used by	InvalidVariedTypedefNameStyle-ML
Options	Comma-separated perl regular expression list of accepted typedef names
Default value	'*,*_u'

effort_dissolve_brackets

Enables the [RedundantLogicalOp-ML](#) Tag to produce more accurate results in the expressions involving small brackets.

By default, the value of the effort_dissolve_brackets parameter is set to 500. Set the value of this parameter to any integer, as setting higher value for this parameter leads to more accurate results in the expressions involving small brackets and similar operators.

NOTE: High value of the parameter may also lead to high Tag runtime.

Used by	RedundantLogicalOp-ML
Options	integer
Default value	500

evaluate_for_loop_index

Use this parameter to consider loop index of Verilog for construct as a static value and report a violation if any of the loop iterations violates the Tag.

However, if the loop index is compared with a constant value, the Tag reports a violation only if the constant value is out of the range of for loop.

Used by	AlwaysFalseTrueCond-ML
Options	yes, no
Default value	no

FUSE_PIN

Use this parameter to specify names of EFUSE instances and pins.
By default, the value of the parameter is ' '.

Set the value of the parameter to a comma-separated list of EFUSE instances and pin names.

Used by	Rule_43
Options	<EFUSE_CELL_NAME,EFSUE_INPUT_PIN1, EFSUE_INPUT_PIN2; same for other cell>
Default value	' '

filter_named_resets

Specifies a list of strings such that VC SpyGlass Lint does not infer asynchronous resets whose names contain the specified strings.

By default, VC SpyGlass Lint infers resets if their names contain the string `rst`, `reset`, `set`, or `res`, even if you specify these strings to the `filter_named_resets` parameter. To disable the inferring of such resets, specify `no_default_list` to this parameter along with the other strings to be ignored.

Used by	Reset_check11
Options	Comma or space-separated list of strings, <code>no_default_list</code>
Default value	<code>clk</code> , <code>clock</code> , <code>scan</code>

reset_reduce_pessimism

Specifies the criteria to infer resets other than the resets inferred by setting the [use_inferred_resets](#) parameter to `yes`.

Default value is `filter_unused_synchronizer` and `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design.

Used by	Reset_check11
Options	filter_unused_synchronizer, same_data_reset_flop, all_potential_resets
Default value	filter_unused_synchronizer, same_data_reset_flop

genlabeltype

Specifies the types of generate blocks to be checked by the [NoGenLabel-ML](#) Tag.

By default, the *genlabeltype* parameter is set to `generate-if`, `generate-for`, and `generate-case`. Therefore, the Tag checks `generate-if`, `generate-for`, and `generate-case` types of generate blocks.

Set this parameter to `all` to check for all types of generate blocks.

However, if you set this parameter to a comma-separated list of one or more of the following values, the Tag checks only the specified types of generate blocks:

- `generate`
- `generate-if`
- `generate-for`
- `generate-case`

Used by	NoGenLabel-ML
Options	<code>all</code> , <code>generate-if</code> , <code>generate-for</code> , <code>generate-case</code> , <code>generate</code>
Default value	<code>generate-if</code> , <code>generate-for</code> , <code>generate-case</code>

global_timescale

Enables the [MaxDelay-ML](#) tag to set timescale for undefined modules.

By default, the value of the parameter is set to 0.

Set the value of the parameter to any positive integer (in picoseconds) to set timescale for undefined modules.

Used by	MaxDelay-ML
Options	any positive integer
Default value	0

handle_const_expressions

Enables the [SelfDeterminedExpr-ML](#) Tag to process expressions involving constants to check for violations.

Used by	SelfDeterminedExpr-ML
Options	yes, no
Default value	no

handle_greybox

Specifies whether the net connected to the greybox terminal should be considered as loaded/driven, depending on the terminal direction.

By default, the value of this Tag parameter is set to `yes`, and the net connected to the grey box terminal are considered as loaded or driven.

Used by	UndrivenOutPort-ML , UndrivenInTerm-ML , UndrivenNet-ML , UnloadedInPort-ML , UnloadedOutTerm-ML , UnloadedNet-ML , UndrivenNUnloaded-ML , UndrivenOutTermNLoaded-ML , HangingInstInput-ML , Rule_34 , Rule_35
Options	yes, no
Default value	yes

handle_hier_clock_reset

Specifies whether the [RegInputOutput-ML](#) Tag should go beyond the module boundary to detect clock and reset ports.

By default, the value of this Tag parameter is set to `no`.

Set the value of this parameter to `yes` to enable the `RegInputOutput-ML` Tag to go beyond the module boundary to detect clock and reset ports.

Used by	<i>RegInputOutput-ML</i>
Options	yes, no
Default value	no

handle_shift_in_concat

Specifies whether the [*SelfDeterminedExpr-ML*](#) tag reports a violation for the shift operations in a concatenation.

By default, this parameter is set to `yes` and the *SelfDeterminedExpr-ML* tag reports a violation for the shift operations in a concatenation.

Set this parameter to `no` to enable the *SelfDeterminedExpr-ML* tag to not report a violation for the shift operations in a concatenation.

Used by	<i>SelfDeterminedExpr-ML</i>
Options	yes, no
Default value	no

handle_signed_truncation

The `handle_signed_truncation` parameter specifies whether the [*SelfDeterminedExpr-ML*](#) Tag reports a violation for truncation due to the left shift operator inside a `$signed/$unsigned` system call, when value becomes `zero`.

By default, this parameter is set to `no` and the *SelfDeterminedExpr-ML* Tag does not report violation in such a case.

Set this parameter to `yes` to allow the *SelfDeterminedExpr-ML* Tag to report violation for truncation due to the left shift operator inside a `$signed/$unsigned` system call, when the value becomes `zero`.

Used By	SelfDeterminedExpr-ML
Options	yes, no
Default Value	no

hierarchical_name_length

Specifies the maximum permissible length of hierarchical names inferred in the design, as checked by the [LongHierName-ML](#) tag.

By default, the value of the `hierarchical_name_length` tag parameter is set to 1024. You can specify a different positive integer value to specify a different permissible length.

Used by	LongHierName-ML
Options	A positive integer number
Default value	1024

ignore_deliberately_unconnected

Enables the [UndrivenInTerm-ML](#) and [UnloadedOutTerm-ML](#) Tags to ignore violations for terminals, which are deliberately unconnected in Verilog.

Consider the following example:

```
sub U0 ( .in1 ( in ),
        .in2 ( ), // No violation for UndrivenInTerm-ML
        .out1(out1),
        .out2() // No violation for UnloadedOutTerm-ML
    ) ;
```

When the value of the `ignore_deliberately_unconnected` parameter is set to yes, the [UndrivenInTerm-ML](#) and [UnloadedOutTerm-ML](#) Tags do not report violations for terminals, which are deliberately unconnected in Verilog

Used by	UndrivenInTerm-ML , UnloadedOutTerm-ML

Options	yes, no
Default value	no

ignore_directive_names

Enables the [DirectiveCheck-ML](#) tag to provide the names of the compiler directives that should be ignored while checking the tag.

By default, the parameter is set to "".

Set the parameter to a comma or space-separated list of PERL regular expressions to provide the names of the compiler directives that should be ignored while checking the tag.

NOTE: This parameter works only if the [directivecheck](#) parameter is enabled.

Used by	DirectiveCheck-ML
Options	comma or space-separated list of PERL regular expressions
Default value	no

ignore_dup_label_stmt

Specifies whether the [DuplicateCaseLabel-ML](#) and [UnusedFlopOutput-ML](#) tags reports all duplicate case labels or ignores the duplicate case labels that have the same statement block.

By default, the value of this parameter is set to no and the [DuplicateCaseLabel-ML](#) Tag reports all duplicate case labels.

Set the value of this parameter to yes to ignore the duplicate case labels having the same statement block.

Used by	DuplicateCaseLabel-ML , UnusedFlopOutput-ML
Options	yes, no
Default value	no

ignore_elements

Specifies the sequential element to be considered as "NOT unreachable" by the *UnsuppCompDir-ML* Tag.

You need to specify comma separated list of hierarchical net-names for the `ignore_elements` Tag parameter. For example,

```
configure_lint_tag_parameter -tag UnsuppCompDir-ML -
parameter ignore_elements -value "<element1-name>,<element2-
name>,..."
```

By default, the *UnsuppCompDir-ML* Tag checks all the sequential elements in the design.

Used by	<i>UnsuppCompDir-ML</i> ,
Options	Comma-separated list of hierarchical net names
Default value	" "

ignore_enum_index

Disables the violations for enum indexes for the *NegativeIndex-ML* tag.

The default value is no. In this case, the tag reports violations for negative or enum array index.

Set the value of the parameter to yes to ignore violations for enum indexes.

Used by	<i>NegativeIndex-ML</i>
Options	Comma-separated list of hierarchical net names
Default value	" "

ignore_explicit_if_blocks

Specifies if the *IfWithoutElse-ML* Tag should not report the `if` blocks that contain explicit begin and end statements.

By default, the `ignore_explicit_if_blocks` parameter is set to no and the *IfWithoutElse-ML* Tag reports `if` blocks that contain explicit begin and end statements.

Set this parameter to `yes` to not report violation for the `if` blocks that contain explicit `begin` and `end` statements.

Used by	<i>IfWithoutElse-ML</i>
Options	yes, no
Default value	no

ignoreforloop

Specifies if the [*SelfAssignment-ML*](#) Tag should report violations for self assignments inside the for loop.

By default, the `ignoreforloop` parameter is set to `no` and the [*SelfAssignment-ML*](#) Tag reports violations for the self assignments inside the for loop.

Set this parameter to `yes` to not report violations for such cases.

Used by	<i>SelfAssignment-ML</i>
Options	yes, no
Default value	no

ignore_generate_blocks

Specifies whether the [*IfWithoutElse-ML*](#) Tag should process inside-generate blocks.

By default, the `ignore_generate_blocks` Tag parameter is set to `no` and the Tag processes inside-generate blocks.

If you set this parameter to `yes`, the Tag does not process inside-generate blocks.

Used by	<i>IfWithoutElse-ML</i>
Options	yes, no
Default value	no

ignore_genvar_condition

Enables the [AlwaysFalseTrueCond-ML](#) Tag to ignore conditions associated with genvars.

By default, the value of the parameter is no and the [AlwaysFalseTrueCond-ML](#) Tag considers the conditions associated with genvars.

Set the value of the parameter to yes to enable the [AlwaysFalseTrueCond-ML](#) Tag to ignore the conditions associated with genvars.

Used by	AlwaysFalseTrueCond-ML
Options	yes, no
Default value	no

ignore_inverters

Specifies whether all the inverters should be ignored while traversing from control pin or data pin as used by [SameControlINDataNet-ML](#) Tag.

By default, the `ignore_inverters` Tag parameter is set to no and internally generated buffers and even number of inverters between the path of control pin and data pin are ignored.

Set the `ignore_inverters` Tag parameter to yes, and the [SameControlINDataNet-ML](#) Tag ignores all the internally generated inverters along with buffers.

Used by	SameControlINDataNet-ML
Options	yes, no
Default value	no

ignore_interface_locals

Enables the [W424](#) tag to not report violations for functions inside an interface that are written to variables/signals in the interface scope.

Enables the [W425](#) tag to not report violations for functions inside an interface that are read to variables/signals in the interface scope.

By default, the value of the parameter is no and the W424 and the W425 tags report violations for functions inside an interface that are written/read to variables/signals in the interface scope respectively.

Set the value of the parameter to yes to ignore such violations.

Used by	W424 , W425
Options	yes, no
Default value	no

ignore_inoutport_expr

Specifies the Perl regular expressions to ignore chip-level IO, PAD IO, or Analog ports by the [NoInoutPort-ML](#) Tag.

When the `ignore_inoutport_expr` Tag parameter is set, it checks for ports at all levels of hierarchy and the values specified using the [inoutlevel](#) and [checkInHierarchy](#) Tag parameters are ignored.

Used by	NoInoutPort-ML
Options	Regular expression
Default value	" "

ignore_latches

Specifies whether the [IfWithoutElse-ML](#) Tag should ignore latch-inferred `if` blocks.

By default, the `ignore_latches` Tag parameter is set to no and the Tag reports latch-inferred `if` blocks.

If you set this parameter to yes, the Tag ignores latch-inferred `if` blocks.

Used by	IfWithoutElse-ML
Options	yes, no
Default value	no

ignore_loop_index

Specifies whether the [IfWithoutElse-ML](#) Tag should ignore the `if` blocks where the `if` condition is only dependent on loop variable.

By default, the `ignore_loop_index` parameter is set to `no`.

Set this parameter to `yes` to ignore violations for the `if` blocks where the `if` condition is only dependent on loop variable.

Used by	IfWithoutElse-ML
Options	yes, no
Default value	no

ignore_module

Specifies whether the specified tags should ignore specified modules.

By default, the `ignore_module` parameter is not set.

If you set this parameter to a comma-separated list of PERL regular expressions, the `FuncWithIntReturn-ML` tag ignores violations for the modules that match any of these specified regular expressions.

Used by	FuncWithIntReturn-ML , NoOutputPort-ML
Options	Comma separated list of PERL regular expressions
Default value	" "

ignore_mult_and_div

Enables the [W116](#) tag to ignore violations for multiplication (*) and division (/) operations.

By default, the value of the parameter is set to `no`. In this case, the `W116` rule reports violations for multiplication (*) and division (/) operations.

Set the value of the parameter to `yes` to ignore violations for such cases.

Used by	W116
Options	yes, no
Default value	no

ignore_multidim

Enables the [ResetFlop-ML](#) Tag to suppress violations from flip-flops, which are formed from multidimensional signals.

By default, the value of `ignore_multidim` parameter is set to `no` and the *ResetFlop-ML* Tag reports violation for flip-flops, which are formed from multidimensional signals.

Set the value of the parameter to `yes` to suppress such violations.

Used by	ResetFlop-ML
Options	yes, no
Default value	no

ignore_non_static_cond

Specifies whether the [NoConstSourceInAlways-ML](#) tag report a violation for non-static `case/if` conditions.

By default, the value of `ignore_non_static_cond` parameter is set to `no`.

Set the value of the parameter to `yes` to not report a violation for non-static `case/if` conditions. When a condition is a combination of static and non-static values, violations are still not reported.

Used by	NoConstSourceInAlways-ML
Options	yes, no
Default value	no

ignore_non_vector_signals

Enables the [W576](#) tag to not report violation for signals where it's not a reg/wire/logic/bit/byte or compound elements (one element is one of the aforementioned types).

By default, the value of the parameter is set to no. In this case, the tag reports violation for signals where it's not a reg/wire/logic/bit/byte or compound elements (one element is one of the aforementioned types).

Used by	W576
Options	yes, no
Default value	no

ignore_single_zero

Specifies whether the [NoWidthInBasedNum-ML](#) Tag should ignore violations for width unspecified based number with single zero.

By default, the `ignore_single_zero` Tag parameter is set to no.

Set this parameter to `yes` to enable the *NoWidthInBasedNum-ML* Tag to ignore violations for width unspecified based numbers with single zero.

Used by	NoWidthInBasedNum-ML
Options	yes, no
Default value	no

ignore_static_condition

Specifies whether the [IfWithoutElse-ML](#) tag should ignore violations for `if` blocks with static condition.

By default, the `ignore_static_condition` tag parameter is set to no.

Set this parameter to `yes` to enable the *IfWithoutElse-ML* tag to ignore violations for `if` blocks with static condition.

Used by	IfWithoutElse-ML
Options	yes, no
Default value	no

ignore_sibling_viol

Use this parameter to ignore sibling violations for the [UndrivenNet-ML](#) tag. By default, the value of the parameter is no. In this case, the tag reports violation for multiple instances of the same module, that is, sibling modules.

Used by	UndrivenNet-ML
Options	yes, no
Default value	no

ignore_sync_reset

Specifies whether the gates generated due to synchronous reset of flip-flops should be ignored by the [RegInputOutput-ML](#) Tag. Also, specifies whether the [NonConstReset-ML](#) Tag reports violation for signals under the asynchronous set/resets condition.

By default, the `ignore_sync_reset` Tag parameter is set to no. In this case, following is the Tag behavior of the [RegInputOutput-ML](#) and [NonConstReset-ML](#) Tags:

- The [RegInputOutput-ML](#) Tag reports violation for gates generated due to synchronous reset of flip-flops.
- The [NonConstReset-ML](#) Tag reports violations for signals that are not assigned to constant values under synchronous and asynchronous set/reset condition.

When you set the value of the `ignore_sync_reset` Tag parameter to yes, following is the Tag behavior of the [RegInputOutput-ML](#) and [NonConstReset-ML](#) Tags:

- The *RegInputOutput-ML* Tag ignore the gates generated due to synchronous reset of flip-flops. For example, consider the following figure:

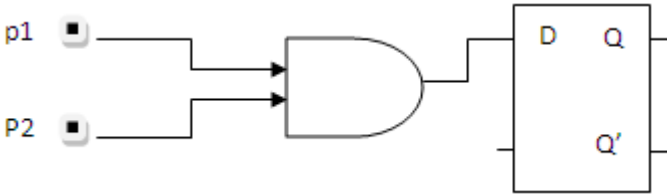


FIGURE 1. Gates generated due to synchronous reset of flip-flops

Here, ports, p1 and p2, are connected with the flip-flop through the AND gate. In this case, if the *ignore_sync_reset* Tag parameter is set to *yes*, the *RegInputOutput-ML* Tag will not report any violation.

- The *NonConstReset-ML* Tag reports violation for signals under asynchronous set/resets condition only.

Used by	<i>RegInputOutput-ML</i> , <i>NonConstReset-ML</i> , <i>UninitializedReset-ML</i>
Options	yes, no
Default value	no

ignore_top_level_pre_assignment

Specifies whether the *IfWithoutElse-ML* Tag should ignore top level pre-assignments.

By default, the *ignore_top_level_pre_assignment* Tag parameter is set to *no* and the Tag report violations for "if" blocks without "else" block if there are pre-assignments for all the variables modified inside "if" block.

If you set this parameter to *yes*, the Tag reports violations for "if" blocks without "else" blocks that are present inside both "case" block and "if" block and the pre-assignment done outside the parent if block.

Used by	IfWithoutElse-ML
Options	yes, no
Default value	no

ignore_unloaded_inst

Ignores combinational blocks from Tag checking.

By default, the value of this parameter is set to `no` and the Tags using this parameter consider combinational blocks to search for a valid load.

Set the value of the parameter to `yes` to ignore combinational blocks from Tag checking so that the Tags using this parameter traverse through the hierarchy to search for a valid load.

NOTE: Set the value of the [checkInHierarchy](#) parameter to `yes` to enable the functionality of this parameter.

Used by	UndrivenOutTermNLoaded-ML , UndrivenNUnloaded-ML , UnloadedInPort-ML , Rule_34 , Rule_35
Options	yes, no
Default value	no

ignore_unloaded_port

Specifies whether the [RegInputOutput-ML](#) Tag should ignore the violations related to unloaded input ports.

By default, the value of the `ignore_unloaded_port` parameter is set to `no` and the [RegInputOutput-ML](#) Tag reports violations for the input ports that are unloaded.

Set the value of this parameter to `yes` to not report violation for such cases.

Used by	RegInputOutput-ML
Options	yes, no
Default value	no

ignore_unused_flop

When the `-ignore_unused_flop` parameter is set for a specified tag, the tag ignores the basic combinational cells (buffer, inverter, OR, AND, NOR, NAND, XOR, XNOR, MUX) and checks across the hierarchy to search for a valid load. When it reaches a flop or latch, it further checks down the hierarchy until a module boundary is reached to see if the flop/latch output is used. If not used, the flop/latch is not considered as a valid load.

The effect of this parameter is only seen when the `checkInHierarchy` is set.

Used by	UndrivenInTerm-ML , UndrivenNet-ML
Options	yes, no
Default value	no

ignoreforindex

Specifies whether the [SignedUnsignedExpr-ML](#) Tag should ignore the violations related to `for` loop and `while` loop indexes.

By default, the value of the `ignoreforindex` parameter is set to `no` and the [SignedUnsignedExpr-ML](#) Tag reports violations related to `for` loop and `while` loop indexes.

Set the value of the parameter to `yes` to ignore such violations.

Used by	SignedUnsignedExpr-ML
Options	yes, no
Default value	no

ignoreInteger

Specifies whether the specified tag ignores a violation for integer index.

By default, the `ignoreInteger` parameter is set to `no`.

Set this parameter to `yes` to ignore a violation for integer index.

Used by	ImproperRangeIndex-ML, W164a
Options	yes, no
Default value	no

inoutlevel

Specifies the hierarchy level at which [NoOutputPort-ML](#) Tag should check for inout ports.

By default, the `inoutlevel` Tag parameter is set to 0 and the `NoInoutPort-ML` checks for inout ports at the top-level only.

You can set the `inoutlevel` Tag parameter to any positive integer number assuming that the top-level is level 0.

NOTE: If the [checkallblocks](#) Tag parameter is set, it overrides the `inoutlevel` Tag parameter.

Used by	NoOutputPort-ML
Options	Positive integer value
Default value	0

int_net_fanout

Specifies the maximum fan-out of internal nets for the [MaxFanoutCount-ML](#) Tag.

By default, the `int_net_fanout` tag parameter is set to 300 and the [MaxFanoutCount-ML](#) Tag flags internal nodes that have a fan-out greater than 300.

You can specify a different positive integer value with the `int_net_fanout` Tag parameter to have the `MaxFanoutCount-ML` Tag check for a different limit.

Used by	MaxFanoutCount-ML
Options	Positive integer value
Default value	300

inv_count

(Optional) Sets the maximum number of inverters allowed between the input/output port and register input/output for the specified tags.

By default, the `inv_count` Tag parameter is set to 1 and the tags specified in the table below allows one inverter. You can set the `inv_count` tag parameter to any positive integer number. You can also set the value of the `inv_count` tag parameter to -1 to ignore all inverters.

Used by	ScopedVarUsedBeforeDefine-ML , RegInputOutput-ML , RegInput-ML , STARCO5-1.6.2.1
Options	-1, 0, positive integer value
Default value	1

inout_name

Specifies a regular expression pattern that an `inout` port name should follow. You can combine regular expressions using `and`, `or`, `xor` and `not` operators.

For example:

```
'/^io_.*$/ and not /oo_.*/'
```

Used by	PortName
Options	Perl regular expression patterns
Default value	" "

input_name

Specifies a regular expression pattern that an `input` port name should follow. You can combine regular expressions using `and`, `or`, `xor` and `not` operators.

For example:


```
'/^i_.*$/ or /^in_.*$/'
```

Used by	PortName
Options	Perl regular expression patterns
Default value	" "

ignore_initial_blocks

Use this parameter to ignore [STARCO5-2.3.1.2a](#) violations in initial blocks.

By default, the value of the parameter is no.

Set the value of the parameter to yes to ignore [STARCO5-2.3.1.2a](#) violations in initial blocks.

Used by	STARCO5-2.3.1.2a
Options	yes, no
Default value	no

latch_name

Specifies naming convention for the latches in the design.

By default, the value of the parameter is set to " ".

Set the value of the parameter to a perl regular expression to specify the naming convention for latches in the design.

Used by	STARCO5-1.6.2.1
Options	Perl regular expression
Default value	" "

max_delta_delay

Enables the [MaxDelay-ML](#) tag to configure delta delay.

By default, the value of the parameter is set to -1 and the tag does not

report any violation.

Set the max delay value in picoseconds without ps as the value (accepted values are 0 and any positive integer) to configure delta delay.

Used by	MaxDelay-ML
Options	0, any positive integer
Default value	-1

mem_size

(Optional) Specifies the recommended memory size specification format as up ([LSB:MSB]) or down ([MSB:LSB]) for the [BitOrder-ML](#) Tag.

By default, the mem_size Tag parameter is set to down and the BitOrder-ML Tag flags memory size specifications that are in the [LSB:MSB] format.

Set the mem_size Tag parameter to up, and the BitOrder-ML Tag flags memory size specifications that are in the [MSB:LSB] format. You can also set this parameter to not_care to not check for the bit order for memory sizes.

Used by	BitOrder-ML
Options	up, down, not_care
Default value	down

min_based_width

Specifies the width of based numbers.

By default, the min_based_width parameter is set to 0 and the [NoWidthInBasedNum-ML](#) reports violation for all based numbers.

When you set this parameter to a positive integer value, the [NoWidthInBasedNum-ML](#) Tag reports violation for those based numbers whose width is greater than the value set for the min_based_width parameter.

Used by	NoWidthInBasedNum-ML
Options	Positive integer value
Default value	0

minShiftRegLength

Specifies the minimum number of bits of inferred shift register for the [UseSVCasting-ML](#) Tag and the [UseSVCasting-ML](#) Tag.

By default, the minShiftRegLength Tag parameter is set to 3. To set the minShiftRegLength Tag parameter to a different number *<num>*, specify the following command while starting VC SpyGlass Lint:

```
%> VC SpyGlass Lint ... configure_lint_tag_parameter -tag
UseSVCasting-ML -parameter minShiftRegLength -value '<num>'
...
```

Used by	UseSVCasting-ML
Options	Integer value
Default value	3

modulename

(Optional) Specifies a logical expression made of PERL regular expressions that is used by the [ModuleName-ML](#) Tag to check design unit names.

By default, the ModuleName-ML Tag checks the design unit names using the following logical expression made of PERL regular expressions:

```
"/^[a-z][a-z0-9_]*$/ and /^.{1,32}$/ and not /__/"
```

The above logical expression means that design unit names should start with a lowercase alphabet and should contain only lowercase alphabets, numbers, and underscores only (controlled by `"/^[a-z][a-z0-9_]*$/"` regular expression), should not be longer than 32 characters (controlled by `"/^.{1,32}$/"` regular expression), and should not contain consecutive underscores (controlled by `not /__/"` regular expression).

Use the modulename Tag parameter to specify a different logical

expression for the ModuleName-ML Tag.

Used by	ModuleName-ML
Options	Perl regular expression
Default value	<code>/^[a-z][a-z0-9_]*\$/</code> and <code>/^{1,32}\$/</code> and not <code>/__</code>

OperatorCount

(Optional) Specifies the maximum number of operators allowed in an expression as checked by the [ComplexExpr-ML](#) Tag.

By default, the ComplexExpr-ML Tag flags all expressions with more than 8 operators.

You can specify a different number with the `OperatorCount` Tag parameter to have the ComplexExpr-ML Tag check for a different limit.

Used by	ComplexExpr-ML
Options	Integer value
Default value	8

output_name

Specifies a regular expression pattern that an output port name should follow. You can combine regular expressions using `and`, `or`, `xor` and `not` operators.

For example:

`'/_o_.*/'`

Used by	PortName
Options	Perl regular expression patterns
Default value	<code>" "</code>

pragma_list_ml

Specifies the pragma string to be detected in order to detect user-specified pragmas and associated comment strings as checked by the [PragmaComments-ML](#) and [ParamOverrideMismatch-ML](#) tag.

If this parameter is not set to any pragma value or not specified, the default pragma synopsis, pragma in case of VHDL and only synopsis in case of Verilog is detected.

Used by	PragmaComments-ML , ParamOverrideMismatch-ML
Options	comma-separated list of pragmas
Default value	synopsys (Verilog), synopsys and pragma (VHDL)

prohibited_data_types

Use this parameter to specify the list of prohibited data types. You can specify a comma-separated list of data types as an input to this parameter. The first data type is considered as the data type that should be used and the others are the data types that should not be used directly.

Also, to specify multiple allowed data types, use semi-colon (;), as shown in the following example:

```
configure_lint_tag_parameter -tag ProhibitedDataTypes-ML -
parameter prohibited_data_types -value
"allowed_type1,dt1,dt2;allowed_type2,dt3,dt4"
```

In the an above example, dt1 and dt2 are prohibited data types and allow_type1 should be used instead. Similarly, dt3 and dt4 are prohibited data types and allow_type2 should be used instead.

The Tag does not check whether the mentioned allowed data types are available in the design or not. However, the following are the prohibited data types:

```
[wire, reg , logic ,integer , int , shortint , longint , real
, shortreal , time , realtime , byte , bit , string, user
defined types]
```

Used by	ProhibitedDataTypes-ML

Options	comma-separated list of data types
Default value	NA

range_bound

Specifies the type of boundary value to be used in the range index.

By default, this parameter not set and the [BitOrder-ML](#) Tag detects recommended bit order only.

Set the `range_bound` parameter to any or a combination of the `zero_bound`, `positive_bound`, and/or `negative_bound` values to specify the type of boundary value to be used in the range index. This enables the Tag to check the negativity and positivity of boundary values. You can select multiple options as well.

The different combinations of the values behave in the following manner:

Value	Behavior
<code>zero_bound</code>	Either value can be zero
<code>positive_bound</code>	Both values should be positive
<code>negative_bound</code>	Both values should be negative
<code>zero_bound, positive_bound</code>	Either both values are positive or one value is zero
<code>zero_bound, negative_bound</code>	Either both values are negative or one value is zero
<code>negative_bound, positive_bound</code>	No value can be zero

Used by	BitOrder-ML
Options	<code>zero_bound, positive_bound, negative_bound</code>
Default value	""

report_all_diff_timescale

Specifies whether the [DiffTimescaleUsed-ML](#) Tag reports a violation for the first instance of each timescale or all the violations for all the timescales used in the design.

NOTE: *The first timescale read in the design is considered as the base timescale. Therefore, no violation is reported for the first timescale.*

By default, the value of the parameter is set to no. Therefore, the [DiffTimescaleUsed-ML](#) Tag reports one violation per timescale, except the first timescale. That is, if a timescale is used multiple times in the design, the [DiffTimescaleUsed-ML](#) Tag reports a violation for only the first instance of the timescale.

Set the value of the parameter to yes to report violations for all the timescales, except the first timescale.

Used by	DiffTimescaleUsed-ML
Options	yes, no
Default value	no

report_all_objects

Use this parameter to check for all the objects in DDO fanout.

The default value of the parameter is no. In this case, the tag reports violation for only one object of the DDO fanout.

Used by	GlitchGenCircuitCheck
Options	yes, no
Default value	no

report_all_params

Enables the [NonOverridenParams-ML](#) tag to report violations for all the parameters.

By default, the value of the parameter is no. In this case, the tag reports only one violation for the module instance.

Set the value of the `report_all_params` parameter to yes to enable the [NonOverridenParams-ML](#) tag to report violations for all the parameters.

Used by	NonOverridenParams-ML
Options	yes, no
Default value	no

report_all_pins

Reports input pins connected to supply nets in addition to the inout pins and output pins.

By default, the parameter is set to no and the [checkPinConnectedToSupply](#) tag reports only the inout pins and output pins connected to supply nets.

Set the value of the parameter to yes to enable the [checkPinConnectedToSupply](#) tag to report input pins connected to supply nets, in addition to the inout pins and output pins.

Used by	checkPinConnectedToSupply
Options	yes, no
Default value	no

report_always_cond

Specifies whether the [AlwaysFalseTrueCond-ML](#) Tag should report always true, always false or both conditions.

By default, this parameter is set to yes and the AlwaysFalseTrueCond-ML Tag reports conditions that are either always true or false.

- If you set the `report_always_cond` parameter to `false` then only always false conditions are reported.
- If you set the `report_always_cond` parameter `true` then only always true conditions are reported.
- If you set this parameter to either `both` or `yes` then the Tag reports both, always false and always true, conditions.

Used by	AlwaysFalseTrueCond-ML

Options	yes, both, false, true
Default value	yes

report_case_block

Specifies whether the [SameAssignment-ML](#) tag reports violations for same assignment expressions in `case` blocks.

By default, the value of the parameter is set to `yes` and the tag reports violations for same assignment expressions in `case` blocks.

Set the value of the parameter to `no` to not report violations for same assignments in `case` blocks.

Used by	SameAssignment-ML
Options	yes, no
Default value	yes

report_default_branch

Specifies whether the [SameAssignment-ML](#) tag reports violations for same assignment expressions in the default branch of `case` blocks.

By default, the value of the parameter is set to `yes` and the tag reports violations for same assignment expressions in the default branch of `case` blocks.

Set the value of the parameter to `no` to not report violations for same assignment expressions in the default branch of `case` blocks.

Used by	SameAssignment-ML
Options	yes, no
Default value	yes

report_for_submodules

By default, the value of the parameter is set to no. In this case, the [NonReusableParametricModule-ML](#) tag checks only the top module.

Set the value of the parameter to yes to report violation for sub modules.

Used by	NonReusableParametricModule-ML
Options	yes, no
Default value	no

report_floating_source

Reports the floating net source.

The default value is no. Set the value of the parameter to yes to report the floating net source.

NOTE: To enable the functionality of the `report_floating_source` parameter, set the value of the [checkInHierarchy](#) parameter to yes.

Used by	UndrivenInTerm-ML
Options	yes, no
Default value	no

report_generate_block

Specifies whether the [SameAssignment-ML](#) tag reports violations for same assignment expressions in generate-case and if blocks.

By default, the value of the parameter is set to no and the tag does not report violations for same assignment expressions in generate-case and if blocks.

Set the value of the parameter to yes to report violations for same assignment expressions in generate-case and if blocks.

Used by	SameAssignment-ML
Options	yes, no
Default value	no

report_hier_path

For UnloadedInPort-ML

Reports the complete hierarchical path in the violations messages.

By default, this parameter is set to `yes` and the [UnloadedInPort-ML](#) Tag reports the complete hierarchical path in the violation messages.

If you set this parameter to `no`, only the unloadedInPort terminal is reported, which could lead to merging of violations from different modules based on the same port names.

For AvoidAsync

Enables the AvoidAsync Tag to report violations for all the instantiations of the module with asynchronous reset signal. The violation message contains hierarchical name of the reset signal.

By default, the value of the parameter is `no`. In this case, the AvoidAsync Tag ignore violations for the instantiations of the module with asynchronous reset signal.

Set the value of the parameter to `yes` to report violations for all such instantiations of the module.

Used by	UnloadedInPort-ML , AvoidAsync
Options	<code>yes</code> , <code>no</code>
Default value	<code>yes</code>

report_if_block

Specifies whether the [SameAssignment-ML](#) tag reports violations for same assignment expressions in `if-else` blocks.

By default, the value of the parameter is set to `yes` and the tag reports violations for same assignment expressions in `if-else` blocks.

Set the value of the parameter to `no` to not report violations for same assignment expressions in `if-else` blocks.

Used by	SameAssignment-ML
Options	yes, no
Default value	yes

report_mark_open

Specifies whether the [UnloadedOutTerm-ML](#) Tag should report violation for unloaded output terminals of instances in VHDL, which are intentionally left open with the OPEN keyword. The `report_mark_open` parameter also specifies whether the [UndrivenNUnloaded-ML](#) Tag should report violation for input/output terminals of an instance, which are left open intentionally using OPEN keyword and are also unloaded/undriven.

By default, the `report_mark_open` Tag parameter is set to `no` and the [UnloadedOutTerm-ML](#) and [UndrivenNUnloaded-ML](#) Tags do not report any violation for such cases.

Set the `report_mark_open` Tag parameter to `yes` to report such violations.

Used by	UnloadedOutTerm-ML , UndrivenNUnloaded-ML
Options	yes, no
Default value	no

report_max_val_overflow

Specifies whether the [ImproperRangeIndex-ML](#) tag should report a violation when the maximum value that can be represented by the index is larger than the maximum value of the range of the signal being indexed.

By default, the `report_max_val_overflow` parameter is set to `no`.

Set the `report_max_val_overflow` parameter to `yes` to report a violation when the maximum value that can be represented by the index is larger than the maximum value of the range of the signal being indexed.

Used by	ImproperRangeIndex-ML , LINT_IMPROPER_RANGE_INDEX
Options	yes, no
Default value	no

report_onehot_case

Specifies whether the [STARC05-2.8.1.4](#) tag reports a violation for one-hot case blocks without default clause.

By default, the `report_onehot_case` parameter is set to yes and the tag reports case constructs if all clauses of conditions are not described in the case statement without the default clause.

Set the value of the `report_onehot_case` rule parameter to no to not report a violation for one-hot case blocks without default clause.

Used by	STARC05-2.8.1.4
Options	yes, no
Default value	no

report_non_const_read

Reports non-constant variable read in if/case conditions inside the reset block.

Consider the following example:

```
always @(posedge clk or negedge rst)
begin
    if (!rst)
    begin
        r_data <= 1'b0;
        if (var1) // violation
            r_data <= 1'b1;
        else
            r_data <= 1'b0;
    end
end
else
```

```
        r_data <= r_data1;  
    end
```

Used by	NonConstReset-ML
Options	yes, no
Default value	no

report_nonstatic_assign

Specifies whether the [EnableXPropagation-ML](#) Tag should report the non-static value assigned to the target signal in the `else` part or `default` clause.

By default this parameter is set to `yes` and the Tag reports violation if non static value is assigned to the target signal in the `else` part or `default` clause.

Set the `report_nonstatic_assign` parameter to `no` to not report violations for such scenarios.

Used by	EnableXPropagation-ML
Options	yes, no
Default value	yes

report_open_port

Specifies whether the [PartConnPort-ML](#) Tag should report violations for open terminal connections.

By default this parameter is set to `no` and the Tag does not report violations for open terminal connections.

Set the `report_open_port` parameter `yes` to enable the [PartConnPort-ML](#) Tag to report violations for open terminal connections.

Set the value of the parameter to `disable` to not report violation for deliberately unconnected terminal connections.

Used by	PartConnPort-ML
Options	yes, no, disable
Default value	no

report_only_first

Reports only one instance of the forward cone starting from the constant input.

By default, the value of the parameter is no. In this case, the [ConstantInput-ML](#) Tag reports all the instances of constant-driven scenarios In the design.

Set the value of the parameter to yes to report only one instance of the forward cone starting from the constant input.

Used by	ConstantInput-ML
Options	yes, no
Default value	no

report_only_from_one_hierarchy

Specifies whether the specified tag should suppress the hierarchy field and report only one violation per assignment.

By default, the value of the parameter is no, and the tag reports violations for multiple hierarchies.

Set the value of the parameter to yes to suppress the hierarchy field and report only one violation per assignment.

Used By	ImproperRangeIndex-ML , W164a , W224 , W287a , STARCO5-2.1.1.2 , SetBeforeRead-ML
Options	yes, no
Default Value	no

report_sequential_always

Specifies whether the [EnableXPropagation-ML](#) Tag should report violations inside sequential always blocks.

By default this parameter is set to no and the Tag does not report violations inside sequential always blocks.

Set the `report_sequential_always` parameter yes to enable the [EnableXPropagation-ML](#) Tag to also report violations inside sequential always blocks.

Used by	EnableXPropagation-ML
Options	yes, no
Default value	no

report_underflow

Specifies whether the [ValueSizeOverflow-ML](#) Tag should also report underflow.

By default this parameter is set to no and the Tag does not report underflow.

Set the `report_underflow` parameter to yes to enable the [ValueSizeOverflow-ML](#) Tag to also report underflow under the following conditions:

- `conv_integer` is used as a first argument to `conv_std_logic_vector`.
- If first argument of `conv_std_logic_vector` is a binary, then it should have at least one operand as `conv_integer`.
- Width of the only argument of `conv_integer` function is greater than 32.
- The overall width of the first argument (after considering the width changes due to binary) is less than the second argument.

Used by	ValueSizeOverflow-ML

Options	yes, no
Default value	no

report_underflow_expressions

Set the value of the parameter to yes to report violations for underflow scenarios also, that is, number of bits in the select expression is insufficient to cover whole range.

Used by	ImproperRangeIndex-ML
Options	yes, no
Default value	no

runcapaTag

(Optional) Specifies whether the [CAPA-ML](#) Tag should run.

By default, the runcapTag Tag parameter is set to no and the CAPA-ML Tag is disabled.

Set the runcapTag Tag parameter to yes to enable the [CAPA-ML](#) Tag.

Used by	CAPA-ML
Options	yes, no
Default value	no

skip_lib_cell_checking

Specifies whether the [UndrivenInTerm-ML](#) Tag should report violations for library cells.

By default, the skip_lib_cell_checking parameter is set to no.

Set the skip_lib_cell_checking parameter to yes to not report violations for library cells.

Used by	UndrivenInTerm-ML
Options	yes, no
Default value	no

showReducedSchematic

Specifies the type of schematic to be displayed in VC SpyGlass Lint. This Tag parameter is used by the [UnsuppCompDir-ML](#) and [UnusedFlopOutput-ML](#) Tag.

By default, the `showReducedSchematic` Tag parameter is set to `min`. You can change the value to `max` or `no`. For example,

```
showReducedSchematic=<value>
```

where `<value>` can be `min`, `max`, or `no`.

The `max` value indicates maximum schematic annotation. In this mode, sequential element and the full fan-out cone of the concerned sequential element output pins to other unreachable sequential elements, hanging nets, and blocking gates are displayed.

For example, in the following schematic, suppose `in2` is blocked, thus the sequential element `tmp_reg` will be highlighted because its path is blocked. And, the other sequential element `o4_reg` will not be displayed in the schematic.

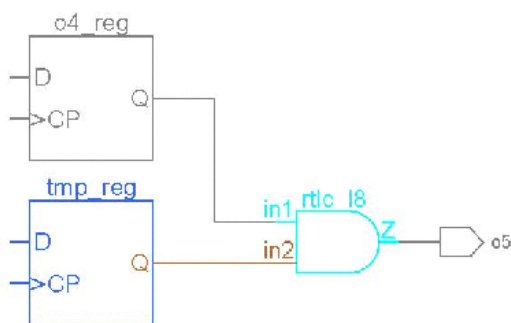


FIGURE 2. Incremental schematic

The `min` value (default) indicates minimum schematic annotation. In this mode, the sequential element and the first cell in the fan-out cone of the concerned sequential element output pin is displayed.

The `no` value indicates no schematic annotation.

Set this parameter to `max` if the reason (`set_case_analysis`) for blocked output pin needs to be reported.

Used by	UnsuppCompDir-ML , UnUsedFlopOutput-ML
Options	min, max, no
Default value	min

states_to_check

Specifies whether the [LINT_FSM_WITHOUT_EXIT_STATE](#) tag should report violation if the FSM is stuck in any states.

By default, the `states_to_check` parameter is set to `CHECK_ALL` and the `LINT_FSM_WITHOUT_EXIT_STATE` tag reports a violation if the FSM is stuck in any states.

If you set this parameter to `IGNORE_UNREACHABLE_STATE`, the tag does not report any violation when the FSM is stuck in any unreachable states.

Used by	LINT_FSM_WITHOUT_EXIT_STATE
Options	<code>CHECK_ALL</code> , <code>IGNORE_UNREACHABLE_STATE</code>
Default	<code>CHECK_ALL</code>

synch_reset_list

Provides a comma-separated list of synchronous reset signals used in the design as used by the [ResetFlop-ML](#) Tag.

Used by	ResetFlop-ML

Options	Comma-separated list of reset names
Default value	null

synth_dir_list

(Optional) Specifies the list of recognized synthesis directives for the [UnrecSynthDir-ML](#) Tag.

By default, the UnrecSynthDir-ML Tag flags synthesis directives other than the following:

async_set_reset	async_set_reset_local	async_set_reset_local_all
dc_script_begin	dc_script_end	dont_infer_multibit
enum	full_case	infer_multibit
infer_mux	label	label_applies_to
map_to_module	one_cold	one_hot
parallel_case	resource	return_port_name
state_vector	sync_set_reset	sync_set_reset_local
sync_set_reset_local_all	template	translate_off
translate_on		

Use the `synth_dir_list` Tag parameter to specify your own list of recognized synthesis directives (overriding the default list). Then, the UnrecSynthDir-ML Tag flags all other synthesis directives.

Used by	UnrecSynthDir-ML
Options	Space-separated list of directives
Default value	translate_off, translate_on, parallel_case, full_case, state_vector, enum, template, dc_script_begin, dc_script_end, map_to_module, return_port_name, infer_mux

synth_name_length

(Optional) Specifies the maximum length allowed for a module name as checked by the [SynthElabDuName-ML](#) Tag.

By default, the `synth_name_length` Tag parameter is set to 40 and the SynthElabDuName-ML Tag flags module names that have length greater than 40.

You can specify a different positive integer value with the `synth_name_length` Tag parameter to have SynthElabDuName-ML Tag check for a different length.

Used by	SynthElabDuName-ML
Options	Positive integer value
Default value	40

use_detailed_message

Enables the [SameAssignment-ML](#) tag to include details of the location of the same assignment in the violation message.

By default, the parameter is set to `no` and the violation message reported by the tag does not include the information about the first occurrence of the assignment.

Set the parameter to `yes` to enable the tag to include information about the first occurrence of the assignment and the corresponding file name in the violation message.

Used by	SameAssignment-ML
Options	yes, no
Default value	no

use_sgdc_reset

Enables the [ResetFlop-ML](#) Tag to provide a list of reset signals.

By default, the parameter is set to `no`. Set the parameter to `yes` and specify the reset signal names by using the reset constraint in the SGDC file.

The parameter supports only the top-level scalar nets as supported by the reset constraint. The propagation of reset signals is performed through basic combinational gates (Buffers, Inverters, OR, AND, XOR, NAND, NOR, XNOR).

Used by	ResetFlop-ML
Options	yes, no
Default value	no

vector_size

(Optional) Specifies the recommended vector size specification format as up ([LSB:MSB]) or down ([MSB:LSB]) for the [BitOrder-ML](#) Tag.

By default, the `vector_size` Tag parameter is set to down and the BitOrder-ML Tag flags vector size specifications that are in the [LSB:MSB] format.

Set the `vector_size` Tag parameter to up, and the BitOrder-ML Tag flags vector size specifications that are in the [MSB:LSB] format.

You can also set this parameter to `not_care` to not check for the bit order for vector sizes.

Used by	BitOrder-ML
Options	up, down, not_care
Default value	down

word_size

(Optional) Specifies the recommended word size specification format as up ([LSB:MSB]) or down ([MSB:LSB]) for the [BitOrder-ML](#) Tag.

By default, the `word_size` Tag parameter is set to down and the BitOrder-ML Tag flags word size specifications that are in the [LSB:MSB] format.

Set the `word_size` Tag parameter to up, and the BitOrder-ML Tag flags

word size specifications that are in the [MSB:LSB] format.

You can also set this parameter to `not_care` to not check for the bit order for word sizes.

Used by	BitOrder-ML
Options	up, down, not_care
Default value	down

wrapperDuCellList

(Optional) Sets the names of the allowed cells in a design unit for [WrapInstance-ML](#) Tag.

The `wrapperDuCellList` Tag parameter takes a comma-separated name lists where each design unit name is followed by names of the black box modules that can be instantiated inside the design unit as in the following example:

```
configure_lint_tag_parameter -tag WrapInstance-ML -parameter
wrapperDuCellList -value "DU1 c1 c2 c3, DU2 c2 c5"
```

Here, instances of cells `c1`, `c2`, and `c3` are allowed in the design unit `DU1` where instances of cells `c2` and `c5` are allowed in the design unit `DU2`.

Then, the `WrapInstance-ML` Tag flags all other black box instances (instances of cells other than the respective allowed cells or instances of design units whose RTL description is not available) in design units `DU1` and `DU2`.

Used by	WrapInstance-ML
Options	Comma-separated list of design units followed by a list of black boxes
Default value	" "

actlowname

(Optional) Specifies a logical expression made of PERL regular expressions that is used by the [ActLowName](#) Tag to check active low signal names.

By default, the ActLowName Tag checks the active low signal names using the following logical expression made of PERL regular expressions:

`"/_n$/ or /_b$/"`

The above logical expression means that active low signal names should end with either `_n` or `_b`.

Used by	ActLowName
Options	Perl regular expression pattern for active low signal name
Default value	<code>"/_n\$/ or /_b\$/"</code>

allReserveWords

(Optional) Specifies whether the [ReserveName](#) (Verilog) Tag flags the first occurrence (default) or all occurrences of the reserved keywords.

Used by	ReserveName (Verilog)
Options	yes, no
Default value	no

archname

(Optional) Specifies a logical expression made of PERL regular expression that the [ArchName](#) Tag uses to check architecture names.

By default, the ArchName Tag checks the architecture names using the following logical expression made of PERL regular expression:

`"/^(rtl|str|sim|tb)$/i and /^{1,16}$/ and not /_/"`

Use the archname Tag parameter to specify a different logical expression for the ArchName Tag.

Used by	ArchName
Options	Perl regular expression pattern for architectural name
Default value	<code>"/^(rtl str sim tb)\$/i and /^{1,16}\$/ and not /_/"</code>

asyncname

Specifies a logical expression made of PERL regular expression that the [AsyncName](#) Tag uses to check asynchronous signal names.

By default, the AsyncName Tag checks the asynchronous signal names using the following logical expression made of PERL regular expression:

```
"/_a$/i"
```

The above logical expression means that asynchronous signal names should end with `_a`.

Use the `asyncname` Tag parameter to specify a different logical expression for the AsyncName Tag.

Used by	AsyncName
Options	Perl regular expression pattern for asynchronous name
Default value	"/_a\$/i"

check_globals

Specifies if the [ConstName](#) and [ParamName](#) tags should check the parameters defined in the global scope.

By default, the value of the parameter is set to no and tags do not report violations for parameters defined in the global scopes.

Set the value of the parameter to yes to enable the tags to report a violation for parameters defined in the global scope.

Used by	ConstName , ParamName
Options	yes, no
Default value	no

check_task

Specifies whether the [FuncName](#) Tag should check for task name.

By default, the `check_task` parameter is set to no, and the Tag does not

check for task name.

Set this parameter to `yes` to enable the Tag checking for task name.

Used by	FuncName
Options	yes, no
Default value	no

check_regoutput_modules

Specifies module names that should be included for Tag checking by the [RegOutputs](#) Tag.

By default, this parameter is not set.

Set this parameter to module names or a PERL regular expression specifying the module names that will be included for the Tag checking.

NOTE: *If the `ignore_regoutput_modules` and `check_regoutput_modules` parameters are specified together then the priority is given to those modules that are specified through the `check_regoutput_modules` parameter.*

Used by	RegOutputs
Options	Comma separated module names, PERL regular expression
Default value	

checkalldimension

Specifies whether the [ArrayIndex](#) Tag checks for only the word size (right most dimension of packed array) of a variable or all the dimensions (both packed and unpacked) or only packed/unpacked.

By default, the `checkalldimension` Tag parameter is set to `no` and the [ArrayIndex](#) Tag checks only the vector size (right-most dimension of packed array) of a variable.

For Verilog

- If this parameter is set to `yes`, the [ArrayIndex](#) Tag checks for all the dimensions (both packed and unpacked) and reports all the violation.

- If the parameter is set to `packed`, the Tag checks for packed dimensions and reports violations.
- If the parameter is set to `unpacked`, the Tag check for unpacked dimensions and reports violations.

For VHDL

The `checkalldimension` parameter takes only two values, `yes` or `no`. If set to `yes`, the [ArrayIndex](#) Tag checks for both packed and unpacked otherwise it checks only for the vector size (right-most dimension of packed array) of a variable.

Used by	ArrayIndex
Options for Verilog	yes, no, packed, unpacked
Options for VHDL	yes, no
Default value	no

checkLowToHigh

Specifies whether the range of a bus should be declared from low to high, as checked by the [ArrayIndex](#) Tag.

By default, the `checkLowToHigh` Tag parameter is set to `no`, and the Tag checks the range from high to low.

You can set the value of the `checkLowToHigh` Tag parameter to `yes` to check for the range from low to high.

Used by	ArrayIndex
Options	yes, no
Default value	no

check_reserve_word

Specifies whether the [PortName](#) tag should report a violation when the name of the port is one of the `c/c++` reserve word.

By default, the value of the parameter is no.

Set the value of the parameter to yes to enable the tag to report a violation when the name of the port is one of the following c/c++ reserve words.

Verilog:

auto, break, char, const, continue, default, do double, enum, extern, float, goto, int, long, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, and malloc.

VHDL:

auto, char, default, double, float, goto, long, short, signed, sizeof, struct, switch, unsigned, volatile, and malloc.

Used by	PortName
Options	yes, no
Default value	no

checkTopLevel

Enables the [ArrayIndex](#) Tag to report violations only on module/entity boundaries.

By default, the value of the parameter is no.

Set the value of the parameter to yes to report violations only on module/entity boundaries.

Used by	ArrayIndex
Options	yes, no
Default value	no

clkname

(Optional) Specifies a logical expression made of PERL regular expression

that is used by the [ClkName](#) Tag to check clock names.

By default, the ClkName Tag checks the clock names using the following logical expression made of PERL regular expression:

```
"/^clk$/i or /^clk_/i"
```

The above logical expression means that clock type signal names should start with clk or clk_.

Use the clkname Tag parameter to specify a different logical expression for the ClkName Tag.

Used by	ClkName
Options	Perl regular expression pattern for a clock name
Default value	"/^clk\$/i or /^clk_/i"

clkname_vector

Specifies a regular expression pattern that is used by the [ClkName](#) tag to check vector type clock names.

By default, the ClkName tag checks the vector type clock names using the following logical expression made of PERL regular expression:

```
"/^clk$/i or /^clk_/i"
```

The above logical expression means that vector type clock names should start with clks or clks_.

Use the clkname_vector parameter to specify a regular expression pattern that a vector type clock name should follow. If a vector type clock name follows a pattern RE1 as well as RE2 but not RE3, then the parameter value should be passed as 'RE1 and RE2 and not RE3'.

NOTE: All the REs should follow the Perl syntax. Currently, 'and', 'or', 'xor' and 'not' operators are supported within the expressions.

Used by	ClkName
Options	Perl regular expression pattern for a vector type clock name
Default value	"/^clks\$/i or /^clks_/i"

comment_max_length

(Optional) Specifies the maximum permissible length for a pure comment line in the RTL code, as checked by the [LineLength](#) Tag.

By default, the comment_max_length Tag parameter is not set.

You can set the value of the comment_max_length Tag parameter to any positive integer value.

Used by	LineLength
Options	Positive integer value
Default value	" "

constname

(Optional) Specifies a logical expression made of PERL regular expressions that is used by the [ConstName](#) Tag to check constant names.

By default, the ConstName Tag checks the constant names using the following logical expression made of PERL regular expressions:

`"/^[A-Z][A-Z0-9_]*$/ and /^.{1,8}$/ and not /__/"`

The above logical expression means that constant names should start with an uppercase alphabet and should contain only uppercase alphabets, numbers, and underscores only (controlled by `"/^[A-Z][A-Z0-9_]*$/` regular expression), should not be longer than 8 characters (controlled by `"/^{1,8}$/` regular expression), and should not contain consecutive underscores (controlled by `not /__/"` regular expression).

Use the constname Tag parameter to specify a different logical expression for the ConstName Tag.

Used by	ConstName
Options	Perl regular expression pattern for a constant name
Default value	<code>"/^[A-Z][A-Z0-9_]*\$/ and /^.{1,8}\$/ and not /__/"</code>

enableE2Q

(Optional) Causes the specified tag to proceed from enable pin to Q (output) pin of a latch, while detecting the combinational loop.

By default, the `enableE2Q` tag parameter is unset and the specified tag does not traverse through latches for locating combination loops.

You can set the value of the `enableE2Q` Tag parameter to `yes` or `1` (to enable the mode) or to `no` or `0` (to disable the mode).

Used by	CombLoop , STARC05-1.2.1.3 , Rule_24 , Rule_28
Options	yes, no
Default value	no
Default Value in GuideWare2.0	yes

enable_latch_based_clusters

Causes the [CombLoop](#) tag to report a violation for loops with more than one latch and also generate a report that is clustered into two parts.

By default, the `enable_latch_based_clusters` tag parameter is set to `no`.

You can set the value of the tag parameter to `yes` to report a violation for loops with more than one latch and also generate a report that is clustered into two parts:

- loops with two or more latches
- other loops

Used by	CombLoop
Options	yes, no
Default value	no

funcname

(Optional) Specifies a logical expression made of PERL regular expressions that is used by the [FuncName](#) Tag to check function names.

By default, the `FuncName` Tag checks the subprogram names using the following logical expression made of PERL regular expressions:

`"/^[a-z][a-z0-9_]*$/ and /^.{1,16}$/ and not /___/"`

The above logical expression means that subprogram names should start with a lowercase alphabet and should contain only lowercase alphabets, numbers, and underscores only (controlled by `"/^[a-z][a-z0-9_]*$/` regular expression), should not be longer than 16 characters (controlled by `"/^.{1,16}$/` regular expression), and should not contain consecutive underscores (controlled by `not /___/` regular expression).

Use the `funcname` Tag parameter to specify a different logical expression for the `FuncName` Tag.

Used by	<i>FuncName</i>
Options	Perl regular expression pattern for a function name
Default value	<code>"/^[a-z][a-z0-9_]*\$/ and /^.{1,16}\$/ and not /___/"</code>

gates

(Optional) Specifies the list of gates or Perl expressions whose instantiations are to be ignored by the [*NoGates*](#) Tag.

By default, the `gates` Tag parameter is set to null, that is, no gate names or Perl expressions are specified. You can set the value of the `gates` Tag parameter to a space-separated list of gate names or Perl expressions.

When the `gates` parameter begins with a `/` (slash), then the parameter will be considered as Perl regular expression.

Used by	<i>NoGates</i>
Options	Space-separated list of gates or Perl regular expression
Default value	<code>" "</code>

generate_header_check_report

Specifies whether the [*FileHdr*](#) Tag should generate a report containing differences of actual file headers and expected file headers of all violated

file headers.

By default, the `generate_header_check_report` parameter is set to `no`.

Set this parameter to `yes` to enable the *FileHdr* Tag to generate a report containing differences of actual file headers and expected file headers of all violated file headers having basic header requirements met. For this parameter to work, the *header_check_file* parameter should be set to an updated header check file.

Used by	<i>FileHdr</i>
Options	yes, no
Default value	no

header_check_file

(Optional) Specifies the name of the file that contains a different PERL subroutine for file header checking with the *FileHdr* Tag in Verilog mode.

By default, the *FileHdr* Tag checks the file header using its built-in `ve_header_check` subroutine.

To have the *FileHdr* Tag check file headers as per your requirements, copy the `ve_header_check` subroutine from the Tagdeck file to an ASCII text file, modify the subroutine contents as required, and save the file (for example as `myHeaderCheck` file in the `/usr/john/scripts` directory). Now, supply the file name as the value of `header_check_file` Tag parameter (`/usr/john/scripts/myHeaderCheck` in the current example).

NOTE: Make sure that you do not change the subroutine name (`ve_header_check`).

NOTE: You can supply the file name either as an absolute file name or as a relative file path name from the directory containing the source HDL file.

Used by	<i>FileHdr</i>
Options	Valid file name
Default value	" "

header_check_file_vh

(Optional) Specifies the name of the file that contains a different PERL subroutine for file header checking with the [FileHdr](#) Tag in VHDL mode.

By default, the FileHdr Tag checks the file header using its built-in `vh_header_check` subroutine.

To have the FileHdr Tag check file headers as per your requirements, copy the `vh_header_check` subroutine from the Tagdeck file to an ASCII text file, modify the subroutine contents as required, and save the file (for example as `myHeaderCheck` file in the `/usr/john/scripts` directory). Now, supply the file name as the value of `header_check_file` Tag parameter (`/usr/john/scripts/myHeaderCheck` in the current example).

NOTE: *Make sure that you do not change the subroutine name (`vh_header_check`).*

NOTE: *You can supply the file name either as an absolute file name or as a relative file path name from the directory containing the source HDL file.*

Used by	FileHdr
Options	Valid file name
Default value	" "

ignore_case_inside

Use this parameter to report violation for case inside statements.

By default, the value of the parameter is set to no. In this case, the tag reports violation for case inside statements.

Set the value of the parameter to yes to ignore violations for case inside statements.

Used by	STARC05-2.10.1.4c
Options	yes, no
Default value	no

ignore_cells

Specifies whether the *RegOutputs* Tag ignores the specified cells.

By default this parameter is not set. Set the `ignore_cells` parameter to comma separated list or regular expressions of cell names to allow the Tag to ignore these cells.

Used by	<i>RegOutputs</i>
Options	comma separated list of cells
Default value	NULL

ignore_conv_function

Specifies whether the *NoTopLogic* Tag reports violation for a function call.

By default, the `ignore_conv_function` Tag parameter is set to `no` and the *NoTopLogic* Tag does not ignore any function.

You can set the value of `ignore_conv_function` Tag parameter to `yes` to ignore simple IEEE Verilog and VHDL conversion functions. This parameter also accepts a comma separated list of functions. In this case, all the IEEE simple conversion functions and user-defined functions specified by this argument are ignored by the *NoTopLogic* Tag.

Used by	<i>NoTopLogic</i>
Options	no, yes, comma separated list of functions
Default value	no

ignore_file

Enable the *PortName* Tag to not report violations for a Verilog file. This parameter supports regular expressions and is applicable for Verilog only.

Used by	<i>PortName</i>
Options	no, yes
Default value	no

ignore_function_port

(Optional) Specifies whether the *NoDup* Tag flags duplicate names for port and function input argument.

By default, the `ignore_function_port` Tag parameter is set to `yes` and the `NoDup` Tag does not flag duplicate names in port and function input argument.

You can set the value of `ignore_function_port` Tag parameter to `no` to report duplicate names for port and function input argument.

Used by	<i>NoDup</i>
Options	yes, no
Default value	yes

ignore_file_ext

Specifies if the *CheckImplicitPort-ML* tag should not report the files having extensions specified through this parameter.

By default, the `ignore_file_ext` parameter is set to `no`

Set the parameter to `yes` to enable the tag to not report a violation for the files having extensions specified through this parameter.

Used by	<i>CheckImplicitPort-ML</i>
Options	yes, no, comma or space separated list of Tags
Default value	no

ignore_hanging_flop

Specifies whether the *IntReset* Tag does not report violation if the output of the flip-flop is hanging.

By default, the `ignore_hanging_flop` Tag parameter is set to `no` and the *IntReset* Tag reports a violation if the output of the flip-flop is hanging.

If you set this parameter to **yes**, the Tag does not report a violation for the flip-flop if it's output is hanging.

Used by	IntReset
Options	yes, no, comma-separated list of Tag names
Default value	no

ignore_iopad

(Optional) Specifies whether the [CombLoop](#), [Rule_24](#) and [Rule_28](#) tags report combinational loops that traverse input and output terminals of a pad cell.

By default, the tags report combinational loops that traverse input and output terminals of a pad cell. If you set this parameter to **yes**, the tags do not report such combinational loops.

Used by	CombLoop , Rule_24 , Rule_28
Options	yes, no
Default value	no

ignore_incr_decr_by_one

Use this parameter to not report violation when increment and decrement by 1 is done within for loop.

Used by	STARC05-2.9.1.2c
Options	yes, no
Default value	no

ignoreInversionOperation

Enables the [NoExprInPort-ML](#) tag to ignore violations for inversion operations in port assignments.

By default, the value of the parameter is set to no. In this case, the

NoExprInPort-ML tag reports violations for inversion operations in port assignments.

Set the value of the parameter to no to ignore these violations.

Used by	NoExprInPort-ML
Options	yes, no
Default value	no

ignoreRealLatch

Enables the specified tag to ignore real latches.

By default, the value of the parameter is set to no. In this case, the specified tag considers real latches for tag checking.

Set the value of the parameter to yes to ignore real latches.

Used by	IfWithoutElse-ML, InferLatch
Options	<file_path>
Default value	" "

ignore_libcells

Specifies whether the *InstName* Tag should ignore lib cell instances in the modules.

By default, the ignore_libcells parameter is set to no.

Set this parameter to yes to ignore violations for lib cell instances in the module.

Used by	InstName
Options	yes, no
Default value	no

ignore_macro_names

Specifies whether the [ConstName](#) tag should ignore violations for macro names.

By default, the `ignore_macro_names` parameter is set to `no`.

Set this parameter to `yes` or the tag name (`ConstName`) to not report any violations for macro names.

Used by	ConstName
Options	yes, no, ConstName
Default value	no

ignore_mux_select

By default, the value of the parameter is set to `no`. In this case, the [SetResetConverge-ML](#) tag traverses through the select pin of a mux to check whether set and reset pin of a flop/latch converge or not.

Set the value of the parameter to `yes` if you do not want the tag to traverse through select pin of multiplexers.

Used by	SetResetConverge-ML
Options	yes, no
Default value	no

ignore_non_reset

Enables the [STARC05-1.3.1.3](#) tag to report violations only for asynchronous reset/preset signals that are used as synchronous reset/preset signals.

By default, the value of the parameter is set to `no`. In this case, the tag report violations for asynchronous reset/preset signals used as non-reset/preset or synchronous reset/preset signals.

Set the value of the parameter to `yes` to enable the tag to report violation only for asynchronous reset/preset signals that are used as synchronous reset/preset signals.

Used by	STARC05-1.3.1.3
Options	yes, no
Default value	no

ignore_libcell_inout_ports

Enables the [NoInoutPort-ML](#) tag to ignore violations for cases where the inout port is connected to a library cell inside the module.

The default value is no. In this case, the tag reports violation for cases where the inout port is connected to a library cell inside the module. Set the value of the parameter to yes to ignore violations for such cases.

Used by	NoInoutPort-ML
Options	yes, no
Default value	no

ignore_modules

Specifies the modules for which violations should be filtered by the [InstName](#).

By default, the `ignore_modules` parameter is not set.

Set this parameter to a comma-separated list of PERL regular expressions specifying the names of modules for which violations should be filtered.

Used by	InstName
Options	Comma-separated list of PERL regular expressions
Default value	""

ignoreMultipleInstances

For `ClkHierName`:

(Optional) Specifies whether the [ClkHierName](#) Tag flags only one message

when a clock is connected to multiple instances of a design unit and the clock name is not preserved across the hierarchy.

By default, the `ignoreMultipleInstances` Tag parameter is set to `no` and the `ClkHierName` Tag flags separate message for each case where a clock does not preserve its name across the hierarchy.

You can set the value of the `ignoreMultipleInstances` parameter to `yes` (to report only one message for multiple instances) or `No` (to report each case separately).

For STARC05-2.2.2.2a

(Optional) Specifies whether the STARC05-2.2.2.2a Tag flags a separate message for each tag-violating signal in each instance of a design unit or flags only one message for a Tag-violating signal of a design unit instantiated multiple times in the design.

By default, the `ignoreMultipleInstances` Tag parameter is set to `no` and the STARC05-2.2.2.2a tag reports a separate message for each tag-violating signal in each instance of a design unit.

You can set the `ignoreMultipleInstances` tag parameter to `yes` so that the STARC05-2.2.2.2a tag flags only one message for a tag-violating signal of a design unit instantiated multiple times in the design.

Used by	ClkHierName , STARC05-2.2.2.2a
Options	yes, no
Default value	no

ignoreNonStaticOverlap

Enables the tag to not report violations for signal, which meets the following conditions:

- Signal is read with non-static bit select expression.
- Some overlapping bits are set before the read statement.

Used by	SetBeforeRead-ML
Options	yes, no
Default value	no

ignore_memory

For CombLoop

Specifies whether the *CombLoop* Tag ignores the loops through combinational timing arcs of a sequential memory.

By default, this parameter is set to no and the *CombLoop* Tag reports the loops through combinational timing arcs of a sequential memory.

Set this parameter to yes to ignore such violations.

For STARC05-1.3.1.3

Specifies whether the *STARC05-1.3.1.3* Tag should ignore violations for reset signal usages inside or after memory cells.

By default, the value of the ignore_memory parameter is set to no.

Set this parameter to yes to ignore violations for reset signal usages inside or after memory cells.

Used by	<i>CombLoop</i> , <i>STARC05-1.3.1.3</i>
Options	yes, no
Default value	no

ignoreSRlatch

(Optional) Specifies to ignore the SR-Latch combinational loops while Tag checking with the *CombLoop*, *Rule_24* and *Rule_28* tags.

By default, the ignoreSRlatch tag parameter is not set.

You can set the value of the ignoreSRlatch parameter to yes (to ignore the SR-Latch combinational loops) or no (to report such loops).

Used by	<i>CombLoop</i> , <i>Rule_24</i> , <i>Rule_28</i>
Options	yes, no
Default value	no

ignore_regoutput_modules

Specifies module names that should be excluded from Tag checking by the [RegOutputs](#) Tag.

By default, the `ignore_regoutput_modules` parameter is not set.

Set this parameter to module names or a PERL regular expression specifying the module names that will be excluded from the Tag checking.

NOTE: *If the `ignore_regoutput_modules` and `check_regoutput_modules` parameters are specified together then the priority is given to those modules that are specified through the `check_regoutput_modules` parameter.*

Used by	RegOutputs
Options	Comma separated module names, PERL regular expression
Default value	NULL

ignore_stmt

Specifies if the [OneStmtLine](#) Tag should ignore violations.

By default, the `ignore_stmt` parameter set to `no`.

You can set the value of this parameter to `yes` to enable the [OneStmtLine](#) Tag to ignore violations for the following scenarios:

- "begin" is allowed on the line creating the block (if/always/generate block)
- "end" is allowed with "else" and "else" with "begin", in any combination
- Single-assignments are allowed with the generating block statement (if/always), but "begin <command> end" is reported as a violator
- For the struct/union/enum type declaration, the first element declaration is allowed on the line creating the struct/union/enum structure

NOTE: *The `ignore_stmt` parameter is applicable for Verilog only.*

Used by	OneStmtLine

Options	yes, no
Default value	no

indent_level

(Optional) Specifies the number of spaces per indent level for the [Indent](#) Tag.

By default, each indent level is assumed to be 4 spaces. You can set the value of the `indent_level` Tag parameter to any positive integer value.

Used by	Indent
Options	Positive integer value
Default value	4

inst_max_length

(Optional) Specifies the maximum permissible length for module instance names to be checked in the [InstNameLength](#) Tag.

By default, the `inst_max_length` Tag parameter is set to 8 and the `InstNameLength` Tag flags the instance names longer than 8 characters.

You can set the `inst_max_length` Tag parameter to any positive integer number. Then, the `InstNameLength` Tag flags names longer than the specified number.

Used by	InstNameLength
Options	Positive integer value
Default value	8

inst_min_length

(Optional) Specifies the minimum permissible length for module instance names to be checked in the [InstNameLength](#) Tag.

By default, the `inst_min_length` is set to 4 and the `InstNameLength`

Tag flags the instance names shorter than 4 characters.

You can set the `inst_min_length` Tag parameter to any positive integer number. Then, the `InstNameLength` Tag flags names shorter than the specified number.

Used by	<i>InstNameLength</i>
Options	Positive integer value
Default value	4

invalid_cell_list_warning

Specifies the invalid cells that can be either cell type or PERL regular expression for cell names.

By default, the `invalid_cell_list_warning` parameter is not set.

Set this parameter to a comma-separated list of strings to define the invalid cells which can be either cell type or PERL regular expression for cell names. A warning is reported for cells specified in this list.

Used by	<i>InitialConnectionCheck</i>
Options	Comma separated list of strings
Default value	""

instname

(Optional) Specifies a logical expression made of PERL regular expressions that is used by the *InstName* Tag to check instance names.

By default, the `InstName` Tag checks the instance names using the following logical expression made of PERL regular expressions:

```
"/^U_/ and /^.{1,16}$/ and not /__/"
```

The above logical expression means that instance names should start with `U_` (controlled by `/^U_/` regular expression), should not be longer than 16 characters (controlled by `/.{1,16}$/` regular expression), and should not contain consecutive underscores (controlled by `not /__/` regular

expression).

Use the `instname` Tag parameter to specify a different logical expression for the `InstName` Tag.

Used by	<i>InstName</i>
Options	Perl regular expression pattern for an instance name
Default value	<code>"/^U_/ and /^.{1,16}\$/ and not /__/"</code>

iocells

(Optional) Specifies the names of the input/output cells that reside within the input/output module as used by the [*pwrndPinConnToSeqOrIOCells*](#) tag.

You can specify the cell names as a space-separated list of Perl regular expressions enclosed in double-quotes.

By default, the value of the `iocells` parameter is set to the following:

```
"bq.*box oq.*box iq.*box bq.*s oq.*s ic18shy"
```

Used by	<i>pwrndPinConnToSeqOrIOCells</i>
Options	Space-separated list of regular expressions
Default value	<code>"bq.*box oq.*box iq.*box bq.*s oq.*s ic18shy"</code>

line_max_length

(Optional) Specifies the maximum permissible length for a line in RTL code to be checked in the [*LineLength*](#) Tag.

By default, the `line_max_length` Tag parameter is set to 132 and the `LineLength` Tag flags RTL code lines longer than 132 characters.

You can set the `line_max_length` Tag parameter to any positive integer number. Then, the `LineLength` Tag flags RTL code lines longer than the specified number.

Used by	LineLength
Options	Positive integer value
Default value	132

limit_if_branch

Specifies the number of `if` branches for nesting level.

By default, the `limit_if_branch` parameter is set to 3.

Set this parameter to a positive integer value to enable the [CaseOverIf](#) Tag to report violations when `if` branches are more than this specified value.

Used by	CaseOverIf
Options	Positive integer value
Default value	3

name_max_length

(Optional) Specifies the maximum permissible length for names to be checked in the [NameLength](#) Tag.

By default, the `name_max_length` Tag parameter is set to 8 and the `NameLength` Tag flags names longer than 8 characters.

You can set the `name_max_length` Tag parameter to any positive integer number. Then, the `NameLength` Tag flags names longer than the specified number.

Used by	NameLength
Options	Positive integer value
Default value	8

name_min_length

(Optional) Specifies the minimum permissible length for names to be checked in the [NameLength](#) Tag.

By default, the `name_min_length` Tag parameter is set to 3 and the `NameLength` Tag flags names smaller than 3 characters.

You can set the `name_min_length` Tag parameter to any positive integer number. Then, the `NameLength` Tag flags names smaller than the specified number.

Used by	NameLength
Options	Positive integer value
Default value	3

nocheck_always_init

Enables the `NoAssignX-ML` to not report initialization statements in always blocks.

By default, the value of the parameter is set to no.

Used by	NoAssignX-ML
Options	yes, no
Default value	no

nocheck_with_package

Enables the `ConsCase` tag to ignore the names used in entity body and architecture body against the names used in all the packages in the design.

By default, the value of the parameter is no. In this case, the `ConsCase` tag checks the names used in entity body and architecture body against the names used in all the packages in the design.

Set the value of the parameter to yes to ignore such cases.

Used by	ConsCase
Options	Positive Integer Value
Default value	3

no_scripts

(Optional) Specifies the list of scripts that should not be allowed in the design for Tag-checking with the [NoScripts](#) Tag.

By default, the `no_scripts` Tag parameter is set to `"dc_script_begin dc_script_end"`. To set the `no_scripts` Tag parameter to different scripts, specify the script names as a space-separated list as the value of the `no_scripts` Tag parameter.

Used by	NoScripts
Options	Space-separated list of scripts
Default value	<code>dc_script_begin dc_script_end</code>

num_max_fsm_states

(Optional) Sets the threshold number of states in an FSM for the [STARC05-2.11.1.4](#) Tag.

By default, the `STARC05-2.11.1.4` Tag flags FSMs that have more than 60 states. You can set the `starc2005_num_max_fsm_states` Tag parameter to any positive integer number.

Used by	STARC05-2.11.1.4
Options	Positive integer value
Default value	0

overlappingLoops

For CombLoop:

(Optional) Detects the presence of overlapping loops while checking with the [CombLoop](#) Tag.

By default, the `overlappingLoops` Tag parameter is set to `yes` or `1` and the `CombLoop` Tag flags all the overlapping loops in the design.

To ignore the overlapping loops you can unset the value of the `overlappingLoops` Tag parameter to `no` or `0`.

For STARC05-1.2.3.1 and STARC05-2.4.1.4:

Sets the STARC05-1.2.1.3 and STARC05-2.4.1.4 tags to report all loops or only the first found loop in case of overlapping loops.

By default, the STARC05-1.2.1.3 tag reports all loops for overlapping loops. You can set the `overlappingLoops` Tag parameter to `yes` or `1` (to report all loops) or to `no` or `0` (to report only the first found loop).

For Rule_24 and Rule_28:

By default, the value is set to `yes`. Set the value of the parameter to `no` to ignore the overlapping loops.

For LINT_LOOP_THRU_FLOP_WITHOUT_SETRESET

Specifies whether the `LINT_LOOP_THRU_FLOP_WITHOUT_SETRESET` tag should validate suffix or prefix naming convention in the next state of FSM.

By default, the `overlappingLoops` parameter is set to `yes` and the `LINT_LOOP_THRU_FLOP_WITHOUT_SETRESET` tag detects all of the overlapping combinational loops. Set this parameter to `no` to not report overlapping combinational loops.

Used by	CombLoop , STARC05-1.2.1.3 , STARC05-2.4.1.4 , Rule_24 , Rule_28 , LINT_LOOP_THRU_FLOP_WITHOUT_SETRESET
Options	yes, no
Default value	yes

only_top_gate_list

Enables you to specify whether you can instantiate any other module,

except the special modules, at the top level.

Used by	STARC05-1.2.1.3
Options	yes, no
Default value	yes

overlapsimloops

Specifies whether the [sim_loop01](#) tag reports overlapping simulation loops. By default, the tag does not report overlapping simulation loops. Set this parameter to yes to report overlapping simulation loops.

Used by	sim_loop01
Options	yes, no
Default value	yes

override_module_list

Enables the [ParamValueOverride-ML](#) tag to specify comma-separated list of regular expressions to match with module names.

By default, the value of the `override_module_list` parameter is no.

Set the value of the parameter to yes or comma-separated list of regular expressions to match with module names.

Used by	ParamValueOverride-ML
Options	yes, comma-separated list of regular expressions
Default value	no

paramname

(Optional) Specifies a logical expression made of PERL regular expressions that is used by the [ParamName](#) Tag to check parameter names.

By default, the [ParamName](#) Tag checks the subprogram names using the following logical expression made of PERL regular expressions:

`"/^.{1,8}$/ and not /__/"`

The above logical expression means that parameter names should not be longer than 8 characters (controlled by `/^.{1,8}$/` regular expression) and should not contain consecutive underscores (controlled by `not /__/"` regular expression).

Use the `paramname` Tag parameter to specify a different logical expression for the ParamName Tag.

Used by	ParamName
Options	Perl regular expression pattern for a parameter name
Default value	<code>"/^.{1,8}\$/ and not /__/"</code>

package_header_check_file

Specifies a file, which should contain new PERL subroutine to check the header format.

By default, the `package_header_check_file` parameter is not set.

Set this parameter to the name of the file that contains a PERL subroutine to check the header format. The file name should either be a full path to the file, or a relative path, which can be found in the PERL, include path `@INCPATH`. The name of the PERL subroutine should be **pack_check**.

Used by	PackHdr
Options	A PERL valid filename
Default value	<code>""</code>

processblk_linecount

Sets the maximum number of lines allowed in a process construct for the [STARCO5-2.6.1.4a](#) and [STARCO5-2.6.1.4b](#) tag.

By default, the [STARCO5-2.6.1.4a](#), [STARCO5-2.6.1.4b](#) tags flags process constructs that have more than 200 HDL, comment, and blank lines. You can set the `processblk_linecount` tag parameter to any positive integer number.

Used by	STARC05-2.6.1.4a , STARC05-2.6.1.4b
Options	Positive integer value
Default value	200

portname

(Optional) Specifies a logical expression made of PERL regular expressions that is used by the [PortName](#) Tag to check port names.

By default, the PortName Tag checks the port names using the following logical expression made of PERL regular expressions:

```
"/^[a-z][a-z0-9_]*$/ and /^.{1,16}$/ and not /__/"
```

The above logical expression means that port names should start with a lowercase alphabet and should contain only lowercase alphabets, numbers, and underscores only (controlled by `"/^[a-z][a-z0-9_]*$/"` regular expression), should not be longer than 16 characters (controlled by `"/^.{1,16}$/"` regular expression), and should not contain consecutive underscores (controlled by `"/__/"` regular expression).

Use the `portname` Tag parameter to specify a different logical expression for PortName Tag.

Used by	PortName
Options	Perl regular expression pattern for a port name
Default value	<code>"/^[a-z][a-z0-9_]*\$/ and /^.{1,16}\$/ and not /__/"</code>

portorder

(Optional) Specifies the order that the port declarations should follow for Tag-checking with the [PortOrder_A](#) and [PortOrder_C](#) Tag.

By default, the `portorder` Tag parameter is set to:

```
'in out inout buffer linkage; clocks resets others'
```

The above setting means that these Tags check port declaration order by type to be as follows (controlled by `in out inout buffer linkage part`):

- 1. Input ports
- 2. Output ports
- 3. Inout ports
- 4. Buffer ports (VHDL only)
- 5. Linkage ports (VHDL only)

Also, within each port type, the PortOrder_A Tag checks that the ports are specified by function in the following order (controlled by `clocks resets others` part):

- 1. Clock ports
- 2. Reset ports
- 3. Other port types

You can specify a different order by supplying a different value of the `portorder` Tag parameter as follows:

```
'<port-type-list>;<port-func-list>'
```

Where `<port-type-list>` is a space-separated list of all of the following keywords in any desired order:

in	out	inout	buffer	linkage

`<port-func-list>` is a space-separated list of all of the following keywords in any desired order:

clock	reset	others

NOTE: *The syntax for providing the port order, that is, within quotes or not, depends on your operating environment (shell).*

Used by	PortOrder_A , PortOrder_C , PreDefAttr
Options	Two space-separated lists separated by a semicolon(:)- first list consists of space-separated order w.r.t. the port direction, and the second list consists of space-separated order w.r.t. port function
Default value	'in out inout buffer linkage; clocks resets others'

procname

(Optional) Specifies a logical expression made of PERL regular expression that the [ProcName](#) Tag uses to check procedure names.

By default, the [ProcName](#) Tag checks the procedure names using the following logical expression made of PERL regular expression:

`"/^.*_PROC$/ and /^.{1,16}$/ and not /__/"`

Use the `procname` Tag parameter to specify a different logical expression for the [ProcName](#) Tag.

Used by	ProcName
Options	Perl regular expression pattern for a process name
Default value	<code>"/^.*_PROC\$/ and /^.{1,16}\$/ and not /__/"</code>

print_recommended_regexp

Specifies whether the *SigName*, *VarName*, *ConstName*, and *PortName* Tags should report the recommended regular expression pattern that is used for signal name.

By default, the `print_recommended_regexp` parameter is set to `no`.

If you set this parameter to `yes` or comma-separated Tag list, the violation messages of the Tags include the recommended regular expression pattern that is used.

Used by	<i>SigName</i> , <i>VarName</i> , <i>SigName</i> , <i>PortName</i>
Options	yes, no, comma-separated Tag list
Default value	no

pwrdsnpin

(Optional) Specifies the name of the power down pins as used by the *pwrdsnPinConnToSeqOrIOCells* tag.

By default, the *pwrdsnPinConnToSeqOrIOCells* tag assumes that all power down pins are named PWRDN. Use the `pwrdsnpin` parameter to specify your own power down pin naming convention as a space-separated list of regular expressions enclosed in double quotes.

Used by	<i>pwrdsnPinConnToSeqOrIOCells</i>
Options	Space-separated list of regular expressions
Default value	"PWRDN"

PNDTREE_CELL_LIST

Use this parameter to specify a comma-separated list of PI or PO pins. If a pin is an input pin, it is treated as PI. If a pin is an output pin, it is treated as PO pin.

```
configure_lint_tag_parameter -tag "Rule_9" -parameter
PNDTREE_CELL_LIST -value {CELL_PI_PO_1,A; CELL_PI_PO_2,A,B}
-goal <goal_name>
```


Here, Cell_PI_PO_1 and Cell_PI_PO_2 are libCell and A and B are pins of libCell.

Used by	Rule_9 , Rule_14 , Rule_22
Options	Comma-separated list of PI/PO pins
Default value	""

regoutname

(Optional) Specifies a logical expression made of PERL regular expressions that is used by the [SameDu-ML](#) Tag to check register output names.

By default, the RegOutName Tag checks the register output names using the following logical expression made of PERL regular expressions:

```
"/^(.*)_r$/"
```

The above logical expression means that register output names should end with `_r`.

Use the `regoutame` Tag parameter to specify a different logical expression for the [SameDu-ML](#) Tag.

Used by	SameDu-ML
Options	Perl regular expression pattern for register output name
Default value	"/^(.*)_r\$/"

report_all_actlow

Specifies whether the active low signal containing a constant in `else` part of the `if-else` construct should be reported by the [ActLowName](#) Tag.

By default, the `report_all_actlow` Tag parameter is set to `no`, and the `ActLowName` Tag does not report the active low signal containing a constant.

Set the `report_all_actlow` Tag parameter to `yes` to report active low signals containing a constant.

Used by	ActLowName
Options	yes, no
Default value	no

reportalwayslatch

Specifies whether the [InferLatch](#) Tag reports always_latch.

By default, the value of the `reportalwayslatch` Tag parameter is set to no, and the InferLatch Tag does not report always_latch.

Set the value of the `reportalwayslatch` Tag parameter to yes to report always_latch.

Used by	InferLatch
Options	yes, no
Default value	no

report_flop_clock_loop

Specifies whether the [CombLoop](#) tag should report a violation if loop exists through clock pin and output pin of a flip-flop.

By default, the `report_flop_clock_loop` parameter is set to no and the tag does not report a violation if loop exists through clock pin and output pin of a flip-flop.

Set the value of the `report_flop_reset_loop` tag parameter to yes to report such cases.

Used by	CombLoop
Options	yes, no
Default value	no

report_flop_reset_loop

Enables the specified tag to report violation if loop exists through set/reset pin and output pin of a flip-flop.

By default, the `report_flop_reset_loop` tag parameter is set to `no` and the specified tag does not report violation if loop exists through set/reset pin and output pin of a flip-flop.

Set the value of the `report_flop_reset_loop` tag parameter to `yes` to report such cases.

Used by	CombLoop , STARC05-1.2.1.3 , Rule_24
Options	yes, no
Default value	no

reportlatchedoutput

Specifies whether the [RegOutputs](#) Tag should report violation for latched output ports.

By default, the `reportlatchedoutput` Tag parameter is set to `no` and the `RegOutputs` Tag does not report violations for latched output ports.

Set the value of the `reportlatchedoutput` Tag parameter to `yes` to report latched output ports.

Used by	RegOutputs
Options	yes, no
Default value	no

report_localparam

Specifies whether the [ConstName](#) Tag should report violations for local parameters.

By default, the `report_localparam` parameter is set to `no`, and the `ConstName` Tag does not report violations for local parameters.

Set the `report_localparam` parameter to `yes` to enable the

ConstName Tag to report violation for local parameters, which do not follow the recommended naming convention.

Used by	ConstName
Options	yes, no
Default value	no

reportHangingLatch

Specifies whether the [InferLatch](#) Tag reports violation for hanging latches.

By default, the `reportHangingLatch` parameter is set to `no`, and the [InferLatch](#) Tag does not report hanging latches.

Set the value of the `reportHangingLatch` parameter to `yes` to report violations for such cases.

Used by	InferLatch
Options	yes, no
Default value	no

reportLatchHierarchy

Specifies if the [InferLatch](#) rule reports violation with hierarchical information. By default, the `reportLatchHierarchy` parameter is set to `no`, and [InferLatch](#) rule does not report the hierarchy in the violation message.

Set the value of the parameter to `yes` to report violations with hierarchical information. The violation messages will include the instance hierarchy information, as shown below:

Latch inferred for signal '<sig-name>' in module '<mod-name>' (instance: '<instance-name>')

Used by	InferLatch
Options	yes, no
Default value	no

ignoreLatchHierarchy

Specifies the latch lib cell names and module names for which the InferLatch rule should not report violations.

By default, the parameter is set to `no` and the rule reports for all latch instance in every hierarchy.

You can also specify a list of latch lib cells names and module names (regular expressions are supported) for which the rule should not report violations.

NOTE: *This parameter is valid only if the `reportLatchHierarchy` parameter is set to `yes`.*

Used by	InferLatch
Options	yes, no
Default value	no

report_mux_select

Specifies whether the tags should flag combinational loops existing through the select pin of a MUX.

By default, the `report_mux_select` tag parameter is set to `yes`, and the tags report violations for combinational loops existing through the select pin of a MUX.

You can change the value of the `report_mux_select` Tag parameter to `no`.

Used by	CombLoop , STARC05-1.2.1.3 , Rule_24 , Rule_28
Options	yes, no
Default value	yes

report_nonstatic_actlow

Specifies whether the [ActLowName](#) or [STARC05-1.1.1.7](#) tag reports a violation for an active low reset branch having non-static assignment in the RHS of

an assign statement.

By default, the `report_nonstatic_actlow` parameter is set to `no`, and the respective Tag does not report a violation for an active low reset branch having non-static assignment in the RHS of an assign statement.

Set the value of the `report_nonstatic_actlow` tag parameter to `yes` to report violations for such cases.

Used by	ActLowName , STARC05-1.1.1.7
Options	yes, no
Default value	no

reserve_names

(Optional) Specifies the VHDL reserve keywords that should not be used in a Verilog design for Tag-checking with the [ReserveName](#) Tag.

By default, the `reserve_names` Tag parameter is set to `spy_default_list` list, which contains the standard VHDL keywords.

To add additional reserved words, change the `reserve_names` Tag parameter by adding a space-separated list of additional reserved words.

For example, to add `myres1` and `myres2` reserve names to the default keyword list, set the `reserve_names` Tag parameter to the following value:

```
spy_default_list myres1 myres2
```

Used by	ReserveName
Options	Space-separated list of keywords
Default value	<code>spy_default_list</code>

reserve_names_vh

(Optional) Specifies the Verilog reserve keywords that should not be used in a VHDL design for Tag-checking with the [ReserveName](#) Tag.

By default, the `reserve_names_vh` Tag parameter is set to `spy_default_list` list, which contains the standard Verilog keywords.

To add additional reserved words, change the `reserve_names_vh` Tag parameter by adding a space-separated list of additional reserved words.

For example, to add `myres1` and `myres2` reserve names to the default keyword list, set the `reserve_names_vh` Tag parameter to the following value:

```
spy_default_list myres1 myres2
```

Used by	ReserveName
Options	Space-separated list of keywords
Default value	<code>spy_default_list</code>

resetname

(Optional) Specifies a logical expression made of PERL regular expression that the [ResetName](#) Tag uses to check the reset names.

By default, the `ResetName` Tag checks the reset names using the following logical expression made of PERL regular expression:

```
"/^rst/i"
```

The above logical expression means that reset type signal names should start with `rst`.

Use the `resetname` Tag parameter to specify a different logical expression for the `ResetName` Tag.

Used by	ResetName
Options	Perl regular expression pattern for a reset name
Default value	<code>"/^rst/i"</code>

SigHierName_option

(Optional) Specifies the mode of Tag-checking by the [SigHierName](#) Tag (VHDL).

By default, the `SigHierName_option` Tag parameter is set to `similar` and the `SigHierName` Tag flags those cases where the name of a signal connected to a non-clock pin on an instance does not contain the name of the pin to which it is attached.

When the `SigHierName_option` Tag parameter is set to `same`, the `SigHierName` Tag checks that the connected signal name is exactly same as the pin's name.

Used by	SigHierName
Options	same, similar
Default value	similar

sighiernamelist

(Optional) Specifies the comma-separated list of signal names that should be checked by the [SigHierName](#) Tag.

By default, the [SigHierName](#) Tag checks all non-clock signals.

Used by	SigHierName
Options	Comma-separated list of signals
Default value	" "

signame

(Optional) Specifies a logical expression made of PERL regular expressions that is used by the [SigName](#) Tag to check signal names.

By default, the `SigName` Tag checks the signal names using the following logical expression made of PERL regular expressions:

`"/^[a-z][a-z0-9_]*$/ and /^.{1,32}$/ and not /__/"`

The above logical expression means that signal names should start with a lowercase alphabet and should contain only lowercase alphabets, numbers, and underscores only (controlled by `"/^[a-z][a-z0-9_]*$/` regular expression), should not be longer than 32 characters (controlled by `"/^.{1,32}$/` regular expression), and should not contain consecutive

underscores (controlled by `not /__/` regular expression).

Use the `signame` Tag parameter to specify a different logical expression for the `SigName` Tag.

Used by	SigName
Options	Perl regular expression pattern for a signal name
Default value	<code>"/^[a-z][a-z0-9_]*\$/ and /^{1,32}\$/ and not /__/"</code>

snake_length

(Optional) Specifies the maximum number of design units that a combinational path can span beyond which the [STARC05-1.6.2.2](#) tag will flag a message.

By default, the [STARC05-1.6.2.2](#) tag flags combinational paths that span more than two design units. Set the `snake_length` rule parameter to any positive integer to change the maximum number of design units. Then, SpyGlass starts traversing from the output pin of a flip-flop and flags a message if the path spans more than `$snake_length` design units. All paths irrespective of their endpoints (data pins of flip-flops/latches, primary output ports, or black box terminals) are checked.

Used by	STARC05-1.6.2.2
Options	Positive integer value
Default value	2

snakepath_ff

(Optional) Specifies the type of checking by the [STARC05-1.6.2.2](#) tag.

By default, the `snakepath_ff` rule parameter is not set and the [STARC05-1.6.2.2](#) tag flags those combinational paths that span more design units than the number specified with the `snake_length` rule parameter irrespective of their endpoints (data pins of flip-flops/latches, primary output ports, or black box terminals).

Set the `snakepath_ff` rule parameter to `yes` and the [STARC05-1.6.2.2](#) rule checks only those combinational paths that span more design units

than the number specified with the `snake_length` rule parameter and end at the data pin of a flip-flop or at a primary output port.

Used by	STARC05-1.6.2.2
Options	yes, no
Default value	no

through

Specifies the module names for which snake path needs to be reported by the [STARC05-1.6.2.2](#) rule.

By default, the `through` parameter is not set. Set this parameter to a comma separated list of module names or a PERL regular expression to specify the module names for which snake path needs to be reported.

Used by	STARC05-1.6.2.2
Options	Comma separated module names, PERL regular expression
Default value	" "

topgates

(Optional) Specifies the list of gates whose instantiations at the top-level of the design are to be ignored by the [NoTopGates](#) Tag.

By default, the `topgates` Tag parameter is set to null, that is, no gate names are specified. You can set the value of the `topgates` Tag parameter to a space-separated list of gate names.

Used by	NoTopGates
Options	Space-separated list gates
Default value	" "

triname

Specifies a logical expression made of PERL regular expressions that is

used by the *TriStateName* tag to check tristate signal names.

By default, the TriStateName tag checks the tristate signal names using the following logical expression made of PERL regular expressions:

`"/_z$/"`

The above logical expression means that tristate signal names should end with `_z`.

Use the `triname` parameter to specify a different logical expression for the TriStateName tag.

Used by	<i>TriStateName</i>
Options	Perl regular expression pattern for a tristate name
Default value	<code>"/_z\$/"</code>

typename

(Optional) Specifies a logical expression made of PERL regular expression that the *TypeName* Tag uses to check type names.

By default, the TypeName Tag checks the type names using the following logical expression made of PERL regular expression:

`"/^[A-Z][A-Z0-9_]*$/ and /^.{1,32}$/ and not /__/"`

Use the `typename` Tag parameter to specify a different logical expression for the TypeName Tag.

Used by	<i>TypeName</i>
Options	Perl regular expression pattern for type name
Default value	<code>"/^[A-Z][A-Z0-9_]*\$/ and /^.{1,32}\$/ and not /__/"</code>

uniq_char_val

(Optional) Specifies the number of characters up to which signal, variable, and port names should be unique for Tag-checking with the *Uniq8Char* Tag.

By default, the `uniq_char_val` Tag parameter is set to 8 and the Uniq8Char Tag flags object names that have the same first 8 characters.

You can set the `uniq_char_val` Tag parameter to any positive integer number. Then, the [Uniq8Char](#) Tag flags object names that have the specified number of same first characters.

Used by	Uniq8Char
Options	Positive integer value
Default value	8

valid_cell_list

Specifies the valid cells that can be either cell type or PERL regular expression for cell names.

By default, the `valid_cell_list` parameter is not set.

Set this parameter to a comma separated list of strings to define the valid cells that can be either cell type or PERL regular expression for cell names.

Used by	InitialConnectionCheck
Options	Comma separated list of strings
Default value	""

varname

(Optional) Specifies a logical expression made of PERL regular expressions that is used by the [VarName](#) Tag to check variable names.

By default, the VarName Tag checks the variable names using the following logical expression made of PERL regular expressions:

```
"/^[a-z][a-z0-9_]*$/ and /^.{1,32}$/ and not /__/"
```

The above logical expression means that variable names should start with a lowercase alphabet and should contain only lowercase alphabets, numbers, and underscores only (controlled by `/^[a-z][a-z0-9_]*$/` regular expression), should not be longer than 32 characters (controlled by `/^.{1,32}$/` regular expression), and should not contain consecutive underscores (controlled by `not /__/` regular expression).

Use the varname Tag parameter to specify a different logical expression for the VarName Tag.

Used by	VarName
Options	Perl regular expression pattern for a variable name
Default value	" /^[a-z][a-z0-9_]*\$/ and /^[1,32]\$/ and not /_/"

alwaysblk_optcount

(Optional) Sets the maximum number of outputs allowed in an always construct for the [STARC05-2.6.1.3](#) Tag.

By default, the STARC05-2.6.1.3 Tag flags always constructs that have more than 5 outputs. You can set the alwaysblk_optcount Tag parameter to any positive integer number.

Used by	STARC05-2.6.1.3
Options	Positive integer value
Default value	5

arithrel_op_maxsize

(Optional) Sets the maximum bit-width of operands beyond which the [STARC05-2.10.5.1](#) Tag flags conditional expressions with same or logically-equivalent sub-expressions of arithmetic or relational operators.

By default, the arithrel_op_maxsize Tag parameter is set to 8 and the STARC05-2.10.5.1 Tag flags when one or more operands are of bit-width more than 8-bits. You can set the arithrel_op_maxsize Tag parameter to any positive integer number.

Used by	STARC05-2.10.5.1
Options	Positive integer value
Default value	8

assign_check

Specifies the type of signal assignment statements to be checked by the [STARC05-2.10.3.2b](#), [STARC05-2.10.3.2b_sa](#), and [STARC05-2.10.3.2b_sb](#) Tags.

The valid values of the `assign_check` Tag parameter are as follows:

Value	Checking
0	(Default) all signal assignment statements
1	Simple signal assignment statements only
2	Signal assignment statements using arithmetic operators only
3	Checks for: - the assignment statements using bit-wise operators only (OR, AND, Ex-OR, Ex-NOR and NOT) - the simple assignment statements, that is, the assignments without any operators on the RHS. NOTE: This parameter value (3) is not applicable for the STARC05-2.10.3.2b_sa, and STARC05-2.10.3.2b_sb Tags.
Used by	STARC05-2.10.3.2b , STARC05-2.10.3.2b_sa , STARC05-2.10.3.2b_sb
Options	0, 1, 2, 3
Default value	0

assume_driver_load_srp

Performs the Tag checking for the [STARC05-2.5.1.5a](#) Tag as follows:

- By default, the [STARC05-2.5.1.5a](#) Tag does not report a violation for unloaded signals or variables. Set this parameter to `load`, `both`, or `yes` to report a violation for such cases.
- By default, the [STARC05-2.5.1.5a](#) Tag does not report a violation for hanging driver nets and drivers of net through the gray-box element. Set this parameter to `driver`, `both`, or `yes` to report a violation for such cases.

- By default, the [STARC05-2.5.1.5a](#) Tag does not report a violation for undriven nets connected to the output port of an instance. Set this parameter to `yes` to report a violation for such cases.

Used by	STARC05-2.5.1.5a
Options	yes, no, both, driver, load
Default value	no

bitwidth_basedconst

(Optional) Sets the maximum bit-width beyond which the [STARC05-2.10.3.6](#) Tag reports constant declarations of based number constants that do not have bit-width specification.

By default, the `bitwidth_basedconst` Tag parameter is set to 4 and the [STARC05-2.10.3.6](#) Tag flags constant declarations of based number constants of bit-width more than 4-bits that do not have bit-width specification. You can set the `bitwidth_basedconst` Tag parameter to any positive integer number.

Used by	STARC05-2.10.3.6
Options	Positive integer value
Default value	4

bit_width_reduction

(Optional) Sets the threshold operand bit-width for the [STARC05-2.1.4.6b](#) Tag.

By default, the [STARC05-2.1.4.6b](#) Tag flags reduction operations on operands of bit-width more than 32 bits. You can set the `bit_width_reduction` Tag parameter to any positive integer number.

Used by	STARC05-2.1.4.6b
Options	Positive integer value
Default value	32

check_all_always

Specifies if the [AlwaysComment](#) and [STARC05-3.5.6.2](#) tags should check all always (always_ff, always_comb and always_latch) blocks and report violations.

By default, the check_all_always tag parameter is set to no.

Set the value of the check_all_always parameter to yes to enable the tags to check all always (always_ff, always_comb and always_latch) blocks and report violations if they do not have a corresponding comment in the same or preceding lines.

Used by	AlwaysComment , STARC05-3.5.6.2
Options	yes, no
Default value	no

check_any_bit_for_xz

Used to report violations for the cases where x and z have other bits also. By default, this parameter is set to no. Set this parameter to yes to enable the [STARC05-2.10.1.4a](#) Tag to report violations for the cases where x and z have other bits also.

Used by	STARC05-2.10.1.4a
Options	yes, no
Default value	no

check_case_keyword

Specifies the type of case-statement reported by the [STARC05-2.8.4.3](#) Tag. By default, the value of the check_case_keyword parameter is set to all and the [STARC05-2.8.4.3](#) Tag reports a violation for both casex and casez statements.

You can set the value of the parameter to either casex or casez to report violation only for casex and casez statements, respectively.

Used by	STARC05-2.8.4.3
Options	casex, casez, all
Default value	all

check_clock_cell

Specifies IP blocks to consider clock signals as a non-clock signals by the [STARC05-1.4.3.4](#) Tag.

By default, the value of the `check_clock_cell` parameter is not set.

Set this parameter to a comma separated list of IP blocks for which you want the [STARC05-1.4.3.4](#) Tag to consider clock signals as a non-clock signals. No violation is reported if the clock signals of specified IP blocks are used as non-clock signals in different logic.

This parameter also supports regular expressions.

Used by	STARC05-1.4.3.4
Options	comma separated list of IP blocks
Default value	null

check_dependency

Specifies whether the [STARC05-2.2.3.1](#) Tag should ignore violations for dependent assignment statements.

By default, the value of the `check_dependency` parameter is set to no.

Set this parameter to yes to not report violations for dependent assignment statements.

Used by	STARC05-2.2.3.1
Options	yes, no
Default value	no

checkDriverInDu

Specifies whether the [STARC05-2.5.1.5a](#) Tag considers undriven instance output ports to be drivers for the connected nets.

By default, the `checkDriverInDu` Tag parameter is set to `no` and the [STARC05-2.5.1.5a](#) Tag does not consider undriven instance output ports to be drivers for the connected nets.

You can set the value of the `checkDriverInDu` Tag parameter to `yes` (to enable the mode) or to `no` (to disable the mode).

Used by	STARC05-2.5.1.5a
Options	yes, no
Default value	no

check_pad_concat

Specifies whether the [STARC05-2.1.3.1](#), [STARC05-3.2.3.2](#), [STARC05-2.10.3.2b_sa](#), and [STARC05-2.10.3.2b_sb](#) Tags to consider leading zeros in concatenation/multiconcatenation operator while width calculation.

By default, the value of the parameter is set to `no` and the Tags mentioned above ignore leading zeros in `concat/multiconcat` operator while width calculation.

Set the value of the parameter to `yes` or `<Tag-name>` to consider leading zeros in concatenation/multiconcatenation operator while width calculation.

Used by	STARC05-2.1.3.1 , STARC05-3.2.3.2 , STARC05-2.10.3.2b_sa , STARC05-2.10.3.2b_sb
Options	yes, no, <Tag-name>
Default value	no

checkFirstInstance

Forces the [STARC05-1.1.2.6a](#) and [STARC05-1.1.2.6b](#) Tags to the following:

- Ignore I/O ports that are directly connected to the instance as in the following example:

```

module top(in1, out1);
    input in1;
    output out1;

    m2 m2_u(.a(in1), .b(out1));
endmodule

module my(a, b);
    input a;
    output b;
    ...
endmodule

```

- Flag only one message for each module even if the module is instantiated multiple times in the design

Used by	STARC05-1.1.2.6a , STARC05-1.1.2.6b
Options	yes, no
Default value	no

check_index

Specifies whether the [STARC05-3.2.2.1](#) Tag reports violations for non-parameterized constants used as array indexes.

By default, the *check_index* parameter is set to `no`. This indicates that the Tag does not report violations for non-parameterized constants used as array indexes. Set this parameter to `yes` to report violations for such cases.

Used by	STARC05-3.2.2.1
Options	yes, no
Default value	no

check_internally_generated_clk

Specifies whether the [STARC05-1.4.3.1b](#) Tag check for the internally

generated clock.

By default, the `check_internally_generated_clk` parameter is set to `no` and the Tag does not check for internally generated clock nets. Set this parameter to `yes` to enable the Tag to consider internally generated nets also during the Tag checking.

Used by	STARC05-1.4.3.1b
Options	yes, no
Default value	no

check_var_in_condition

Specifies whether the [STARC05-2.3.2.4](#) tag reports violations for the following cases:

- Variables used inside `if-else`, `case`, or `while` loop conditions
- Variables used as a loop index of the `for` loop
- Variables used as an array index of another variable

By default, the `check_var_in_condition` parameter is set to `yes`. This indicates the Tag does not report violations for the above-mentioned cases. Set this parameter to `no` to report violations for such cases.

Used by	STARC05-2.3.2.4
Options	yes, no
Default value	yes

clkGeneratorName

Specifies a comma-separated list of clock generator design unit or cell names as checked by the [STARC05-1.4.1.1](#) Tag.

The [STARC05-1.4.1.1](#) tag does not flag instances generated inside the clock generator design unit or cell names specified by this Tag parameter.

Used by	STARC05-1.4.1.1
Options	A comma-separated list of clock generator design unit or cell names
Default value	""

clk_EnableLatch

(Optional) Specifies that the [STARC05-1.4.3.4](#) Tag should ignore flip-flop clock signals that are also used as latch enable signals.

By default, the STARC05-1.4.3.4 Tag flags flip-flop clock signals that are being used as non-clock signals (including as latch enable signals) in other descriptions.

Used by	STARC05-1.4.3.4
Options	yes, no
Default value	no

clk_gen_du

Use this parameter to specify the list of clock-generator-modules for the [LINT_CLKGEN_AVOID_SEQSTMT](#) tag.

Also, supports regular expressions.

Used by	LINT_CLKGEN_AVOID_SEQSTMT
Options	comma-separated list of clock-generator-module names
Default value	" "

comb_logic_style

(Optional) Sets the preferred modeling style for the [STARC05-2.1.1.1](#) Tag.

By default, the STARC05-2.1.1.1 flags always constructs used for describing

combinational logic. You can set the value of `comb_logic_style` Tag parameter to `always` (for `always` constructs to be used for describing combinational logic) or to `function` (for `function` constructs with `assign` statements to be used for describing combinational logic).

Used by	STARC05-2.1.1.1
Options	<code>function, always</code>
Default value	<code>function</code>

const_string

Specifies the constant name search patterns as checked by the [STARC05-1.1.4.2v](#) Tag.

By default, the `const_string` parameter is set to `/^C_/` and `/^[A-Z0-9_]+$/` and the [STARC05-1.1.4.2v](#) Tag reports the constant names where:

- The name does not start with `C_`.
- All characters of the name are not in uppercase.
- The name contains characters other than alphanumeric characters or underscore.

NOTE: *This parameter supports the ignore case feature.*

Used by	STARC05-1.1.4.2v
Options	Perl regular expression
Default value	<code>/^C_/</code> and <code> /^[A-Z0-9_]+\$/</code>

fixedinputenableflop

Causes the [STARC05-2.3.5.1](#) Tag to report enabled flip-flops only when both enable pin and data pin are connected to supply or enable pin is open and data pin is connected to supply.

By default, the [STARC05-2.3.5.1](#) Tag flags enabled flip-flops only when the data pin is held to a constant value.

Used by	STARC05-2.3.5.1
Options	yes, no
Default value	no

flag_only_induction_var

(Optional) Specifies the type of non-constants to be checked by the [STARC05-2.9.2.1](#), [STARC05-2.9.2.2](#), and [STARC05-2.9.2.3](#) Tags.

By default, the `flag_only_induction_var` Tag parameter is set to `no` and the corresponding Tags flag all non-constants used in arithmetic, logical, or relational operations.

When the `flag_only_induction_var` Tag parameter is set to `yes`, the corresponding Tags flag only those non-constants that are induction variables.

Used by	STARC05-2.9.2.1 , STARC05-2.9.2.2 , STARC05-2.9.2.3
Options	yes, no
Default value	no

functask_match_global_var

Specifies whether the [STARC05-1.1.1.5](#) Tag should violation if outer scope variable is used inside a function or task.

By default, the `functask_match_global_var` parameter is set to `yes` and the [STARC05-1.1.1.5](#) Tag reports violation if outer scope variable is used inside a function or task.

Set this parameter to `no` to not report such violations.

Used by	STARC05-1.1.1.5
Options	yes, no
Default value	yes

handle_latch_setreset

Specifies whether the [STARC05-1.3.1.3](#) Tag also check for latches.

By default this parameter is set to no and the [STARC05-1.3.1.3](#) Tag considers the flip-flop's set/reset only as a start point. It reports a violation if the flip-flop's set/reset is going to another latch's set/reset pin.

If you set to this parameter to yes, the [STARC05-1.3.1.3](#) Tag does not give violation starting from the flip-flop's set/reset and reaching to another latch's set/reset. The Tag also honors latch's set/reset as a start point in this case.

Used by	STARC05-1.3.1.3
Options	yes, no
Default value	no

handle_negation_semantics

Specifies whether the [STARC05-2.10.2.3](#) Tag also should report/ignore violations for logical/bitwise negations.

By default this parameter is set to no.

If set to "yes", the [STARC05-2.10.2.3](#) Tag does not report violations in following scenarios.

- Logical negation of a vector used in if condition - if(!vec)
- Logical negation of a vector assigned to a bit - bit_a = !vec_b

The [STARC05-2.10.2.3](#) Tag reports additional violations in following scenarios:

- Bitwise negation used on a vector in a if condition - if(~vec)
- Bitwise negation used on a vector and assigned to a non vector - bit_a = ~vec_b

Used by	STARC05-2.10.2.3
Options	yes, no
Default value	no

ignore_based_width

Specifies whether the [STARC05-2.10.3.7](#) Tag reports violations for hexal, octal, and decimal based numbers, if the width specified is greater than the actual bit-width of the based number.

By default, the value of this parameter is set to `no` and the [STARC05-2.10.3.7](#) Tag reports a violation for binary, hexal, octal, and decimal based numbers, if the width specified is greater than the actual bit-width of the based number.

You can set the value of the parameter to `yes` to disable violations reported by the [STARC05-2.10.3.7](#) for such cases.

Used by	STARC05-2.10.3.7
Options	<code>no</code> , <code>yes</code>
Default value	<code>no</code>

ignore_cell

Enables the [STARC05-1.4.3.4](#) and [STARC05-1.3.1.3](#) Tags to ignore IP blocks while finding clock or reset signal used as non-clock or non-reset signal in a design, respectively.

By default, the value of the `ignore_cell` parameter is not set and the Tags do not ignore such cases.

Set this parameter to a comma separated list of IP blocks that you want to ignore while finding clock ([STARC05-1.4.3.4](#)) or reset ([STARC05-1.3.1.3](#)) signal used as non-clock or non-reset signal respectively in the design. When this parameter is set, the Tag checks in fan-out of ignored IP blocks for a clock or reset signal used as a non-clock or non-reset signal, respectively.

This parameter also supports regular expressions.

Used by	STARC05-1.4.3.4 , STARC05-1.3.1.3
Options	comma separated list of IP blocks
Default value	<code>null</code>

ignore_direct_connection

Specifies whether the [STARCO5-1.6.2.1](#) Tag reports a violation when an output port of a module is connected to its input directly or through buffers.

By default, the value of the `ignore_direct_connection` parameter is set to `no` and the `STARCO5-1.6.2.1` Tag reports a violation, if the output port of a module is connected to its input directly or through buffers. You can set the value of the parameter to `yes` to disable violations flagged by the `STARCO5-1.6.2.1` for such cases.

Used by	STARCO5-1.6.2.1
Options	yes, no
Default value	no

ignoreElseIf

Use this parameter to ignore tag checking for if statements in sequential circuits that do not have corresponding else statements.

Used by	STARCO5-2.7.1.3b
Options	no, yes
Default value	no

ignore_empty_lines

Specifies whether the [STARCO5-3.5.6.2](#), [AlwaysComment](#), [PortComment](#), [SignalComment](#), and [VariableComment](#) tags should ignore the number of empty lines as specified in the value.

By default, the `ignore_empty_lines` tag parameter is set to `-1` and the tags ignore any number of white spaces/blank lines before the construct to check the comment.

You can set the value of `ignore_empty_lines` parameter to `0` or any positive integer to restrict the number of blank lines before the construct to match the comment. That is, by setting the value to `0` or any positive

integer, the tags ignore that number of empty lines as specified in the value.

Used by	AlwaysComment , PortComment , SignalComment , Variable-Comment , STARC05-3.5.6.2
Options	yes, no
Default value	no

ignore_file_with_multiple_modules

Specifies whether the [STARC05-1.1.1.1](#) Tag should ignore violations for the files containing multiple modules.

By default, the `ignore_file_with_multiple_modules` is not set and the [STARC05-1.1.1.1](#) Tag reports violations for files containing multiple modules.

Set this parameter to `yes` to not report violations for such cases.

For example, if a file contains multiple modules and the file name does not match with name of any module, you can ignore violations for this by setting the `ignore_file_with_multiple_modules` parameter to `yes`.

Used by	STARC05-1.1.1.1
Options	yes, no
Default value	no

ignore_fsm_counter

Specifies whether the [STARC05-2.11.3.1](#) Tag reports a violation for sequential block infer counter logic.

By default, the value of the `ignore_fsm_counter` parameter is set to `no` and the [STARC05-2.11.3.1](#) Tag reports a violation for sequential block infer counter logic.

Set the value of this parameter to `yes` to ignore violations for such cases.

This parameter is only applicable for Verilog.

Used by	STARC05-2.11.3.1
Options	no, yes
Default value	no

ignore_macro

Specifies whether the [STARC05-1.1.1.2](#) Tag should ignore the specified macro.

By default, the value of the `ignore_macro` parameter is not set.

If a violated name is resulting from an expansion of a macro, whose macro name matches with any of regular expression patterns specified through this parameter, then the Tag does not report violation.

Used by	STARC05-1.1.1.2
Options	Comma separated PERL regular expression pattern
Default value	""

ignore_one_hot

Specifies whether the [SpareCaseExprBit-ML](#) tag should check for redundant bits in case expression.

By default, the value of the parameter is set to no.

Set the parameter to `yes` to enable the [SpareCaseExprBit-ML](#) tag to not report violations for case blocks with one-hot case items.

Used by	SpareCaseExprBit-ML
Options	yes, no
Default value	no

ignore_one_hot_labels

Specifies whether the [STARC05-2.11.4.2](#) Tag should process on-hot labels.

By default, the value of the `ignore_one_hot_labels` parameter is not

VC SpyGlass Lint Parameters

set.

Set the value of this parameter to `yes` if you do not want to process one-hot labels for violations.

Used by	STARC05-2.11.4.2
Options	no, yes
Default value	no

ignore_pad_cell

Specifies whether the [STARC05-1.3.2.2](#) Tag should ignore the violations when reset is passing through a pad cell.

By default, the value of the ignore_pad_cell parameter is not set.

Set the value of this parameter to yes, to ignore violations when reset is passing through a pad cell.

Used by	STARC05-1.3.2.2
Options	no, yes
Default value	no

ignore_parameter_in_port

Specifies whether the [STARC05-1.1.4.8](#) Tag should ignore the parameters that are used only in signal/variable range either at the time of declaration or usage as bit-select of part-select.

By default, the value of the ignore_parameter_in_port parameter is set to no.

Set the value of this parameter to yes to ignore the parameters that are used only in signal/variable range either at the time of declaration or used as bit-select of part-select.

Used by	STARC05-1.1.4.8
Options	no, yes
Default value	no

ignore_sv_keywords

Specifies whether the [STARC05-1.1.1.3](#) Tag should ignore the usages of SystemVerilog keywords or their case variants when SystemVerilog is not enabled.

By default, the value of the ignore_sv_keywords parameter is set to no.

Set the value of this parameter to *yes*, to ignore the usages of SystemVerilog keywords or their case variants when SystemVerilog is not enabled.

Used by	STARC05-1.1.1.3
Options	no, yes
Default value	no

ignore_vector_bit_selection

Specifies whether the [STARC05-2.10.3.6](#) Tag ignores violations for bit selects, part selects, array bit selects and array part selects.

By default, the `ignore_vector_bit_selection` parameter is set to *no* and the Tag checks bit selects, part selects, array bit selects and array part selects. Set the value of this parameter to *yes* to ignore for such cases.

Used by	STARC05-2.10.3.6
Options	no, yes
Default value	no

lib_sync_cells

Specify the list of cell names. This parameter is used by [RegOutputs](#) tag. By default this parameter is not included for checking. Set the value of the parameter to a comma-separated list of cells to ignore violations for the output ports, which are going through lib cells specified through this parameter.

Used by	RegOutputs
Options	<comma-separated list of cells>
Default value	NULL

line_length_max

(Optional) Specifies the maximum permissible length for an RTL code line to be checked by the [STARC05-3.1.4.5](#) Tag.

By default, the STARC05-3.1.4.5 Tag flags RTL code lines that are longer than 110 characters. You can set the `line_length_max` Tag parameter to any positive integer number.

Used by	STARC05-3.1.4.5
Options	Positive integer value
Default value	110

maxLoopIter

(Optional) Specifies the number of iterations in `for` constructs, where the non-constants are involved in a logical or relational operation. This parameter is used by [STARC05-2.9.2.3](#) Tag.

By default, the `maxLoopIter` Tag parameter is set to value 10. You can change the default value of `maxLoopIter` Tag parameter to any positive integer value.

Used by	STARC05-2.9.2.3
Options	Positive integer value
Default value	10

nesting_level

Specifies the maximum nesting level in conditional operator (`?:`) expressions as checked by the [STARC05-2.1.5.1](#) tag.

By default, the `nesting_level` tag parameter is set to 0 and the STARC05-2.1.5.1 tag does not allow any nesting in the conditional operator(`?:`) expressions.

You can set the `nesting_level` tag parameter to any positive integer number. Then, the STARC05-2.1.5.1 tag flags only those conditional

operator (?:) expressions where the nesting level is more than the specified number.

Used by	STARC05-2.1.5.1
Options	Positive integer value
Default value	0

num_max_bit_changes

(Optional) Sets the maximum allowed number of bit changes during state transitions for the [STARC05-2.11.1.2](#) Tag.

By default, the [STARC05-2.11.1.2](#) Tag flags case constructs where the number of bits changed between states is more than one. You can set the num_max_bit_changes Tag parameter to any positive integer number.

Used by	STARC05-2.11.1.2
Options	Positive integer value
Default value	1

numerator_size

Use this parameter to an integer value to enable the "[DivideByConstant-ML](#)" tag to report violations only for dividers whose numerator size (width, according to the LRM) is greater than or equal to the cutoff size specified using this parameter.

The default value of this parameter is 0. You can set the value of the parameter to an integer value.

Used by	DivideByConstant-ML
Options	Positive integer value
Default value	0

param_string

Specifies the parameter name search patterns as checked by the [STARC05-1.1.4.2a](#) Tag.

By default, the `param_string` parameter is set to `/^P_/` and the [STARC05-1.1.4.2a](#) Tag reports all parameter names that do not start with `P_`.

NOTE: *This parameter supports the ignore case feature.*

Used by	STARC05-1.1.4.2a
Options	Perl regular expression
Default value	" / [^] P_/"

processblk_opcount

(Optional) Sets the maximum allowed number of output signals, ports, or variables are being assigned in a `process` construct for the [STARC05-2.6.1.3](#) Tag.

By default, the [STARC05-2.6.1.3](#) Tag flags `process` constructs that have more than 5 outputs. You can set the `processblk_opcount` Tag parameter to any positive integer number.

Used by	STARC05-2.6.1.3
Options	Positive integer value
Default value	5

report_allclk

Specifies whether the [STARC05-1.4.3.1a](#) and [STARC05-1.4.3.4](#) Tag reports all violations for a clock source.

By default, the [STARC05-1.4.3.1a](#) and [STARC05-1.4.3.4](#) Tags report only one violation for a clock source.

If you set the value of the `report_allclk` parameter to `yes`, the supported Tags report all violations for a clock source.

Used by	STARC05-1.4.3.1a , STARC05-1.4.3.4
Options	yes, no, comma separated list of Tags
Default value	no

report_all_datatype

Specifies whether the [STARC05-3.2.4.1](#) Tag reports violations for all the data types used in a generic declaration except integer.

By default, the [STARC05-3.2.4.1](#) Tag reports violations for non-integers, such as `time` and `real`, if used in generic declaration.

If you set the value of the `report_all_datatype` parameter to `yes`, the Tag reports violations for all data types used in a generic declaration, except integer.

Used by	STARC05-3.2.4.1
Options	yes, no
Default value	no

report_instance_pin

Specifies if instance pins of a netlist design should appear in the violation message.

Allowed Values of the report_instance_pin Parameter

The allowed values of this parameter are described in the following table:

Value	Description
yes	Specify this value to enable the tag to report instance-pin names.
no (default value)	Specify this value to report the output-net name of an instance instead of the instance-pin name.

Value	Description
bbox	Specify this value so that the tag reports the black-box pin of the source and destination of a crossing of an RTL or a netlist design.
seqCell	Specify this value so that the tag reports the library sequential-cell pin of the source and destination of a crossing of an RTL or a netlist design.
flop	Specify this value so that the tag reports the flip-flop pin of the source and destination of a crossing for a netlist design. This value has no impact on RTL designs.
latch	Specify this value so that the tag reports the latch pin of the source and destination of a crossing for a netlist design. This value has no impact on RTL designs.

Used by	Reset_check11
Options	yes, no, bbox, seqCell, flop, latch, all
Default value	no

reportallnets

Specifies whether the [UndrivenNUnloaded-ML](#) tag should report a violation on all the nets.

The default value is no.

Used by	UndrivenNUnloaded-ML

Options	yes, no
Default value	no

report_all_rst

Specifies whether the [STARC05-1.3.1.3](#) Tag reports all occurrences of non-reset/preset signals or synchronous reset/preset signals of asynchronous reset/preset signals in the RTL.

By default, the [STARC05-1.3.1.3](#) Tag reports only the first violation it encounters.

Set the value of the `report_all_rst` parameter to `yes`, to report all such violations.

Used by	STARC05-1.3.1.3
Options	yes, no
Default value	no

report_full_path

Specifies whether the [STARC05-1.6.2.2](#) rule should report the full complete snake path.

By default, the `report_full_path` parameter is set to `no`.

Set the value of this parameter to `yes` to enable the rule to report the complete snake path.

Used by	STARC05-1.6.2.2
Options	yes, no
Default value	no

report_reset_type

Specifies whether the [STARC05-1.3.2.2](#) Tag should report asynchronous, synchronous or both resets.

By default, the `report_reset_type` parameter is set to `async`, and the STARC05-1.3.2.2 Tag reports asynchronous resets only.

You can set this parameter to `async`, `sync`, or `all`. The behavior of STARC05-1.3.2.2 Tag for these parameter values is as follows:

- **sync**: The Tag reports sync resets.
- **all**: The Tag reports violation for both asynchronous and synchronous resets.
- **async**: The Tag reports for asynchronous resets only.

Used by	STARC05-1.3.2.2
Options	<code>async</code> , <code>sync</code> , <code>all</code>
Default value	<code>async</code>

report_typedef

Specifies whether the [STARC05-1.1.1.2](#) Tag should report violations on typedefs.

By default, the `report_typedef` parameter is set to `no` and the [STARC05-1.1.1.2](#) reports violations for typedef elements, however it does not report any violations for typedef itself.

If you set the `report_typedef` parameter to `yes`, the Tag also reports violations on typedefs. The Tag reports the unused typedefs as well.

Used by	STARC05-1.1.1.2
Options	<code>yes</code> , <code>no</code>
Default value	<code>no</code>

report_violation_type

Specifies if the [W416](#) tag should report a violation for different type of width mismatches (truncation, extension).

By default, the parameter is set to `both` and the [W416](#) tag reports a

violation for any width mismatch.

Set the value of the parameter to `truncation` to report a violation when the function return value width is more than function return type width.

Set the value of the parameter to `extension` to report a violation when the function return value width is less than function return type width.

Used by	W416
Options	truncation, extension, both
Default value	both

reset_synchronizer_modname

Specifies whether the [STARC05-1.3.1.3](#) Tag ignores flip-flops.

Set the value of the `reset_synchronizer_modname` parameter to a comma-separated list of module names. The specified modules are considered as reset synchronizers, and flip-flops falling down the hierarchy of these modules are ignored during the Tag checking.

Consider this situation:

If reset pin of flip-flop A hits at the `d` pin of flip-flop B and flip-flop B is part of a module, which is specified with the `reset_synchronizer_modname` parameter.

In this case no violation for reset pin of flip-flop A is reported.

Used by	STARC05-1.3.1.3
Options	Comma-separated list of hierarchical module names
Default value	sgdummy1, sgdummy2

rtl_acc_max_length

Specifies the maximum value for the accumulated RTL length in a line where both RTL and comments are present. Enables the [LineLength](#) tag to report a violation if the RTL length is greater than the specified maximum RTL length.

Used by	LineLength
Options	positive integer value
Default value	" "

src_master_dir

(Optional) Sets the master directory names, which can be a comma separated list of source directory names, for the [STARCO5-3.5.2.1](#) Tag (Verilog).

If master directory argument is not specified, the STARCO5-3.5.2.1 Tag only checks for relative file path names.

Used by	STARCO5-3.5.2.1
Options	Comma-separated list of source directory names
Default value	" "

src_master_dir_vh

(Optional) Sets the master directory, which can be a comma separated list of source directory names for the [STARCO5-3.5.2.1v](#) Tag (VHDL).

If master directory argument is not specified, the STARCO5-3.5.2.1v Tag only checks for relative file path names.

Used by	STARCO5-3.5.2.1v
Options	Comma-separated list of source directory names
Default value	" "

srlatch

(Optional) Sets the [STARCO5-1.2.1.3](#) Tag to ignore SR-Latch type combinational loops. By default, the STARCO5-1.2.1.3 Tag reports SR-Latch type combinational loops.

You can set the `srlatch` Tag parameter to `yes` or `1` (to ignore SR-Latch

type combinational loops) or to `no` or 0 (to report SR-Latch type combinational loops).

Used by	STARC05-1.2.1.3
Options	yes, no
Default value	no

srp_gen_1316_report

Specifies whether the [STARC05-1.3.1.6](#) Tag should generate a report for all the synchronous and asynchronous usage of reset nets.

By default, the value of the `srp_gen_1316_report` Tag parameter is set to `no`, and the STARC05-1.3.1.6 Tag does not generate any report for reset nets.

Used by	STARC05-1.3.1.6
Options	yes, no
Default value	no

sr_max

(Optional) Specifies the maximum number of shift-register bits for the [ShiftReg](#) tag.

By default, the ShiftReg tag flags shift operations on signals or part-selects of signals of more than 20 bits. Use the `sr_max` parameter to specify a different integer number.

Used by	ShiftReg
Options	Positive integer value
Default value	20

starc_file_ext_vhdl

Checks for the valid extension for the VHDL design source files.

You can specify a comma-separated list of strings, such as, ".vhd,.vhdl,.vhld". In this case, the [STARCO5-1.1.1.1](#) Tag checks for file names with the extension ".vhd", ".vhdl", or ".vhld".

If no file name with the specified extension is found, the STARCO5-1.1.1.1 Tag flags a violation.

Used by	STARCO5-1.1.1.1
Options	.vhd, .vhdl, .vhld
Default value	.vhd, .vhdl

starc_file_ext_vlog

Checks for the valid extension for the Verilog design source files.

You can specify a comma-separated list of strings, such as, ".v, .sv, .bv". In this case, the [STARCO5-1.1.1.1](#) Tag checks for file names with the extension ".v", ".sv", or ".bv".

If no file name with the specified extension is found, the STARCO5-1.1.1.1 Tag flags a violation.

Used by	STARCO5-1.1.1.1
Options	.v, .sv, .bv
Default value	.v, .sv

starc_inst_name

Specifies the instance name patterns as checked by the [STARCO5-1.1.1.8v](#) Tag.

By default, the `starc_inst_name` Tag parameter is set to `U`, and the [STARCO5-1.1.1.8v](#) Tag checks for instance names as `U_<entity-name>`.

You can set the `starc_inst_name` Tag parameter to any standard regular expression as described in [Using PERL Regular Expressions](#).

Used By	STARC05-1.1.1.8v
Options	Perl regular expression
Default Value	U

starc_through_block

Filters the violations in the rule [STARC05-1.6.2.2](#).

By default, the `starc_through_block` parameter is not set.

You can set this parameter to a comma-separated list of module names or PERL regular expressions to enable the rule to report violations only for the paths going through specified modules.

If multiple module names are specified, the rule reports violations only for the paths going through all the specified modules.

Used by	STARC05-1.6.2.2
Options	Comma-separated list modules, PERL regular expressions
Default value	""

starc2005_always_line1

Specifies the moderate limit on the size of always construct and is used by the [STARC05-2.6.1.4a](#) (Verilog) Tag.

By default, the `starc2005_always_line1` Tag parameter is set to value 300. You can set the `starc2005_always_line1` Tag parameter to any integer value.

Used by	STARC05-2.6.1.4a
Options	Integer value
Default value	300

starc2005_always_line2

Specifies the upper limit on the size of always construct as used by the

[STARC05-2.6.1.4a](#) (Verilog), [STARC05-2.6.1.4b](#) (Verilog) Tags.

By default, the `starc2005_always_line2` Tag parameter is set to value 2000. You can set the `starc2005_always_line2` Tag parameter to a different integer value.

Used by	STARC05-2.6.1.4a , STARC05-2.6.1.4b
Options	Integer value
Default value	2000

starc2005_arc_name

Specifies the character string to be included in the architecture names as used by [STARC05-1.1.6.1](#) Tag.

By default, the `starc2005_arc_name` is set to the string RTL.

Used by	STARC05-1.1.6.1
Options	Character string
Default value	RTL

starc_check_sequential_block

Specifies the missing defaults for sequential blocks for [STARC05-2.8.1.4](#) Tag.

By default, the `starc_check_sequential_block` Tag parameter is set to no, and the [STARC05-2.8.1.4](#) Tag does not check for sequential blocks.

Set the `starc_check_sequential_block` Tag parameter to yes to check for sequential blocks. For example, the [STARC05-2.8.1.4](#) Tag reports violation in the following case when this Tag parameter is set to yes:

```
always @(posedge clk_stbus or negedge rst_stbus_n)
always_ff /always_latch
```

Used by	STARC05-2.8.1.4
Options	yes, no

starc2005_compare_str

Specifies the object name search patterns for the [STARC05-1.1.1.4](#) (Verilog and VHDL) Tag.

NOTE: *Tag checking is case-insensitive for VHDL.*

By default, the `starc2005_compare_str` Tag parameter is set to `VDD, VSS, GND, VCC, VREF` and the `STARC05-1.1.1.4` Tag flags object names starting with `VDD, VSS, GND, VCC, VREF`.

You can set the `starc2005_compare_str` Tag parameter to any desired search string list.

The general format of the `starc2005_compare_str` Tag parameter value is as follows:

`"str1, str2, str3, ...."`

Where, `str1, str2, str3`, and so on are optional strings.

If you set the value of this parameter to `"str1", "str2", "str3"`, the [STARC05-1.1.1.4](#) Tag reports violation for object names, which start with strings `"str1", "str2", and "str3"`.

For example, if the value of the `starc2005_compare_str` parameter is set to `VSS, VDD, VREF`, the `STARC05-1.1.1.4` Tag will report violation for the object name starting with either `VSS, VDD, or VREF`.

Used by	STARC05-1.1.1.4
Options	Object name search pattern
Default value	<code>VDD, VSS, GND, VCC, VREF</code>

starc2005_define_max

Specifies the upper limit on the length of the macro define name as used by the [STARC05-1.1.3.3e](#) Tag.

By default, the `starc2005_define_max` Tag parameter is set to value 40. You can change the value of the `starc2005_define_max` Tag parameter to any positive integer.

Used by	STARC05-1.1.3.3e
Options	Positive integer value
Default value	40

starc2005_define_min

Specifies the lower limit on the length of the macro define name as used by the [STARC05-1.1.3.3e](#) Tag.

By default, the `starc2005_define_min` Tag parameter is set to value 2. You can change the value of the `starc2005_define_min` Tag parameter to any positive integer.

Used by	STARC05-1.1.3.3e
Options	Positive integer value
Default value	2

starc2005_define_string

(Optional) Specifies a regular expression, which should be used to name define macros as used by the [STARC05-1.1.4.2b](#) Tag.

By default, the `starc2005_define_string` parameter is set to `/^D_[a-zA-Z0-9]*[a-zA-Z0-9]$/`.

NOTE: *This parameter supports the ignore case feature.*

Used by	STARC05-1.1.4.2b
Options	Perl regular expression
Default value	<code>/^D_[a-zA-Z0-9]*[a-zA-Z0-9]\$/</code>

starc2005_define_viol

Specifies maximum number of violations per file for Tag [STARC05-3.2.2.4](#) Tag.

By default, the `starc2005_define_viol` Tag parameter is set to 1.

You can set the `starc2005_define_viol` Tag parameter to any positive integer value. When the value for this parameter is set to zero, STARC05-3.2.2.2b Tag reports all violations.

Used by	STARC05-3.2.2.4
Options	Positive integer value
Default value	1

starc2005_ext_string_vhdl

Specifies the package file naming convention as checked by the [STARC05-1.1.4.1va](#) Tag.

By default, the STARC05-1.1.4.1va Tag flags those package files whose names do not end with `_pac` (with `.vhd` or `.vhdl` as extensions).

You can set the `starc2005_ext_string_vhdl` Tag parameter to a comma-separated list of suffixes/extensions as in the following example:

```
configure_lint_tag_parameter -tag STARC05-1.1.4.1va -
parameter starc2005_ext_string_vhdl -value
"_pack\.vhd,_pack\.vhdl"
```

Please note that the dot (.) character needs to be escaped. That is, you need to use backslash (\) before a dot (.), if the exact character (.) is to be matched; otherwise, (.) will match any character.

Used by	STARC05-1.1.4.1va
Options	Comma-separated list of strings
Default value	"_pac\.vhd,_pac\.vhdl"

starc2005_ext_string_vlog

Specifies the 'include file extensions as checked by the [STARC05-1.1.4.1](#) Tag.

By default, the STARC05-1.1.4.1 Tag flags included files that do not have `.h`, `.vh`, or `.inc` extensions.

You can set the `starc2005_ext_string_vlog` Tag parameter to a comma-separated list of extensions as in the following example:

```
configure_lint_tag_parameter -tag STARC05-1.1.4.1va -  
parameter starc2005_ext_string_vlog -value "\.myh,\.mytsk"
```

Please note that the dot (.) character needs to be escaped. That is, to match an exact character (.) , you need to give \. (escape the dot character); otherwise, (.) character will match any character.

For example, to include extension .abc , you need to give \.abc as value of the `starc2005_ext_string_vlog` Tag parameter. If you give .abc, then the STARC05-1.1.4.1 Tag will not report any violation for file extension with .abc but will also not report violations for any file extensions ending in abc, such as .tabc.

Used by	STARC05-1.1.4.1
Options	Comma-separated list of strings
Default value	"\h,\vh,\inc"

starc2005_function_max

Specifies the upper limit on the length of the function name as used by the [STARC05-1.1.3.3d](#) Tag.

By default, the `starc2005_function_max` Tag parameter is set to 40. You can change the default value of the `starc2005_function_max` Tag parameter to any positive integer value.

Used by	STARC05-1.1.3.3d
Options	Positive integer value
Default value	40

starc2005_function_min

Specifies the lower limit on the length of the function name as used by the [STARC05-1.1.3.3d](#) Tag.

By default, the `starc2005_function_min` Tag parameter is set to 2. You can change the default value of the `starc2005_function_min` Tag parameter to any positive integer value.

Used by	STARC05-1.1.3.3d
Options	Positive integer value
Default value	2

starc2005_gate_maxcount

(Optional) Sets the maximum number of gates allowed in a basic block for the [STARC05-1.1.3.3d](#) Tag. Default value is 40000.

By default, the `starc2005_gate_maxcount` Tag parameter is set to 40000. You can set the `starc2005_gate_maxcount` Tag parameter to any positive integer number.

Used by	STARC05-1.1.3.3d
Options	Positive integer value
Default value	40000

starc2005_ifelse_branchlevel_reco

Specifies the maximum allowed `if..else` construct branching as checked by the [STARC05-2.7.3.1b](#) Tag.

By default, the `starc2005_ifelse_branchlevel_reco` Tag parameter is set to 7 and the `STARC05-2.7.3.1b` Tag flags `if..else` constructs with more than seven branches that test the same signal in all branches.

You can set the `starc2005_ifelse_branchlevel_reco` Tag parameter to any positive integer number.

Used by	STARC05-2.7.3.1b
Options	Positive integer value
Default value	7

starc2005_ifelse_chainlevel_reco

Specifies the maximum allowed if..else construct chaining as checked by the [STARC05-2.7.3.1c](#) Tag.

By default, the `starc2005_ifelse_chainlevel_reco` Tag parameter is set to 7 and the [STARC05-2.7.3.1c](#) Tag flags if..else constructs that are more than seven levels deep.

For Verilog designs, the depth of a if..else.if construct is assumed to be the number of if statements. For VHDL designs, the depth of a if..elsif construct is assumed to be the number of if or elsif statements in the construct.

You can set the `starc2005_ifelse_chainlevel_reco` Tag parameter to any positive integer number.

Used by	STARC05-2.7.3.1c
Options	Positive integer value
Default value	7

starc2005_ifelse_nestlevel_reco

Specifies the maximum allowed nesting of if..else constructs as checked by the [STARC05-2.7.3.1a](#) Tag.

By default, the `starc2005_ifelse_nestlevel_reco` Tag parameter is set to 7 and the [STARC05-2.7.3.1a](#) Tag flags those nested if..else constructs that are more than seven levels deep.

You can set the `starc2005_ifelse_nestlevel_reco` Tag parameter to any positive integer number.

Used by	STARC05-2.7.3.1a
Options	Positive integer value
Default value	7

starc2005_maxCaseClause

Specifies the maximum allowed number of case branches for case

constructs as checked by the [STARC05-2.8.2.1](#) Tag.

The STARC05-2.8.2.1 Tag flags the following:

1. case constructs where case construct condition bit-width exceeds the value of the [starc2005_maxInputSize](#) Tag parameter.
2. case constructs that do not violate the above conditions but have (((case construct condition bit-width > value of the [starc2005_maxCondBitWidth](#) Tag parameter) OR (number of case clauses > value of the [starc2005_maxCaseClause](#) Tag parameter)) AND (case construct output bit-width > value of the [starc2005_maxOutBitWidth](#) Tag parameter)).

The default value of the `starc2005_maxCaseClause` Tag parameter is 5 and you can set it to any positive integer number.

Used by	STARC05-2.8.2.1
Options	Positive integer value
Default value	5

starc2005_maxCondBitWidth

Specifies the maximum allowed condition bit-width for case constructs as checked by the [STARC05-2.8.2.1](#) Tag.

The STARC05-2.8.2.1 Tag flags the following:

1. case constructs where case construct condition bit-width exceeds the value of the [starc2005_maxInputSize](#) Tag parameter.
2. case constructs that do not violate the above conditions but have (((case construct condition bit-width > value of the `starc2005_maxCondBitWidth` Tag parameter) OR (number of case clauses > value of the [starc2005_maxCaseClause](#) Tag parameter)) AND (case construct output bit-width > value of the [starc2005_maxOutBitWidth](#) Tag parameter)).

The default value of the `starc2005_maxCondBitWidth` Tag parameter is 4 and you can set it to any positive integer number.

Used by	STARC05-2.8.2.1

Options	Positive integer value
Default value	4

starc2005_maxInputSize

Specifies the maximum allowed condition size for case constructs as checked by the [STARC05-2.8.2.1](#) Tag.

The STARC05-2.8.2.1 Tag flags the following:

1. case constructs where case construct condition bit-width exceeds the value of the `starc2005_maxInputSize` Tag parameter.
2. case constructs that do not violate the above conditions but have (((case construct condition bit-width > value of the [starc2005_maxCondBitWidth](#) Tag parameter) OR (number of case clauses > value of the [starc2005_maxCaseClause](#) Tag parameter)) AND (case construct output bit-width > value of the [starc2005_maxOutBitWidth](#) Tag parameter)).

The default value of the `starc2005_maxInputSize` Tag parameter is 20 and you can set it to any positive integer number.

Used by	STARC05-2.8.2.1
Options	Positive integer value
Default value	20

starc2005_maxOutBitWidth

Specifies the maximum allowed output bit-width for case constructs as checked by the [STARC05-2.8.2.1](#) Tag.

The STARC05-2.8.2.1 Tag flags the following:

1. case constructs where case construct condition bit-width exceeds the value of the [starc2005_maxInputSize](#) Tag parameter.
2. case constructs that do not violate the above conditions but have (((case construct condition bit-width > value of the [starc2005_maxCondBitWidth](#) Tag parameter) OR (number of case clauses > value of the [starc2005_maxCaseClause](#) Tag parameter)) AND (case construct output bit-width > value of the `starc2005_maxOutBitWidth` Tag parameter)).

The default value of the `starc2005_maxOutBitWidth` Tag parameter is 12 and you can set it to any positive integer number.

Used by	STARC05-2.8.2.1
Options	Positive integer value
Default value	12

starc2005_maxTableSize

Specifies the maximum allowed number of branches for case constructs as checked by the [STARC05-2.8.2.2](#) Tag.

By default, the STARC05-2.8.2.2 Tag flags case constructs with more than 200 branches.

You can set the `starc2005_maxTableSize` Tag parameter to any positive integer number.

Used by	STARC05-2.8.2.2
Options	Positive integer value
Default value	200

starc2005_max_inst_length

Specifies the maximum name length for instances as checked by the [STARC05-1.1.2.1b](#) Tag.

By default, the STARC05-1.1.2.1b Tags flags instance names longer than 32 characters.

You can set the `starc2005_max_inst_length` Tag parameter to any positive integer number.

Used by	STARC05-1.1.2.1b
Options	Positive integer value
Default value	32

starc2005_max_mod_ent_char_length

Specifies the upper limit on the length of the top-level module/entity name as used by the [STARC05-1.1.1.9b](#) Tag.

By default, the `starc2005_max_mod_ent_char_length` Tag parameter is set to 16 and the `STARC05-1.1.1.9b` Tag flags top-level module/entity names when the length of the top-level module/entity name is greater than the value specified by the `starc2005_max_mod_ent_char_length` Tag parameter.

You can set the `starc2005_max_mod_ent_char_length` Tag parameter to any positive integer value.

Used by	STARC05-1.1.1.9b
Options	Positive integer value
Default value	16

starc2005_max_mod_ent_length

Specifies the maximum name length for design units (modules or entities) as checked by the [STARC05-1.1.2.1a](#) Tag.

By default, the `STARC05-1.1.2.1a` Tags flags design unit names longer than 32 characters.

You can set the `starc2005_max_mod_ent_length` Tag parameter to any positive integer number.

Used by	STARC05-1.1.2.1a
Options	Positive integer value
Default value	32

starc2005_max_port_char_length

Specifies the upper limit on the length of the top-level port name as used by the [STARC05-1.1.1.9d](#) Tag.

By default, the `starc2005_max_port_char_length` Tag parameter

is set to 16. You can set the `starc2005_max_port_char_length` Tag parameter to a different positive integer value.

Used by	STARC05-1.1.1.9d
Options	Positive integer value
Default value	16

starc2005_min_inst_length

Specifies the minimum name length for instances as checked by the [STARC05-1.1.2.1b](#) Tag.

By default, the STARC05-1.1.2.1b Tags flags instance names shorter than 2 characters.

You can set the `starc2005_min_inst_length` Tag parameter to any positive integer number.

Used by	STARC05-1.1.2.1b
Options	Positive integer value
Default value	2

starc2005_min_mod_ent_length

Specifies the minimum name length for design units (modules or entities) as checked by the [STARC05-1.1.2.1a](#) Tag.

By default, the STARC05-1.1.2.1a Tags flags design unit names shorter than 2 characters.

You can set the `starc2005_min_mod_ent_length` Tag parameter to any positive integer number.

Used by	STARC05-1.1.2.1a
Options	Positive integer value
Default value	2

starc2005_negative_str

Specifies the negative logic object name search patterns for the [STARC05-1.1.1.7](#) Tag.

By default, the `starc2005_negative_str` Tag parameter is set to `_X` and `_N` and the `STARC05-1.1.1.7` Tag flags negative logic object names that do not end with `_X` or `_N`.

You can set the `starc2005_negative_str` Tag parameter to a comma-separated list of strings. Then, the `STARC05-1.1.1.7` Tag flags those negative logic object names that do not end with any of the specified strings.

For example, the following specification indicates that the `STARC05-1.1.1.7` Tag should flag any negative logic object name that does not end with `_X` or `_n`:

```
configure_lint_tag_parameter -tag STARC05-1.1.1.7 -parameter
starc2005_negative_str -value "_X,_n"
```

Thus, the names like `rst_y` or `rst` are to be flagged but not `rst_X` or `rst_n`.

Used by	STARC05-1.1.1.7
Options	Comma-separated list of strings
Default value	"_X,_N"

starc2005_number_for_indent

Specifies the allowed number of spaces at the first character in a line inside the `process/always` constructs, if statements, and case statements as used by the [STARC05-3.1.4.2](#) Tag.

If the value of the `starc2005_number_for_indent` Tag parameter is `n`, then the allowed number of spaces are 0, `n`, `2n`, and so on.

By default, the `starc2005_number_for_indent` Tag parameter is set to two. You can set the `starc2005_number_for_indent` Tag parameter to a different positive integer value.

Used by	STARC05-3.1.4.2
Options	Positive integer value
Default value	2

starc2005_num_max_fsm_states

(Optional) Sets the threshold number of states in an FSM for the [STARC05-2.11.1.4](#) Tag.

By default, the STARC05-2.11.1.4 Tag flags FSMs that have more than 60 states. You can set the `starc2005_num_max_fsm_states` Tag parameter to any positive integer number.

Used by	STARC05-2.11.1.4
Options	Positive integer value
Default value	60

starc2005_parameter_max

Specifies the upper limit on the length of the parameter name as used by the [STARC05-1.1.3.3c](#) Tag.

By default, the `starc2005_parameter_max` Tag parameter is set to value 40. You can change the value of `starc2005_parameter_max` Tag parameter to a any positive integer value.

Used by	STARC05-1.1.3.3c
Options	Positive integer value
Default value	40

starc2005_parameter_min

Specifies the lower limit on the length of the parameter name as used by the [STARC05-1.1.3.3c](#) Tag.

By default, the `starc2005_parameter_min` Tag parameter is set to

value 2. You can change the value of `starc2005_parameter_min` Tag parameter to a any positive integer value.

Used by	STARC05-1.1.3.3c
Options	Positive integer value
Default value	2

starc2005_port_max

Specifies the upper limit on the length of the port name as used by the [STARC05-1.1.3.3b](#) Tag.

By default, the `starc2005_port_max` Tag parameter is set to value 40. You can set the `starc2005_port_max` Tag parameter to a different positive integer value.

Used by	STARC05-1.1.3.3b
Options	Positive integer value
Default value	40

starc2005_port_min

Specifies the lower limit on the length of the port name as used by the [STARC05-1.1.3.3b](#) Tag.

By default, the `starc2005_port_min` Tag parameter is set to value 2. You can set the `starc2005_port_min` Tag parameter to a different positive integer value.

Used by	STARC05-1.1.3.3b
Options	Positive integer value
Default value	2

starc2005_port_maxcount

(Optional) Sets the maximum number of ports allowed in a basic block for

the [STARC05-1.1.3.3b](#) Tag.

By default, the `starc2005_port_maxcount` Tag parameter is set to 500. You can set the `starc2005_port_maxcount` Tag parameter to any positive integer number.

Used by	STARC05-1.1.3.3b
Options	Positive integer value
Default value	500

starc_report_casez

Use this parameter to report violation for the casez constructs. By default, the value of the parameter is set to no. Set the value of the parameter to yes to report violation for the casez constructs.

Used by	STARC05-2.8.2.1
Options	yes, no
Default value	no

priority_set_reset

Allows the [STARC05-1.3.2.1a](#) Tag to report either gated set or reset or both.

By default, the value of the `priority_set_reset` parameter is set to `bothsetreset` and the [STARC05-1.3.2.1a](#) Tag reports both the gated set and reset. You can set the value of the `priority_set_reset` parameter to either `set` or `reset`.

Used by	STARC05-1.3.2.1a
Options	<code>bothsetreset</code> , <code>reset</code> , <code>set</code>
Default value	<code>bothsetreset</code>

starc2005_process_line1

Specifies the moderate limit on the size of process construct as used by the [STARC05-2.6.1.4a](#) (VHDL) Tag.

By default, the `starc2005_process_line1` Tag parameter is set to value 300. You can set the `starc2005_process_line1` Tag parameter to a different integer value.

Used by	STARC05-2.6.1.4a
Options	Positive integer value
Default value	300

starc2005_process_line2

Specifies the upper limit on the size of process construct as used by the [STARC05-2.6.1.4a](#) (VHDL), [STARC05-2.6.1.4b](#) (VHDL) Tags.

By default, the `starc2005_process_line2` Tag parameter is set to value 2000. You can set the `starc2005_process_line2` Tag parameter to a different integer value.

Used by	STARC05-2.6.1.4a , STARC05-2.6.1.4b
Options	Positive integer value
Default value	2000

starc2005_reset_string

Specifies the reset signal name search patterns as checked by the [STARC05-1.1.5.2b](#) Tag.

By default, the `starc2005_reset_string` parameter is set to `/RESET_X/i` or `/RST_X/i` and the [STARC05-1.1.5.2b](#) Tag reports all reset signal names that do not contain `RESET_X` or `RST_X`.

NOTE: *This parameter supports the ignore case feature.*

Used by	STARC05-1.1.5.2b

Options	Perl regular expression
Default value	"/RESET_X/i or /RST_X/i"

starc2005_signal_max

Specifies the upper limit on the length of the signal name as used by the [STARC05-1.1.3.3a](#) Tag.

By default, the `sytarc2005_signal_max` Tag parameter is set to value 40. You can change the value of `starc2005_signal_max` Tag parameter to a any positive integer value.

Used by	STARC05-1.1.3.3a
Options	Positive integer value
Default value	40

starc2005_signal_min

Specifies the lower limit on the length of the signal name as used by the [STARC05-1.1.3.3a](#) Tag.

By default, the `sytarc2005_signal_min` Tag parameter is set to value 2. You can change the value of `starc2005_signal_min` Tag parameter to a any positive integer value.

Used by	STARC05-1.1.3.3a
Options	Positive integer value
Default value	2

starc2005_size_of_mult_max

(Optional) Sets the threshold operand bit-width for the [STARC05-2.10.6.5](#) Tag.

By default, the Tag reports multipliers with an operand of bit-width more than 16 bits. You can set the `starc2005_size_of_mult_max` parameter to any positive integer number.

Used by	STARC05-2.10.6.5
Options	Positive integer value
Default value	16

starc2005_top_mod_ent_name

(Optional) Specifies a regular expression, which is used to name top-level modules as used by the [STARC05-1.1.1.9a](#) Tag.

The default value of the *starc2005_top_mod_ent_name* parameter is `/^[A-Z_0-9]+$ /` or `/^[a-z_0-9]+$ /`.

NOTE: *This parameter supports the ignore case feature.*

Used by	STARC05-1.1.1.9a
Options	Perl regular expression
Default value	<code>/^[A-Z_0-9]+\$ /</code> or <code>/^[a-z_0-9]+\$ /</code>

starc2005_top_port_name

Specifies a regular expression, which should be used to name top-level ports as used by the [STARC05-1.1.1.9c](#) Tag.

By default, the *starc2005_top_port_name* parameter takes the following value:

`/^[A-Z_0-9]+$ /` or `/^[a-z_0-9]+$ /`

NOTE: *This parameter supports the ignore case feature.*

Used by	STARC05-1.1.1.9c
Options	Perl regular expression
Default value	<code>/^[A-Z_0-9]+\$ /</code> or <code>/^[a-z_0-9]+\$ /</code>

starc_use_natural_width

Specifies if the [STARC05-2.8.2.1](#) tag should consider static width of constants and parameters.

By default, the `starc_use_natural_width` parameter is set to `no`.
Set this parameter to `yes` to consider static width of constants and parameters.

Used by	STARC05-2.8.2.1
Options	yes, no
Default value	no

strict_constsig

(Optional) Specifies that the ConstSig tag should also report direct assignments to constants (0 or 1)

By default, the `strict_constsig` tag parameter is set to 0 and the ConstSig does not report direct assignments to constants, as shown in the following example:

```
assign a = 1'b0;
```

Set the `strict_constsig` tag parameter to 1 to report such assignments.

Used by	ConstSig
Options	yes, no
Default value	no

suggest_always_also

Enables the [STARC05-2.1.1.1](#) Tag to also suggest `always_comb` and `always@(*)` also in violation message.

By default, the `suggest_always_also` parameter is set to `no` and the Tag suggests only `function+assign` for function style in the violation message.

Set this parameter to `yes` to enable the Tag to suggest `always_comb` `always@(*)` also for function style in the violation message in addition to the default suggestion of `function+assign`.

Used by	STARC05-2.1.1.1
Options	yes, no
Default value	no

tag_arguments

Specifies if the [Indent](#) tag can change its configuration as in the PERL deck. By default, the tag_arguments parameter is not set.

The tag_arguments parameter can specify the statement categories to be checked and the corresponding indentation levels in the parameter using the following syntax:

```
"[checkgivencat {<cat-id> <level-num>}}][ignore {<cat-id>}] "
```

NOTE: For more information on the configuration, see the *Indent* rule description.

Used by	Indent
Options	yes, no
Default value	no

top_gate_list

(Optional) Sets the modules to be described at the top-layer of the design for the [STARC05-1.6.1.4](#) Tag.

By default, the STARC05-1.6.1.4 Tag flags the modules named `clockgenmodule` and `resetgenmodule` if these modules are not in the top-level layer of the design. You can set the `top_gate_list` Tag parameter to a comma-separated list of design unit names.

Used by	STARC05-1.6.1.4
Options	Comma-separated list of gates
Default value	"clockgenmodule, resetgenmodule"

TRANSMISSION_GATE_CELL_LIST

Use this parameter to specify names of transmission gate cells.
The default value of the parameter is ' '.

Used by	Rule_8
Options	Comma-separated list of transmission gates
Default value	" "

traverse_through_mux

Enables the [STARC05-1.4.3.4](#) tag to traverse through multiplexers in the clock path and detect clock sources.

By default, the traverse_through_mux parameter is set to no.

Set this parameter to yes to enable the tag to traverse through multiplexers in the clock path and detect clock sources.

Used by	STARC05-1.4.3.4
Options	yes, no
Default value	no

tristate_count

(Optional) Sets the threshold number of drivers for tristate bus checking by the [STARC05-2.5.1.4](#) Tag.

By default, the STARC05-2.5.1.4 Tag flags those tristate buses that are driven by more than 5 drivers. You can set the tristate_count Tag parameter to any positive integer number.

Used by	STARC05-2.5.1.4
Options	Positive integer value
Default value	5

typedef_regex

Specifies the accepted typedef names for the [InvalidTypedefName-ML](#) tag.

By default, the value of the typedef_regex tag parameter is set to '.*_t'.

Set the value of this parameter to a comma-separated list of PERL regular expressions specifying accepted typedef names.

Used by	InvalidTypedefName-ML
Options	List of comma-separated PERL regular expressions of accepted typedef names
Default value	.*_t"

unsized_param_cutoff

Reports violations if the parameter is overridden by an unsized based number, where the width of the unsized based number is greater than and equal to the value given in the parameter.

Used by	UseParamWidthInOverriding-ML
Options	non-negative integer
Default value	32

user_defined_resets

Specifies the names of the reset signals to the [SyncRstFirstUse-ML](#) tag.

By default, the value of the *user_defined_resets* tag parameter is set to no.

Set the value of this parameter to a comma-separated list of synchronous reset signal names or regular expressions to specify the synchronous reset signals for the SyncRstFirstUse-ML tag.

Used by	SyncRstFirstUse-ML
Options	Comma-separated list of synchronous reset names or regular expressions
Default value	" "

use_inferred_resets

By default, the `use_inferred_resets` parameter is not set.

Set the parameter to `yes` to specify that the other reset tags should use the auto-generated reset information in addition to any user-defined resets (specified using the reset constraint in a `.sgdc` file).

Used by	Reset_check11
Options	Comma-separated list of synchronous reset names or regular expressions
Default value	" "

use_new_message

Specifies whether the [STARC05-1.1.1.1](#) Tag should report the updated message.

By default, the `use_new_message` parameter is set to `no`. Set this parameter to `yes` to enable the Tag to report the updated message that includes "multiple independent modules/entities."

Used by	STARC05-1.1.1.1
Options	yes, no
Default value	no

use_secondary_message

Specifies whether the [STARC05-2.1.4.5](#) tag should report the updated message.

By default, the `use_secondary_message` parameter is set to `no` and the tag uses the old message format to report violations.

Set this parameter to `yes` to enable the tag to report the updated message that includes *'Avoid using logical operator '<Logical_Operator>' in expression '<Expression>'* to report violations.

Used by	STARC05-2.1.4.5
Options	yes, no
Default value	no

valid_conn_types

Enables you to specify valid type mismatches.

Used by	PortTypeMismatch-ML
Options	yes, no
Default value	no

violate_on_mixed_mode

By default, the `violate_on_mixed_mode` parameter is set to `no`. Set the value of the parameter to `yes` to enable the [STARC05-2.1.1.1](#) tag to report violations only when function as well as assign statement constructs are there in the design. The tag reports violations for both type of constructs.

Used by	STARC05-2.1.1.1
Options	yes, no
Default value	no

viol_wrapper_module

Specifies whether the [UnifiedNetList-ML](#) tag should report violation for wrapper modules, that is, modules containing only one lib cell or only one black box.

By default, the value of this parameter is set to `no`, and the [UnifiedNetList-ML](#) tag does not report violation for wrapper modules. Set the value of this parameter to `yes` to report such cases.

Used by	UnifiedNetList-ML

Options	no, yes
Default value	no

whenelse_cnt

(Optional) Sets the threshold number of when-else levels allowed in conditional signal assignment constructs for the [STARCO5-2.1.5.1v](#) Tag.

By default, the STARCO5-2.1.5.1v Tag flags conditional signal assignment constructs with more than 5 levels of when-else levels. You can set the `whenelse_cnt` Tag parameter to any positive integer number.

Used by	STARCO5-2.1.5.1v
Options	Positive integer value
Default value	5

matchtype

Specifies a matching format for the [RESERVE_NAME](#) Tag to check the occurrence of VDD, VSS, VCC, or VREF in the names of modules, instances, nets, or ports.

By default, the `matchtype` Tag parameter is set to `exact` and the [RESERVE_NAME](#) Tag compares the exact names of modules, instances, nets, or ports for the occurrence of VDD, VSS, VCC, or VREF.

If the `matchtype` Tag parameter is set to `forward`, the [RESERVE_NAME](#) Tag checks for the occurrence of VDD, VSS, VCC, or VREF at the beginning of the names of modules, instances, nets, or ports.

If the `matchtype` Tag parameter is set to `backward`, the [RESERVE_NAME](#) Tag checks for the occurrence of VDD, VSS, VCC, or VREF at the end of the names of modules, instances, nets, or ports.

Used by	RESERVE_NAME
Options	exact, forward, backward
Default value	exact

latch_clock

Specifies a list of primary clock port names for the [ConstDrivenNet-ML](#) and [W422L](#) Tags.

Use the `latch_clock` Tag parameter as follows:

```
configure_lint_tag_parameter -tag ConstDrivenNet-ML -
parameter latch_clock -value <clk-name-list>
```

Used by	ConstDrivenNet-ML , W422L
Options	Name of the clock pin for latches in the design
Default value	Empty string

latch_verbose

(Optional) Specifies [W392aL](#) and [W392bL](#) tags to generate detailed messages.

Use the `latch_verbose` parameter as follows:

```
configure_lint_tag_parameter -tag W392aL -parameter
latch_verbose -value on
```

or

```
configure_lint_tag_parameter -tag W392bL -parameter
latch_verbose -value off
```

Used by	W392aL and W392bL
Options	on, off, comma-separated list of Tags
Default value	off

latch_clock_file

(Optional) Specifies the location where the detailed messages file that is generated by the [ConstDrivenNet-ML](#) Tag when the [latch_verbose](#) Tag parameter is set.

By default, these detailed messages are saved to a file named `latch_clock_edges` in the current directory. Use the `latch_clock_file` Tag parameter to specify a different file.

Use the `latch_clock_file` Tag parameter as follows:

```
configure_lint_tag_parameter -tag ConstDrivenNet-ML -
parameter latch_clock_file -value <file-name>
```

Used by	ConstDrivenNet-ML
Options	Name of a file in which the detailed messages are saved
Default value	<code>latch_clock_edges</code>

latch_reset_file

(Optional) Specifies the location where the detailed messages file that is generated by the [LatchReset](#), [W392aL](#), and [W392bL](#) tags when the [latch_verbose](#) parameter is set.

By default, these detailed messages are saved to a file named `latch_reset` in the current directory. Use the `latch_reset_file` Tag parameter to specify a different file.

Use the `latch_reset_file` Tag parameter as follows:

```
configure_lint_tag_parameter -tag LatchReset -parameter
latch_reset_file -value <file-name>
```

Used by	LatchReset , W392aL , W392bL
Options	Name of a file in which the detailed messaged are saved
Default value	<code>latch_reset</code>

overlappingLatchLoops

Enables detection of overlapping loops.

By default, the value of this parameter is set to `yes` and the [LatchFeedback](#)

Tag detects overlapping loops.

Set the value of this parameter to `no` to disable detection of overlapping loops.

Violations of overlapping loops are reported, even if part of the overlapping loop does not contain a latch.

Used by	LatchFeedback
Options	yes, no
Default value	yes

allow_combo

Specifies whether the [Rule_27](#) tag should consider combinational elements while performing the fan-in traversal of input signals of cells.

By default, the `allow_combo` parameter is set to `no` and the [Rule_27](#) tag ignores combinational elements while performing fan-in traversal of input signals of cells.

When this parameter is set to `yes`, the tag does not ignore combinational elements while performing the fan-in traversal of input signals of cells.

Used by	Rule_27
Options	yes, no
Default value	no

check_depth

Specifies if the [Rule_16](#) tag should limit the back-tracing depth while traversing through the fan-in of the set/reset pins, for checking the convergence.

By default, this parameter is set to `-1` and the [Rule_16](#) tag back-traces until top.

If the value of parameter `check_depth` is 0, the tag reports a violation when set and reset pins of a flip-flop are converging through the same source pin/net, which can be present in any module boundary and no logic

exists between the source and set/reset pins.

Used by	Rule_16
Options	-1, 0
Default value	-1

delay_cell_list

Specifies whether the [Rule_20](#) tag should report a violation when the name of a cell in the fanin cone matches the given name.

By default, the `delay_cell_list` rule parameter is set to "" and the tag reports a violation when cells having the `sec_cell_type: hold_buffer` attribute value are used in the fanin cone of the clock/enable pin of the sequential element/latch.

Set the value of the `delay_cell_list` rule parameter to a comma-separated list of PERL regular expressions to report a violation when the name of a cell in the fanin cone matches the given name.

Used by	Rule_20
Options	comma-separated list of PERL regular expressions
Default value	""

ignore_digital_pad

Specifies whether the [Rule_19](#) tag should stop propagation at digital pad.

By default, the `ignore_digital_pad` rule parameter is set to no and the tag does not stop propagation at digital pad.

Used by	Rule_19
Options	<floating point value>
Default value	0.0

ignoreRtlBuffer

Specifies whether the SpyGlass generated buffers should be ignored by the tags.

By default, the `ignoreRtlBuffer` parameter is set to `no`, and the RTL buffers are not ignored (since they are considered as loads or drivers).

Set the `ignoreRtlBuffer` parameter to `yes`, and all the RTL buffers in the complete fan-in/fan-out cone of the net will be ignored along the path. The traversal will stop as soon as a known RTL cell buffer is found.

Used by	Rule_34 , Rule_35 , UndrivenInTerm-ML , UnloadedNet-ML , UndrivenNet-ML , UnloadedOutTerm-ML , UndrivenNUnloaded-ML , UndrivenOutPort-ML , UnloadedInPort-ML
Options	yes, no
Default value	no (Rule_34 , Rule_35) yes (UndrivenInTerm-ML , UnloadedNet-ML , UndrivenNet-ML , UnloadedOutTerm-ML , UndrivenNUnloaded-ML , UndrivenOutPort-ML , UnloadedInPort-ML)

margin_value

Use this parameter to specify the permissible difference in the voltage values of source and destination pins.

The tag will not report violations if the difference of voltage values of source and destination pins are within the range of this parameter value.

Used by	Rule_5
Options	<floating point value>
Default value	0.0

spare_cell_name

Specifies the naming convention pattern for the spare cells.

The default value of this parameter is `spare_.*`.

Set the value of the parameter to a PERL regular expression pattern to specify the spare cell name to be used by the [Rule_23](#), [Rule_26](#), [Rule_36](#)

rules.

Used by	Rule_23 , Rule_26 , Rule_36
Options	PERL regular expression pattern
Default value	spare_.*
Example	

ENABLE_CHECKING_ON_ICGC

This parameter enables the [LatchDataConstant](#), [LatchEnableConstant](#) to report violations when the Enable pin or Clock pin of a Clock Gate cell is constant.

The default value of this parameter is `no` and the specified tags do not perform any rule checking on the Data and Enable Pin of a Clock Gating Cell (ICGC).

Set the value of this parameter to `yes` for enabling the rule checking for Data and Enable Pin of an ICGC.

Used by	LatchDataConstant , LatchEnableConstant
Options	yes, no
Default value	no

VC SpyGlass Lint Application Variables

VC Static consists of application variables (app vars) that allow you to set application variable to the specific value.

For more information on VC Static application variables, see *VC Static Command Reference Guide*.

However, VC SpyGlass Lint provides `enable_lint` app var, which is used to enable SpyGlass compatibility mode for Tag names, waivers and reporting.

To use this app var, specify the following command in the top Tcl file or in the "vc_static_shell" prior to "sg_read_project" command:

```
set_app_var enable_lint true
```

Product Reports

The VC SpyGlass Lint product generates the following reports:

Report Name	Description
Moresimple	The moresimple report displays the name and alias of the violating Tag.
Count	The count report lists the number of times VC SpyGlass Lint found each type of message
Waiver	A waiver report is generated when messages are waived during VC SpyGlass Lint run.
Default	This is the default report generated by the VC Static product.

Rule_24_Report

The **Rule_24_Report.rpt** report is generated by the [Rule_24](#) tag. This report contain the details of all the combinational loops detected by the [Rule_24](#) tag.

A sample format of Rule_24_Report.rpt is shown below:

```
=====Rule_24 Message report =====
SNO      FILE      LINE      Message
=====
1        loop1.v    23        loop/w1-loop/w2-loop/w3-loop/w4
```

You can also access the Rule_24_Report report at the consolidated_reports directory.

Rule_28_Report

The **Rule_28_Report.rpt** report is generated by the [Rule_28](#) tag. This report contain the details of all the combinational loops detected by the [Rule_28](#) tag.

A sample format of Rule_28_Report.rpt is shown below:

=====Rule_28 Message report =====			
SNO	FILE	LINE	Message
=====			
1	loop1.v	23	loop/w1-loop/w2-loop/w3-loop/w4

You can also access the Rule_28_Report report at the consolidated_reports directory.

all_register_info Report

The `all_register_info` report is generated by the [RegisterInfo-ML](#) tag. The report displays all register information in the RTL design, displays total counts ((Sync Total, Async Total, None Total, Flop/Latch count), and contains both flops and latches with their reset type (async reset / sync reset / none reset).

The following shows a sample format for the `all_register_info.rpt` report:

```

Sync Total   : 1
              Flop   : 1
              Latch   : 0
              Async Total : 3
              Flop    : 1
              Latch    : 2
None Total   : 2
              Flop    : 1
              Latch    : 1
    
```

glitch_circuit

The glitch_circuit.rpt report is generated by the [GlitchGenCircuitCheck](#) tag. This report contains information on the violation messages generated by the tag.

The following is a sample report:

```
The Circuit is constituted by the Following. Glitch generation circuit exists in the design.
```

```
ERROR: Driving Pin:top.I_BUF_w (top, [ OUT ]) Loaded Pin: top.I1 (MAXCELL_AND4, [ IN1, IN2, IN3, IN4 ]) Connected to EXTERNAL  
PORT top.out
```

SpyGlass Design Constraints

For more details of using the SpyGlass Design Constraints, see the *SpyGlass Constraints Rules Reference Guide*.

assume_path

Purpose

Clock domain propagation through black boxes is normally limited as the exact relationship between the input pins and output pins is not known. If only one clock signal is propagated to a black box instance, it is assumed that all pins of the black box instance are in the domain of that clock signal and further clock domain propagation is performed. However, if more than one clock signal propagates to the instance, it is not possible to assume any clock domain information.

One way to extend the clock domain propagation through a black box instance is to specify which output pins belong to the same clock domain as a particular input pin.

The `assume_path` constraint is used by the *InActiveReset-ML* tag. You can specify that paths exist between input pins and output pins of black boxes and thus, clock domain propagation can traverse through these black boxes. These paths can be specified using the `assume_path` keyword in a SpyGlass Design Constraints file.

Syntax

The syntax of the `assume_path` constraint is as follows:

```
assume_path -name <bbdu-name>  
            -input <input-pin-name>  
            -output <output-pin-name-list>
```

Arguments

The `assume_path` constraint has the following arguments:

<bbdu-name>

The black box module name (for Verilog designs) or the black box design unit name in *<entity-name>.<arch-name>* format (for VHDL designs).

<input-pin-name>

The name of an input pin of the black box design unit *<bbdu-name>*.

<output-pin-name>

The list of output pin names of the black box design unit *<bbdu-name>*.

Consider the following example:

```
assume_path -name BBOX -input d -output q qbar
```

This constraint directive indicates that paths exist between input pin *d* and output pins *q* and *qbar* of black box design unit BBOX.

set_case_analysis

SpyGlass also considers the restrictions imposed by case analysis.

Under this feature, if any combinational gate is found in the path between the clock signal and the flip-flop clock pin, it is assumed that the clock signal would reach the flip-flop depending upon the other inputs to these gates. If the clock signal is found to be blocked due to the scalar values specified for other inputs at these gates using the *set_case_analysis* keyword in a SpyGlass Design Constraints file, SpyGlass stops further processing of that clock path.

SpyGlass considers the scalar values specified as follows in a SpyGlass Design Constraints file:

```
current_design <du-name>
  set_case_analysis -name <name> -value 0 | 1
```

Where *<du-name>* is the module name (for Verilog designs) or design unit name in *<entity-name>.<arch-name>* format (for VHDL designs) in the design and *<name>* is a primary port name or an internal pin's hierarchical name.

The following example shows that the `set_case_analysis` constraint has been defined on internal net `top.U1.U13.tm1` and `top.U3.tm3` with value 0:

```
current_design top
  set_case_analysis
    -name top.U1.U13.tm1, top.U3.tm3 -value 0
```

For primary ports, you can also specify the simple port name as in the following example:

```
current_design top
  set_case_analysis -name in15 -value 1
```

NOTE: *The value set on the signal specified with the `set_case_analysis` keyword is automatically propagated through the design. You do not need to set values for other signals in the path.*

NOTE: *If the `set_case_analysis` keyword is not specified or is incorrectly specified, the feature is not enabled.*

Consider the following example:

```
module test(clk1, clk2, testclock, tm, data, out);
  input clk1, clk2, testclock, tm, data;
  output out;

  reg out, temp;

  wire clock1, clock2;

  assign clock1 = tm ? testclock : clk1;
  assign clock2 = tm ? testclock : clk2;

  always @(posedge clock1)
    temp <= data;

  always @(posedge clock2)
    out <= temp;
endmodule
```

Normally, SpyGlass will mark the clock crossing between flip-flops `temp` and `out` if the clock names `clk1` and `clk2` are specified using the `clock`

keyword in a design constraints file.

However, SpyGlass will not mark the clock crossing between flip-flops temp and out if you also supply a design constraint file as follow:

```
current_design test
  set_case_analysis -name tm -value 1
```

To specify the values for a vector signal, you can specify the value as a Binary/Decimal/Hexadecimal number enclosed in curly brackets as in the following example:

```
set_case_analysis -name a[7:0] -value {h 0F}
```

The above specification can also be written with different value bases as follows:

```
set_case_analysis -name a[7:0] -value {d 15}
set_case_analysis -name a -value {b 00001111}
```

Values are assigned to vectors from left to right. If you have not given values for all the bits of the vector, then the remaining bits will all be assigned a value of 0. For example,

```
set_case_analysis -name a[7:0] -value {b 0101}
```

is equivalent to the following bit-value assignment:

```
a[0] = 0
a[1] = 0
a[2] = 0
a[3] = 0
a[4] = 1
a[5] = 0
a[6] = 1
a[7] = 0
```

The `set_case_analysis` constraint is used by the *InActiveReset-ML* tag.

Tags in VC SpyGlass Lint

This section explains the Tags available in the VC SpyGlass Lint product, under the following categories:

- [ERC Tags](#)
- [Latch Tags](#)
- [Lint Tags](#)
- [Miscellaneous Tags](#)
- [MoreLint Tags](#)
- [OpenMore Tags](#)
- [Simulation Tags](#)
- [STARC05 Tags](#)
- [Starcad-21 Tags](#)
- [Timing Tags](#)
- [Netlist Tags](#)

ERC Tags

This section describes the ERC Tags in VC SpyGlass Lint.

checkIOPinConnectedToNet

Input pins and IO pins of cells must be connected to a net.

Description

The *checkIOPinConnectedToNet* tag flags cell instances where an input pin or inout pin is not connected or is connected to dangling nets.

The *checkIOPinConnectedToNet* tag assumes that the nets specified in a file using the [supplyLow](#) and [supplyHigh](#) parameters are supply nets and ignores the input or inout pins connected to these nets for tag checking.

By default, the *checkIOPinConnectedToNet* tag reports only one violation per source. Set the [allviol](#) parameter to `yes` or `<tag-name>` to report all the pins connected to a dangling net.

By default, the *checkIOPinConnectedToNet* tag checks all cell instances. Set the value of the [report_only_libcell](#) parameter to `yes` to check for only lib cells.

Message Details

The following message appears for an input or inout pin `<pin-name>` that is unconnected or connected to a dangling net in instance `<inst-name>` of cell `<cell-name>`:

Input `<pin-name>` for instance `<inst-name>` (Cell `<cell-name>`) is unconnected

Severity

Warning

Examples

Consider the following example:

```
module top(in1, in2, out1, out2);
    input in1, in2;
    output out1, out2;

    wire n1, n2;
```

```
LAN2 I1(.A(),.B(in2),.Z(n1));  
LAN2 I2(.A(in1),.B(n2),.Z(out1));  
endmodule
```

In the instance I1 of the cell LAN2, the input A is unconnected. Therefore, the tag generates the following message:

Input top.I1.A for instance top.I1 (Cell LAN2) is unconnected

In the instance I2 of the cell LAN2, the input B is connected to a dangling net n2. Therefore, the tag generates the following message:

Input top.I2.B for instance top.I2 (Cell LAN2) is unconnected

checkMultipleDrivers

Only tristate pins may be WORed (multiple drivers only allowed, if all driving pins are of type tristate).

Description

The *checkMultipleDrivers* Tag flags internal nets with multiple drivers that are not all tristate signals.

This Tag ignores black box pins that are connected to the internal net.

If a net has multiple drivers and all the drivers are not tristate pins (pins with type `tristate` in the associated `.lib` file), the net is flagged and all drivers are listed.

By default, the tag extracts all the drivers in the fan-in cone. Set the value of the [MAX_DRIVER_COUNT](#) parameter to a positive integer value to limit the extraction of drivers in fan-in cone to a specified value.

Message Details

The following message appears for an internal net *<net-name>* that has multiple drivers *<driver-list>* and at least one driver is a non-tristate pin:

```
List of non tristate drivers for signal <net-name> : <driver-list>
```

Tag Severity

Warning

Examples

Consider the following example where the internal net `n1` has multiple drivers that are not tristate pins:

```
module top(in1, in2, in3, in4, out);
    input in1, in2, in3, in4;
    output out;

    wire n1, n2;

    LAN2 I1(.A(in1), .B(in2), .Z(n1));
```

```
    LAN2 I2(.A(in3),.B(in2),.Z(n1));  
    LAN2 I3(.A(n1),.B(in4),.Z(out));  
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

List of non tristate drivers for signal top.n1 : top.I1.Z
top.I2.Z

checkNetDriver

Each internal net must have at least one driver.

Description

The *checkNetDriver* tag flags undriven internal nets.

The *checkNetDriver* tag assumes that the nets specified in a file using the *supplyLow* and *supplyHigh* parameters are supply nets and ignores them for checking.

Message Details

The following message appears for an internal net *<hier-net-name>* that is undriven:

```
signal <hier-net-name> does not have any driver
```

The following message appears for the internal net *<hier-net-name>* that is tied to high impedance:

```
signal <hier-net-name> tied to high-impedance
```

Severity

Warning

Examples

Consider the following example where the wire n2 is undriven:

```
module top(in1, in2, out);
  input in1, in2;
  output out;

  wire n1, n2;

  LAN2 I1(.A(in1), .B(n1), .Z(out));
  LAN2 I2(.A(n2), .B(in2), .Z(n1));
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

```
signal top.n2 does not have any driver
```

FloatingInputs

Floating Input to a combinatorial gate

Description

The *FloatingInputs* Tag reports combinatorial cell instances that have floating inputs.

This tag reports violation even if the output of comb-gate is an internal net and the tag is unable to trace to a user-net.

The *FloatingInputs* Tag ignores input pins of combinatorial cell instances that are directly connected to a net specified using the [supplyNetFile](#) Tag parameter.

In most common IC technologies today, it is illegal to leave an input pin floating. If you do not need to use the input pin, it should be tied to the ground or VDD, as appropriate.

This tag supports [seq_atpg](#), that is, this Tag does not report a violation when the input is tied to a constant through SGDC constraint.

By default, the *FloatingInputs* Tag reports only one violation per source. Set the [ignore_chain_cell](#) Tag parameter to `yes` or `<Tag-name>` to report all messages.

Message Details

The following message appears at the location of cell `<cell-name>`, where the input pin `<in-pin-name>` is floating due to undriven net `<net-name>`:

```
[WARNING] Input pin <in-pin-name> on <cell-type> gate (master  
<cell-name>) corresponding to net <net-name> is undriven  
(connected to <pin-name>)
```

Where, `<pin-name>` is the name of the hanging net to which the input pin is connected and `<cell-type>` is the type of the cell, where the net is present.

Tag Severity

Warning

Examples

Consider the following example where input pin B of RTL_AND gate is connected to pin w1 of TOP module, which is undriven:

```
module blk (A, B, D);
    input A, B;
    output D;

    assign D = A & B;
endmodule

module TOP (A, B, D);
    input A, B;
    output D;

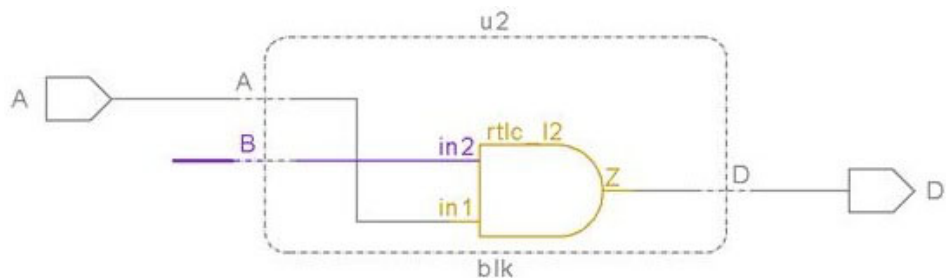
    wire w1;

    blk u2 (.A(A), .B(w1), .D(D));
endmodule
```

For this example, VC SpyGlass Lint reports the following message:

Input pin in2 on comb-and gate (master RTL_AND) corresponding to net TOP.D is undriven(connected to TOP.w1)

The following is the schematic for this example:



FlopClockUndriven

Flip-flop clock pin not driven

Description

The *FlopClockUndriven* Tag flags flip-flop instances where the clock pin is undriven. This necessarily means that either no net is connected to the clock pin or no gate drives the clock net.

This Tag supports [seq_atpg](#), that is, this Tag does not report a violation when the clock is tied to a constant through SGDC constraint.

Message Details

The following message appears at the location of the instance *<inst-name>* of the flip-flop cell *<cell-name>* where the clock pin *<clk-pin-name>* is undriven:

Clock pin *<clk-pin-name>* on Flop *<inst-name>* (master *<cell-name>*) is undriven(connected to *<pin-name>*)

Where, *<pin-name>* is the name of the hanging net to which the clock pin is connected.

By default, the *FlopClockUndriven* Tag reports only one violation per source. Set the [ignore_chain_cell](#) Tag parameter to *yes* or *<Tag-name>* to report all messages.

Tag Severity

Warning

Examples

Consider the following example where the clock pin *clk*, of the module *myFD*, connected to the wire *w5* is undriven:

```
module myFD (q, d, clk);
output q;
input d;
input clk;
reg q;
    always @ (posedge clk)
```

ERC Tags

```
begin
    q<=d;
end
endmodule

module Temp (out1, in1, in2);
output out1;
input in1,in2;
wire w2, w3, w5;
    myFD Flop1 (w3, w2,w5);
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Clock pin CP on Flop Temp.Flop1.q_reg (master RTL_FD) is undriven(connected to Temp.w5)

FlopDataUndriven

Flip-flop data pin not driven

Description

The *FlopDataUndriven* Tag flags flip-flop instances where the data pin is undriven.

The *FlopDataUndriven* Tag ignores the flip-flop instances where the data pin is directly connected to a net specified using the [supplyNetFile](#) Tag parameter.

This Tag supports [seq_atpg](#), that is, this Tag does not report a violation when the flip-flop data is tied to a constant through SGDC constraint.

By default, the *FlopDataUndriven* Tag reports only one violation per source. Set the [ignore_chain_cell](#) Tag parameter to `yes` or `<Tag-name>` to report all messages.

Message Details

The following message appears at the location of the instance `<inst-name>` of the flip-flop cell `<cell-name>` where the data pin `<data-pin-name>` is undriven:

Data pin `<data-pin-name>` on Flop `<inst-name>` (master `<cell-name>`) is undriven(connected to `<pin-name>`)

Where `<pin-name>` is the name of the hanging net to which the data pin is connected.

Tag Severity

Warning

Examples

Consider the following example where the data pin `d`, of the module `myFD`, connected to `top.q1` is undriven:

```
module myFD (q, d, clk);
output q;
input d;
input clk;
reg q;
```


ERC Tags

```
        always @(posedge clk)
        begin
            q<=d;
        end
    endmodule

module top (out, Clk);
input    Clk;
output out;
wire     q1;
    myFD    fd2(out,q1,clk);
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Data pin D on Flop top.fd2.q_reg (master RTL_FD) is
undriven(connected to top.q1)

DisabledAnd

Reports disabled AND or NAND gates

When to Use

Use this Tag to detect AND or NAND gates that are disabled.

Description

The *DisabledAnd* Tag reports AND or NAND gate instances for which at least one input is tied low. Therefore, these gate instances are effectively disabled.

If source net and reason for two AND gates that are disabled is same, then one violation is reported unless the *ignore_chain_cell* parameter is specified. This is not true if a common source net is a supply net, as different 1'b0/1'b1 in the module may lead to common VSS/VCC in the netlist.

Parameters

- *ignore_chain_cell*: Default value is no. This indicates the Tag reports only one violation per source. Set this parameter to yes or *<Tag-name>* to report all violation messages.
- *report_hanging_cell*: Default value is no. This indicates the Tag does not report the instances that have a hanging output. Set this parameter to yes to report all such instances in the design.
- *ignore_chain_cell*: Default value is no. This indicates the Tag reports violation for cascading chain of cells. Set this parameter to yes to not report violations for cascading chain of cells.
- *IGNORE_SAME_SUPPLY_SOURCE*: Default value is yes. Set the value of the parameter to no to enable the affected tags to report all the violations directly connected to the supply source.

Constraints

- *set_case_analysis* (Optional): The *DisabledAnd* Tag reports a violation when the AND or NAND gate is tied low through this SGDC constraint.

Messages and Suggested Fix

The following message appears at the location of the instance

`<inst-name>` of the AND or NAND type gate `<gate-name>` where the input pin `<pin-name>` is tied low through the net `<net-name>`:

[WARNING] Input pin `<pin-name>` on `<gate-type>` gate `<inst-name>` (master `<gate-name>`) is tiedlow(connected to `<net-name>`)

Where, `<gate-type>` can be `comb-and` or `comb-nand`.

Potential Issues

This violation appears when at least one input terminal of the AND or NAND gate is tied low.

Consequences of Not Fixing

If the input terminals of the AND or NAND gates are tied low, these gates are permanently disabled.

How to Debug and Fix

View the Incremental Schematic of the violation message. It highlights the AND or NAND gate, and the path between net that is tied low and the terminal of the AND or NAND gate.

To fix this problem, do not propagate a constant value to the terminal of the AND or NAND gate.

Example Code and/or Schematic

Example 1

Consider the following example where the input pin `in2` of the module `myand` is tied low:

```
module test(in1,out1);
input in1;
output out1;
supply0 w1;
    myand and1(out1,in1,w1);
endmodule
```

```
module myand(z,in1,in2);
input in1;
input in2;
output z;
    assign z = in1 & in2;
```

```
endmodule
```

Example 2

Consider the following example:

```
module DisabledAnd(output x, output y);  
wire a,b,c,d;  
  
assign x = a & b;  
assign y = c & d;  
  
assign a = 1'b0;  
assign c = 1'b0;  
endmodule
```

For the above example, the following two violations are reported:

Input pin in1 on comb-and gate DisabledAnd.x (master RTL_AND) is tiedlow(connected to DisabledAnd.a)

Input pin in1 on comb-and gate DisabledAnd.y (master RTL_AND) is tiedlow(connected to DisabledAnd.c)

Example 3

Consider the following example:

```
module DisabledAnd(output x, output y);  
wire a,b,c,d,common;  
  
assign x = a & b;  
assign y = c & d;  
  
assign a = common;  
assign c = common;  
  
assign common = 1'b0;  
endmodule
```

For the above example, the following violation is reported:

Input pin in1 on comb-and gate DisabledAnd.x (master RTL_AND) is tiedlow(connected to DisabledAnd.a)

ERC Tags

Default Severity Label

Warning

Tag Group

Constant Pin

Reports and Related Files

None

DisabledOr

Reports disabled OR or NOR gates

When to Use

Use this Tag to detect OR or NOR gates that are disabled.

Description

The *DisabledOr* Tag reports OR or NOR type gate instances for which at least one input is tied high. Therefore, these gate instances are effectively disabled.

Parameters

- *ignore_chain_cell*: Default value is `no`. This indicates the Tag reports only one violation per source. Set this parameter to `yes` or `<Tag-name>` to report all violation messages.
- *report_hanging_cell*: Default value is `no`. This indicates the Tag does not report the instances that have a hanging output. Set this parameter to `yes` to report all such instances in the design.
- *ignore_chain_cell*: Default value is `no`. This indicates the Tag reports violation for cascading chain of cells. Set this parameter to `yes` to not report violations for cascading chain of cells.
- *IGNORE_SAME_SUPPLY_SOURCE*: Default value is `yes`. Set the value of the parameter to `no` to enable the affected tags to report all the violations directly connected to the supply source.
- *REPORT_CONST_CONNECT*: Default value is `no`. Set the value of the parameter to `yes` to enable the tags to report constant source details in the violation message.

Constraints

- *set_case_analysis* (Optional): The *DisabledOr* Tag reports a violation when the OR or NOR gate is tied high through this SGDC constraint.

Messages and Suggested Fix

The following message appears at the location of the instance

`<inst-name>` of the OR or NOR type gate `<gate-name>` where the input pin `<pin-name>` is tied high through the net `<net-name>`:

[WARNING] Input pin `<pin-name>` on `<gate-type>` gate `<inst-name>` (master `<gate-name>`) is tiedhigh(connected to `<net-name>`)

Where, `<gate-type>` can be `comb-or` or `comb-nor`.

Potential Issues

This violation appears when at least one input terminal of the OR or NOR gate is tied high.

Consequences of Not Fixing

If the input terminals of the OR or NOR gates are tied high, these gates are permanently disabled.

How to Debug and Fix

View the Incremental Schematic of the violation message. It highlights the OR or NOR gate and the path between net that is tied high and the terminal of the OR or NOR gate.

To fix this problem, do not propagate a constant value to the terminal of the OR or NOR gate.

Example Code and/or Schematic

Consider the following example where the input pin `in2` of the module `myor` is tied high:

```
module test(in1,out1);
input in1;
output out1;
supply1 w1;
  myor or1(out1,in1,w1);
endmodule
```

```
module myor(z,in1,in2);
input in1;
input in2;
output z;
  assign z = in1 | in2;
endmodule
```

Default Severity Label

Warning

Tag Group

Constant Pin

Reports and Related Files

None

FlopClockConstant

Reports flip-flop clock pin driven by a constant value

When to Use

Use this Tag to detect flip-flops for which a clock pin is driven by a constant value.

Description

The *FlopClockConstant* Tag reports flip-flop instances for which a clock pin is tied to a constant value. This Tag assumes an explicit assignment of `x` to a clock pin as a constant assignment and reports such assignments.

Parameters

- ***allviol***: Default value is `no`. This indicates the Tag reports only one violation per source. Set this parameter to `yes` or `<Tag-name>` to report all violation messages.
- ***generate_connpoint_report***: Default value is `no`. When you set this parameter to `yes`, the tag generates a report that contains the connection point information of the constant source.
- ***report_inferred_cell***: Default value is `no`. This indicates the Tag reports all the flip-flops, including instantiated and inferred flip-flops, in the design. Set this parameter to `yes` to report only inferred flip-flops in the design.
- ***ignore_hanging_flop***: Default value is `no`. When you set this parameter to `yes`, the Tag does not report a violation for the flip-flop if its output is hanging.
- ***IGNORE_SAME_SUPPLY_SOURCE***: Default value is `yes`. Set the value of the parameter to `no` to enable the affected tags to report all the violations directly connected to the supply source.

Constraints

- ***set_case_analysis*** (Optional): The *FlopClockConstant* Tag reports a violation when the clock pin is tied to a constant value through this SGDC constraint.

Messages and Suggested Fix

The following message appears at the location of the instance `<inst-name>` of the flip-flop cell `<cell-name>` where the clock pin `<clk-pin-name>` is tied to a constant value, that is, tied high or tied low, through the net `<net-name>`:

[WARNING] Clock pin `<clk-pin-name>` on Flop `<inst-name>` (master `<cell-name>`) is `<value>` (connected to `<net-name>`)

Where, `<value>` can be `tiedhigh` or `tiedlow`.

Potential Issues

This violation appears when the clock pin of a flip-flop is driven by a constant value.

Consequences of Not Fixing

None

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the path from the flip-flop clock pin to the offending VSS or VDD net. It also highlights the path from the flip-flop clock pin to the offending net on which a constant value is set through the [set_case_analysis](#) constraint.

To fix the problem, do not connect the flip-flop clock pin to a constant.

Example Code and/or Schematic

Consider the following example where the `clk` clock pin, of the `myFD` module, connected to the `top.VSS` net is tied low:

```
module myFD (q, d, clk);
output q;
input d;
input clk;
reg q;
    always @(posedge clk)
    begin
        q<=d;
    end
endmodule
```

ERC Tags

```
module top();  
  wire    q7;  
  wire    d;  
  supply0 VSS;  
  
  myFD    fd1 (q7, d, VSS);  
endmodule
```

Default Severity Label

Warning

Tag Group

Constant Pin

Reports and Related Files

None

FlopDataConstant

Reports flip-flop data pin driven by a constant value

When to Use

Use this Tag to report flip-flop instances for which the data pin is tied to a constant value. This Tag improves functionality and testability (die size in some cases).

Description

The *FlopDataConstant* Tag reports flip-flop instances for which the data pin is tied to a constant value.

Parameters

- *ignore_chain_cell*: Default value is no. This indicates the Tag reports only one violation per source. Set this parameter to yes or *<Tag-name>* to report all violation messages.
- *report_hanging_cell*: Default value is no. This indicates the Tag does not report the instances that have a hanging output. Set this parameter to yes to report all such instances in the design.
- *IGNORE_SAME_SUPPLY_SOURCE*: Default value is yes. Set the value of the parameter to no to enable the affected tags to report all the violations directly connected to the supply source.
- *REPORT_CONST_CONNECT*: Default value is no. Set the value of the parameter to yes to enable the tags to report constant source details in the violation message.

Constraints

- *set_case_analysis* (Optional): The *FlopDataConstant* Tag reports a violation when the flip-flop data pin is tied to a constant value through this SGDC constraint.

Messages and Suggested Fix

The following message appears at the location of the instance *<inst-name>* of the flip-flop cell *<cell-name>* where the data pin *<data-pin-name>* is tied to a constant value that is tied high or tied low

through the net `<net-name>`:

[WARNING] Data pin `<clk-pin-name>` on Flop `<inst-name>` (master `<cell-name>`) is `<value>` (connected to `<net-name>`)

Where, `<value>` can be `tiedhigh` or `tiedlow`.

Potential Issues

This violation appears when the data pin of a flip-flop is driven by a constant value.

Consequences of Not Fixing

When the data pin of a flip-flop is tied to a constant value, asynchronous set-reset pins are required to register opposite data in the flip-flop. These schemes usually impact testability and timing.

However, in most cases, the logic can be further simplified if the data pin is tied to a constant value. It results in reducing the number of gates in the design and saving the area.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the path from the flip-flop data pin to the offending VSS or VDD net. It also highlights the path from the flip-flop data pin to the offending net on which a constant value is set through the [set_case_analysis](#) constraint.

To fix the problem, do not connect the flip-flop data pin to a constant.

Example Code and/or Schematic

Consider the following example where the data pin `d`, of the module `myFD`, connected to the net `top.VSS` is tied low:

```
module myFD (q, d, clk);
output q;
input d;
input clk;
reg q;
    always @(posedge clk)
    begin
        q<=d;
    end
end
```

```
endmodule

module top (out, Clk);
input Clk;
output out;
wire q1;
supply0 VSS;
    myFD fd2(out,VSS,clk);
endmodule
```

Default Severity Label

Warning

Tag Group

Constant Pin

Reports and Related Files

None

FlopEConst

Reports permanently disabled or enabled flip-flop enable pins

When to Use

Use this Tag to report flip-flops for which the enable pin is permanently disabled or enabled.

Description

The *FlopEConst* Tag reports flip-flop instances for which the enable pin is permanently disabled, that is, tied low if active high and tied high if active low, or permanently enabled, that is, tied high if active high and tied low if active low.

Parameters

- ***generate_connpoint_report***: Default value is no. When you set this parameter to yes, the tag generates a report that contains the connection point information of the constant source.
- ***ignore_chain_cell***: Default value is no. This indicates the Tag reports only one violation per source. Set this parameter to yes or *<Tag-name>* to report all violation messages.
- ***report_inferred_cell***: Default value is no. This indicates the Tag reports all the flip-flops, including instantiated and inferred flip-flops, in the design. Set this parameter to yes to report only inferred flip-flops in the design.
- ***ignore_hanging_flop***: Default value is no. When you set this parameter to yes, the Tag does not report a violation for the flip-flop if it's output is hanging.
- ***IGNORE_SAME_SUPPLY_SOURCE***: Default value is yes. Set the value of the parameter to no to enable the affected tags to report all the violations directly connected to the supply source.
- ***REPORT_CONST_CONNECT***: Default value is no. Set the value of the parameter to yes to enable the tags to report constant source details in the violation message.

Constraints

- [set_case_analysis](#) (Optional): The *FlopEConst* Tag reports a violation when the flip-flop enable pin is tied to a constant value through this SGDC constraint.

Messages and Suggested Fix

The following message appears at the location of the instance `<inst-name>` of the flip-flop cell `<cell-name>` where the enable pin `<en-pin-name>` is permanently enabled or disabled:

[WARNING] Enable pin `<en-pin-name>` on Flop `<inst-name>` (master `<cell-name>`) is always `<state1>`(`<state2>`)(connected to `<net-name>`)

Where, `<state1>` can be enabled or disabled and `<state2>` can be tied low or tied high.

Potential Issues

This violation appears if your design contains flip-flops for which the enable pin is permanently enabled or disabled.

Consequences of Not Fixing

If the enable pin is held constant, the flip-flop would be either always clocked or never clocked. In either case, the original intent needs to be reviewed.

However, in most cases, the logic can be further simplified if the enable pin is tied to a constant value. It results in reducing the number of gates in the design and saving the area.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the path from the flip-flop enable pin to the offending VSS or VDD net. It also highlights the path from the flip-flop enable pin to the offending net on which a constant value is set through the [set_case_analysis](#) constraint.

To fix the problem, do not connect the flip-flop enable pin to a constant.

Example Code and/or Schematic

Consider the following example where the enable pin `enable`, of the module `myflop`, connected to the pin `top.VSS` is always disabled:

ERC Tags

```
module myflop(q,clk,enable,d);
output q;
reg    q;
input  clk;
input  enable;
input  d;

    always @ (posedge clk)
        if (enable)
            q <= d;
endmodule

module top(q,clk,d);
output q;
input  clk;
input  d;

supply0 VSS;
    myflop f1(q,clk,VSS,d);

endmodule
```

Default Severity Label

Warning

Tag Group

Constant Pin

Reports and Related Files

None

FlopSRConst

Reports permanently enabled flip-flop set or reset pins

When to Use

Use this Tag to report flip-flops for which the set or reset pin is permanently enabled.

Description

The *FlopSRConst* Tag reports flip-flop instances for which the set or reset pins are permanently enabled, that is, tied high if active high and tied low if active low.

Parameters

- *ignore_chain_cell*: Default value is `no`. This indicates the Tag reports only one violation per source. Set this parameter to `yes` or `<Tag-name>` to report all violation messages.
- *report_inferred_cell*: Default value is `no`. This indicates the Tag reports all the flip-flops, including instantiated and inferred flip-flops, in the design. Set this parameter to `yes` to report only inferred flip-flops in the design.
- *ignore_hanging_flop*: Default value is `no`. When you set this parameter to `yes`, the Tag does not report a violation for the flip-flop if its output is hanging.
- *IGNORE_SAME_SUPPLY_SOURCE*: Default value is `yes`. Set the value of the parameter to `no` to enable the affected tags to report all the violations directly connected to the supply source.

Constraints

- *set_case_analysis* (Optional): The *FlopSRConst* Tag reports a violation when the flip-flop set or reset pin is tied to a constant value through this SGDC constraint.

Messages and Suggested Fix

The following message appears at the location of the instance `<inst-name>` of the flip-flop cell `<cell-name>` where the set or reset pin `<rst-pin-name>` is permanently enabled:

[WARNING] <pin-type> pin <rst-pin-name> on Flop <inst-name> (master <cell-name>) is always <state1> (<state2>)(connected to <net-name>)

Where, <pin-type> can be Set or Reset, <state1> can be set or reset, and <state2> can be tied low or tied high.

Potential Issues

This violation appears when a flip-flop cell has a set or reset pin that is permanently enabled.

Consequences of Not Fixing

When the set or reset pin of a flip-flop is permanently enabled, the output of the flip-flop is held at a constant value. This will impact testability of the logic cones including the flip-flop.

However, in most cases, the logic can be further simplified if the data pin is tied to a constant value. It results in reducing the number of gates in the design and saving the area.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the path from the flip-flop set or reset pin to the offending VSS or VDD net. It also highlights the path from the flip-flop set or reset pin to the offending net on which a constant value is set through the [set_case_analysis](#) constraint.

To fix the problem, do not connect the flip-flop set or reset pin to a constant.

Example Code and/or Schematic

Consider the following example where the set pin `rst` of the module `myFD` is tied high:

```
module myFD (q, d, clk, rst, clr);
    output q;
    input d;
    input clk;
    input rst;
    input clr;
    reg q;
    always @(posedge clk or posedge rst or posedge clr)
```

```
begin
  if(rst)
    q<=1;
  else if(clr)
    q<=0;
  else
    q<=d;
  end
endmodule

module top (out, Clk, q1, q2);
input Clk;
input q1;
input q2;

output out;
wire q1;
wire q2;
supply0 VSS;
supply1 VCC;
  myFD fd2(out,q1,clk,VCC,q2);
endmodule
```

Default Severity Label

Warning

Tag Group

Constant Pin

Reports and Related Files

None

LatchDataConstant

Reports latch data pins driven by a constant value

When to Use

Use this Tag to report latches for which the data pin is driven by a constant value.

Description

The Tag reports latch instances for which the data pin is tied to a constant value.

NOTE: For RTL_LDPC created by VC SpyGlass Lint, a violation is reported for constant D pin, only if corresponding set or reset pin is 'tied-high.' No violation is reported for the D pin if set or reset pin is 'tied-low' or 'non-constant,' even if D pin is a constant.

Parameters

- [*ignore_chain_cell*](#): Default value is `no`. This indicates the Tag reports only one violation per source. Set this parameter to `yes` or `<Tag-name>` to report all violation messages.
- [*report_hanging_cell*](#): Default value is `no`. This indicates the Tag does not report the instances that have a hanging output. Set this parameter to `yes` to report all such instances in the design.
- [*IGNORE_SAME_SUPPLY_SOURCE*](#): Default value is `yes`. Set the value of the parameter to `no` to enable the affected tags to report all the violations directly connected to the supply source.
- [*ENABLE_CHECKING_ON_ICGC*](#): The default value of this parameter is `no` and the tag does not perform any rule checking on the Data and Enable Pin of a Clock gating Cell Check (ICGC). Set the value of the parameter to `yes` for enabling the rule checking for Data and Enable Pin of an ICGC cell.

Constraints

- [*set_case_analysis*](#) (Optional): The Tag reports a violation when the latch data pin is tied to a constant value through this SGDC constraint.

Messages and Suggested Fix

The following message appears at the location of the instance `<inst-name>` of the latch cell `<cell-name>` where the data pin `<data-pin-name>` is tied to a constant value that is tied high or tied low through the net `<net-name>`:

[WARNING] Data pin `<data-pin-name>` on Latch `<inst-name>` (master `<cell-name>`) is `<value>` (connected to `<net-name>`)

Where, `<value>` can be `tiedhigh` or `tiedlow`.

Potential Issues

This violation appears when the data pin of a latch is driven by a constant value.

Consequences of Not Fixing

The logic might be intended. If it is not intended, the design intent needs to be reviewed.

However, in most cases, the logic can be further simplified if the data pin is tied to a constant value. It results in reducing the number of gates in the design and saving the area.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the path from the latch data pin to the offending VSS or VDD net. It also highlights the path from the latch data pin to the offending net on which a constant value is set through the [set_case_analysis](#) constraint.

To fix the problem, do not connect the latch data pin to a constant.

Example Code and/or Schematic

In the following example, the data pin `D`, of the module `myLatch`, connected to the net `LatchDataUndriven.w1` is tied low:

```
module myLatch (Q, D, E);
output Q;
input D, E;
reg Q;
    always @(D or E or Q)
begin
```

ERC Tags

```
        if (E)
            Q = D;
        end
    endmodule

    module LatchDataUndriven (out1, in1);
        output out1;
        input in1;
        supply0 w1;
        myLatch latch_2(out1,w1,in1);
    endmodule
```

Default Severity Label

Warning

Tag Group

Constant Pin

Reports and Related Files

None

LatchDataUndriven

Latch data pin not driven

Description

The *LatchDataUndriven* tag flags latch instances where the data pin is undriven.

The *LatchDataUndriven* tag flags a message if the data pin of a latch is undriven, that is, if no net is connected to the data pin or if no gate drives the data net.

This tag supports *seq_atpg*, that is, this tag does not report a violation when the latch data is tied to a constant through SGDC constraint.

The *LatchDataUndriven* tag ignores the latch instances where the data pin is directly connected to a net specified using the *supplyNetFile* parameter.

By default, the *LatchDataUndriven* tag reports only one violation per source. Set the *allviol* parameter to *yes* or *<tag-name>* to report all messages.

Message Details

The following message appears at the location of the instance *<inst-name>* of the latch cell *<cell-name>* where the data pin *<data-pin-name>* is undriven:

Data pin *<data-pin-name>* on latch *<inst-name>* (master *<cell-name>*) is undriven(connected to *<pin-name>*)

Where, *<pin-name>* is the name of the hanging net to which the data pin is connected.

Severity

Warning

Examples

Consider the following example where the data pin D, of the module myLatch, connected to the net LatchDataUndriven.w1 is undriven:

```
module myLatch (Q, D, E);  
output Q;
```



```

input D, E;
reg Q;
  always @(D or E or Q)
  begin
    if (E)
      Q = D;
    end
  endmodule

module LatchDataUndriven (out1, in1);
output out1;
input in1;
wire w1;
wire w2;
  myLatch latch_2(out1,w1,in1);
endmodule

```

For this example, VC SpyGlass generates the following message:

Data pin D on Latch LatchDataUndriven.latch_2.Q_reg (master RTL_LD) is undriven(connected to LatchDataUndriven.w1)

LatchEnableConstant

Reports latch-enable pin driven by a constant value

When to Use

Use this Tag to report latches for which the enable pin is driven by a constant value.

Description

The *LatchEnableConstant* Tag reports latch instances for which the enable pin is tied to a constant value.

For RTL_LDPC created by VC SpyGlass Lint, a violation is reported for constant Enable pin, only if corresponding set or reset pin is 'tied-high.' No violation is reported for the Enable pin if set or reset pin is 'tied-low' or 'non-constant,' even if the Enable pin is a constant.

Parameters

- *ignore_chain_cell*: Default value is no. This indicates the Tag reports only one violation per source. Set this parameter to yes or *<Tag-name>* to report all violation messages.
- *report_hanging_cell*: Default value is no. This indicates the Tag does not report the instances that have a hanging output. Set this parameter to yes to report all such instances in the design.
- *IGNORE_SAME_SUPPLY_SOURCE*: Default value is yes. Set the value of the parameter to no to enable the affected tags to report all the violations directly connected to the supply source.
- *ENABLE_CHECKING_ON_ICGC*: The default value of this parameter is no and the tag does not perform any rule checking on the Data and Enable Pin of a Clock Gating Cell (ICGC). Set the value of the parameter to yes for enabling the rule checking for Data and Enable Pin of an ICGC cell

Constraints

- *set_case_analysis* (Optional): The *LatchEnableConstant* Tag reports a violation when the latch enable pin is tied to a constant value through this SGDC constraint.

Messages and Suggested Fix

The following message appears at the location of the instance `<inst-name>` of the latch cell `<cell-name>` where the enable pin `<en-pin-name>` is tied to a constant value, that is tied high or tied low, through the net `<net-name>`:

[WARNING] Enable pin `<en-pin-name>` on latch `<inst-name>` (master `<cell-name>`) is always `<state1>`(`<state2>`)(connected to `<net-name>`)

Where, `<state1>` can be enabled or disabled and `<state2>` can be tied low or tied high.

Potential Issues

This violation appears when the enable pin of a latch is driven by a constant value.

Consequences of Not Fixing

When the enable pin of a latch is permanently enabled, the latch is transparent.

However, in most cases, the logic can be further simplified if the latch enable pin is driven by a constant value. It results in reducing the number of gates in the design and saving the area.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the path from the latch enable pin to the offending VSS or VDD net. It also highlights the path from the latch enable pin to the offending net on which a constant value is set through the [set_case_analysis](#) constraint.

To fix the problem, do not connect the latch enable pin to a constant.

Example Code and/or Schematic

In the following example, the enable pin E, of the module `myLatch`, connected to the pin `M1.w1` is always disabled:

```
module M1(out1,in1);
  output out1;
  input in1;
```

```
supply0 w1;  
  myLatch latch1(out1,in1,w1);  
endmodule  
  
module myLatch (Q, D, E);  
output Q;  
input D, E;  
reg Q;  
  always @(D or E or Q)  
  begin  
    if (E)  
      Q = D;  
  end  
endmodule
```

Default Severity Label

Warning

Tag Group

Constant Pin

Reports and Related Files

None

TristateConst

Reports tristate gate-enable pins driven by a constant value

When to Use

Use this Tag to report tristate gate instances for which the enable pin is driven by a constant value. This Tag improves functionality (bug escape), reliability, gate count, and power consumption.

Description

The *TristateConst* Tag reports tristate gate instances for which the enable pin is driven by a constant value.

Parameters

- [*IGNORE_SAME_SUPPLY_SOURCE*](#): Default value is yes. Set the value of the parameter to no to enable the affected tags to report all the violations directly connected to the supply source.

Constraints

- [*set_case_analysis*](#) (Optional): Use this SGDC constraint to report a violation when the tristate enable pin is tied to a constant value.

Messages and Suggested Fix

The following message appears at the location of the instance `<inst-name>` of the tristate gate `<gate-name>` where the enable pin `<en-pin-name>` is tied to a constant value through the net `<net-name>`:

[WARNING] Enable pin `<en-pin-name>` on Tristate gate `<inst-name>` (master `<gate-name>`) is always enabled (`<value>`)(connected to `<net-name>`)

Where, `<value>` can be tied high or tied low.

Potential Issues

This violation appears when an enable pin of a tristate gate is driven by a constant value.

Consequences of Not Fixing

The logic might be intended. The design intent needs to be reviewed.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the path from the tristate gate enable pin to the offending VSS or VDD net. It also highlights the path from the tristate gate enable pin to the offending net on which a constant value is set through the [set_case_analysis](#) constraint.

To fix the problem, do not connect the tristate gate enable pin to a constant.

In few IP reuse situations, the designer cannot change the IP and is forced to set the tristate enable to a constant value. In such cases, you can follow any of the following approaches:

- **Constant tristate enable permanently enables the driver**

Here, the tristate buffer can be replaced by a regular buffer simplifying test and verification efforts.

In this case, the original intent of using the tristate buffer should be reviewed. If there are other tristate buffers driving the same signal, the permanently enabled buffer's output can lead to electrical conflict, high current draw, and potential harmful electrical effects.

- **Constant tristate enable permanently disables the driver**

Here, the design intent should be reviewed. If the situation is acceptable, the tristate buffer can be considered as redundant and can be removed from the design.

Both the above approaches need a designer review.

Example Code and/or Schematic

In the following example, `en` is tied to a constant value that is connected to an enable pin of a tristate gate.

```
module tri_state_const (in, out);
    input in;
    output out;
    wire en;
    assign en = 1'b0;
    assign out = en? in : 1'bz;
endmodule
```

ERC Tags

**Default
Severity Label**

Warning

Tag Group

Constant Pin

**Reports and
Related Files**

None

checkPinConnectedToSupply

Reports IO or output pins of cells or modules that are connected to supply signals

When to Use

Use this Tag to report IO or output pins of cells or modules that are connected to supply signals.

Description

The *checkPinConnectedToSupply* Tag reports cell instances for which the inout (IO) or output pins are connected to supply lines VSS or VDD.

Parameters

- *allviol*: Default value is no. Set the value of the parameter to yes to report all messages. This includes reporting violations for sibling instances from different hierarchy. However, set the value of the *report_sibling_violation* app var to no to ignore the violations for the sibling instances. In this case, the *allviol* parameter is guarded under *report_sibling_violation* app var. The *allviol* parameter works only when the *report_sibling_violation* is set.
- *supplyLow*: Default value is `_null_`. Set this parameter to the name of the file containing supply low net names.
- *supplyHigh*: Default value is `_null_`. Set this parameter to the name of the file containing supply high net names.
- *ignore_chain_cell*: Default value is no. This indicates the Tag reports only one violation per source. Set this parameter to *yes* or *<Tag-name>* to report all violation messages.

Constraints

None

Messages and Suggested Fix

The following message appears for the instance *<inst-name>* of the cell *<cell-name>* where the inout or output pin *<pin-name>* is connected

to supply lines:

[WARNING] pin <pin-name> for instance <inst-name> (Cell <cell-name>) connected to supply signals

Potential Issues

This violation appears if your design contains an inout or output pin of a cell instance that is connected to a supply signal.

Consequences of Not Fixing

When the output of a cell is connected to a supply net, a conflict arises. As an example, consider the case when the output of an inverter is connected to VDD. When the inverter is driving out a 0, VDD supply is shorted to ground through the n-transistor of the inverter cell. This can lead to excessive current drawn from VDD and can potentially burn out the cell or chip.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the inout or output pin of a cell or a module is connected to a supply signal VSS or VDD.

Fix this problem by ensuring that the inout or output pin of a cell instance is not connected to supply signals.

Example Code and/or Schematic

Example 1

Consider the following example:

```
module top(in1, in2, in3, out);
    input in1, in2, in3;
    output out;

    LAN2 I1(.A(in1), .B(in2), .Z(VDD));
    LAN2 I2(.A(in3), .B(in2), .Z(VSS));
endmodule
```

The *checkPinConnectedToSupply* Tag reports a violation in the following cases:

- In the instance I1 of the cell LAN2, the output Z is connected to the VDD supply. VDD is present in the file specified by using the [supplyHigh](#) parameter.
- In the instance I2 of the cell LAN2, the output Z is connected to the VSS supply. VSS is present in the file specified by using the [supplyLow](#) parameter.

Example 2

Consider the following example:

```
interface simple_bus (input wire clk, input bit D, SI, output
bit SO, Q);
    logic mode;
    modport functionalMode (input mode, clk, SI, D, output Q,
SO);
    modport testMode (input mode, clk, SI, D, output Q, SO);
endinterface

module captureMode(simple_bus.functionalMode a);
    always @(*) begin
        if (a.clk && !a.mode)
            a.Q <= a.D;
        end
    endmodule

module shiftMode(simple_bus.testMode b);
    always_latch begin
        if (b.clk && b.mode)
            b.SO <= b.SI;
        end
    endmodule

module top(input logic D, CP, SI, output bit Q);
    supply1 SO;
    supply0 S1;
    simple_bus IF1 (.*, .Q(S1), .clk(CP));
    simple_bus IF2 (.SI, .D, .SO, .Q, .clk(CP));
    captureMode C1 (IF1.functionalMode);
    shiftMode C2 (IF2.testMode);
```

ERC Tags

```
endmodule
```

The *checkPinConnectedToSupply* Tag reports a violation in the following cases:

- When the pin `z` with the instance `top.rtlc_I1` is connected to supply signals.
- When the pin `z` with the instance `top.rtlc_I5` is connected to supply signals.

Default Severity Label

Warning

Tag Group

Electrical

Reports and Related Files

None

checkTristateBuses

Reports tristate signals that are not connected to busholder cells

When to Use

Use this Tag to report tristate signals that are not connected to the busholder cell instances. This Tag improves functionality (bug escape), reliability, reusability, and testability of a design.

Description

The *checkTristateBuses* Tag reports tristate signals that are not connected to the busholder cell instances. Tristate signals are the bus signals with the `tristate` type specified in the associated `.lib` file. These signals must be connected to the instances of only those cells for which the `bus_holder` attribute is set in the associated `.lib` file.

Most designers ensure that the tristate signals do not remain undriven for extended periods of time. However, sometimes the designer does not cover all the driver cases exhaustively. Therefore, it is best to have a busholder cell that eventually holds the signal to a known logic value of 0 or 1 and prevents non-digital behavior of the signal.

When an active driver is enabled for the signal, the busholder cell effect is masked due to vast difference in drive strengths of an active driver and a busholder cell. Busholder cells also prevent floating of the signal in test mode when it is possible to have no driver active for a given tristate signal.

Parameters

None

Constraints

None

Messages and Suggested Fix

The following message appears for an instance of a cell without the attribute `bus_holder` in the associated `.lib` file that is connected by the tristate bus `<net-name>`:

[WARNING] Tristate signal <net-name> is not connected to busholder cell

Potential Issues

This violation appears if your design contains tristate signals that are not connected to busholder cell instances.

Consequences of Not Fixing

Today, most of the logic designs do not use tristate buses. However, there are cases, such as legacy code and gate/power saving, where tristate buses are present in the design. The design intent needs to be reviewed.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the tristate signal that is not connected to busholder cell instances.

Fix this problem by connecting tristate signals to the busholder cells.

Example Code and/or Schematic

In the following example, the tristate signal `n1` is connected to the instance `I1` of the cell `LTS16` that does not have the attribute `bus_holder` specified in the associated `.lib` file:

```
module top(in1, in2, in3, in4, out);
    input in1, in2, in3, in4;
    output out;

    LTS16 I1(.A(in1), .E(in2), .Z(n1));
    LAN2 I2 (.A(n1), .B(in3), .Z(n2));
    LAN2 I3(.A(n2), .B(in4), .Z(out));
endmodule
```

Default Severity Label

Warning

Tag Group

Electrical

Reports and Related Files

None

NoContAssign

Continuous assignment statement present in technology-mapped netlist

Description

The *NoContAssign* Tag flags continuous assignment statements present in technology-mapped netlists (Verilog format).

It is a common situation where `assign` statements result in a netlist sometimes due to feedthrough paths or two outputs being generated from the same logic where a buffer has not been inserted. `assign` statements cause problems in downstream Place and Route tools.

Message Details

The following message appears for an `assign` statement found in a technology-mapped netlist:

```
Continuous assignment statement present in netlist design
```

Severity

Warning

Examples

Consider the following Verilog example where continuous assignment statements are present:

```
module top(in1,in2,out1,out2,out3,out4);  
  input in1, in2;  
  output out1,out2,out3,out4;  
  wire w1,w2,w3;  
  low inst1 (in1,in2,w1,w2);  
  assign out1 = w1;  
  assign out2 = w1;  
  low inst2 (in1,in2,out1,out2);  
  low inst3 (in1,in2,out2,out3);  
  low inst4 (in1,in2,out3,out4);  
endmodule
```

```
module low (in1,in2,o1,o2);  
input in1, in2;  
output o1,o2;  
  
assign o1 = in1 || in2;  
assign o2 = in1 && in2;  
endmodule
```


MuxSelConst

Reports MUX select pins driven by a constant value

When to Use

Use this Tag to report MUXes for which the select pin is driven by a constant value.

Description

The *MuxSelConst* Tag reports the MUX gate instances for which the select pin is tied to a constant value.

Parameters

- *ignore_chain_cell*: Default value is no. This indicates the Tag reports only one violation per source. Set this parameter to yes or *<Tag-name>* to report all violation messages.
- *strict*: Default value is no. This indicates the Tag does not honor the *set_case_analysis* values. Set this parameter to yes or MuxSelConst to honor the *set_case_analysis* values.
- *IGNORE_SAME_SUPPLY_SOURCE*: Default value is yes. Set the value of the parameter to no to enable the affected tags to report all the violations directly connected to the supply source.

Constraints

- *set_case_analysis* (Optional): Use this SGDC constraint to specify the case analysis conditions. By default, the *MuxSelConst* Tag does not honor the values specified by this constraint. To honor such values, set the value of the *strict* Tag parameter to yes or MuxSelConst.

Messages and Suggested Fix

The following message appears at the location of the instance *<inst-name>* of the MUX gate *<gate-name>* where the select pin *<sel-pin-name>* is tied to a constant value through the net *<net-name>*:

[WARNING] select pin *<sel-pin-name>* on Mux *<inst-name>* (master *<gate-name>*) is *<value>*(connected to *<net-name>*)

Where, *<value>* can be `tiedhigh` or `tiedlow`.

Potential Issues

This violation appears when the select pin of a MUX is driven by a constant value.

Consequences of Not Fixing

If the select pin of a MUX is constant, only one input pin of MUX is selected. In that case, the logic can be further simplified if MUX select is a constant. It results in reducing the number of gates in the design and saving the area.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the path from the MUX select pin to the offending VSS or VDD net. It also highlights the path from the MUX select pin to the offending net on which a constant value is set through the [set_case_analysis](#) constraint.

To fix the problem, do not connect the MUX select pin to a constant.

Example Code and/or Schematic

In the following example, the select pin `sel`, of the module `Mux2X1`, connected to the net `top.e` is tied low:

```
module top(in1,out);
input in1;
wire [1:0] w1;
output out;
supply0 e;
    Mux2X1 mux(w1,e,out);
endmodule

module Mux2X1(in1,sel,out);
input [1:0] in1;
input sel;
output out;
    assign out = in1[sel];
endmodule
```

ERC Tags

Default Severity Label

Warning

Tag Group

Constant Pin

Reports and Related Files

None

listTristateBuses

Lists all tristate buses used in the design

When to Use

Use this Tag to list all tristate buses used in the design.

Description

The *listTristateBuses* Tag lists all tristate buses in the design. Tristate signals are the bus signals with the `tristate` type specified in the associated .lib file.

Parameters

None

Constraints

None

Messages and Suggested Fix

The following message lists all tristate buses in the design:

[INFO] List of all tristate buses in the design :
<net-name-list>

Where, <*net-name-list*> is the comma-separated list of tristate buses found in the design.

Potential Issues

This violation appears if tristate buses are used in the design.

Consequences of Not Fixing

Tristate buses cause problems for DFT and require special attention to ensure that:

- They are driven under all circumstances.
- They can never be driven from more than one place at any given time.

How to Debug and Fix

ERC Tags

The violation message indicates the hierarchical signals that are used as tristate buses. Double-click the violation message. The HDL Viewer window highlights the line where the top-level module is declared.

To fix this problem, use MUX-based busing. MUX-based busing is a much safer strategy, wherever feasible. For all signals in the design, check if it is a tristate bus and write it out as a MUX.

Example Code and/or Schematic

Consider the following example that has one tristate bus signal n1:

```
module top(in1, in2, in3, in4, out);
    input in1, in2, in3, in4;
    output out;

    LTS16 I1(.A(in1), .E(in2), .Z(n1));
    LAN2 I2 (.A(n1), .B(in3), .Z(n2));
    LAN2 I3(.A(n2), .B(in4), .Z(out));
endmodule
```

Default Severity Label

Info

Tag Group

Electrical

Reports and Related Files

None

OutNotUsed

No output of a gate is used

Description

The *OutNotUsed* tag flags gate instances where none of their outputs is used in the design.

The *OutNotUsed* tag checks that at least one output of a gate instance is used.

By default, the number of messages has been restricted to 500 per design. Set the allviol tag parameter to *yes* or *<tag-name>* to report all messages.

Message Details

The following message appears at the location of instance *<inst-name>* of gate *<gate-name>* where none of the outputs of the gate instance is used in the design:

No outputs of *<gate-type>* gate *<inst-name>* (master *<gate-name>*) are used

Where, *<gate-type>* can be *comb-and*, *comb-or*, *flop*, or *latch*.

Tag Severity

Warning

Examples

Consider the following example where output of and gate instance of *top.M2.out1* is not used:

```
module top(in1,in2,out);  
  input in1;  
  input in2;  
  output out;  
  wire w2;  
      M2 m2(w2,in1,in2);  
endmodule
```

ERC Tags

```
module myand(z,in1,in2);  
  input in1;  
  input in2;  
  output z;  
    assign z = in1 & in2 ;  
endmodule  
  
module M2(out1,in4,in5);  
  output out1;  
  input in4,in5;  
  
    myand and2_1(out1,in4,in5);  
endmodule
```

For this example, SpyGlass generates the following message:

No outputs of comb-and gate top.m2.and2_1.rtlc_I2 (master RTL_AND) are used

FlopClockX

Flip-flop clock pin driven by an indeterminate value

Description

The *FlopClockX* tag flags flip-flop instances where the clock pin is driven by signals with indeterminate value.

The *FlopClockX* tag checks that there is no floating signals in the logic cone driving the clock pin of a flip-flop.

The *FlopClockX* tag does not flag explicit assignments of x to a clock pin. Such assignments are flagged by the *FlopClockConstant* tag.

This tag supports *seq_atpg*, that is, this tag does not report a violation when the clock is tied to a constant through the SGDC constraint.

By default, the *FlopClockX* tag reports only one violation per source. Set the *allviol* parameter to *yes* or *<tag-name>* to report all messages.

Message Details

The following message appears at the location of instance *<inst-name>* of flip-flop cell *<cell-name>* where there is a pin in the logic cone of the clock pin *<clk-pin-name>* that is connected to the net *<net-name>* of an indeterminate value:

Clock pin *<clk-pin-name>* on Flop *<inst-name>* (master *<flop-name>*) is xdriven(connected to *<net-name>*)

Severity

Warning

Examples

Consider the following example where the clock pin *clk*, of the module *myFD*, is connected to *FlopClockX.w2* of an indeterminate value:

```
module myFD (q, d, clk);  
output q;  
input d;  
input clk;  
reg q;
```



```

        always @(posedge clk)
        begin
            q<=d;
        end
    endmodule

module myAnd (z, in1, in2);
output z;
input  in1;
input  in2;
    assign z=in1 & in2;
endmodule

module FlopClockX(out1,in1,in2,in3,in4);
input in1,in2,in3,in4;
output out1;
wire w1,w2;
    myAnd and1(w2,in1,);
    myFD flop1(out1,w1,w2);
endmodule

```

For this example, VC SpyGlass Lint generates the following message:

Clock pin CP on Flop FlopClockX.flop1.q_reg (master RTL_FD) is xdriven(connected to FlopClockX.w2)

FlopDataX

Flip-flop data pin driven by an indeterminate value

Description

The *FlopDataX* tag flags flip-flop instances where the data pin is driven by signals with indeterminate value.

The *FlopDataX* tag ignores the flip-flop instances where all undriven signals connected to the data pin (ignoring combinational logic in the path) have been specified using the [supplyNetFile](#) parameter.

The *FlopDataX* tag checks that there is no floating signals in the logic cone driving the data pin of a flip-flop.

This tag supports [seq_atpg](#), that is, this tag does not report a violation when the flip-flop data is tied to a constant through the SGDC constraint.

By default, the *FlopDataX* tag reports only one violation per source. Set the [allviol](#) parameter to *yes* or *<tag-name>* to report all messages.

Message Details

The following message appears at the location of the instance *<inst-name>* of the flip-flop cell *<cell-name>* where there is a pin in the logic cone of the data pin *<data-pin-name>* that is connected to the net *<net-name>* of an indeterminate value:

Data pin *<clk-pin-name>* on flip-flop *<inst-name>* (master *<cell-name>*) is xdriven(connected to *<net-name>*)

Severity

Warning

Examples

Consider the following example where the data pin *d*, of the module *myFD*, is connected to the net *top.q1* of an indeterminate value:

```
module myFD (q, d, clk);  
output q;  
input d;  
input clk;
```

ERC Tags

```

reg q;
  always @(posedge clk)
  begin
    q<=d;
  end
endmodule

module myAnd(z,in1,in2);
output z;
input in1,in2;
  assign z = in1 & in2;
endmodule

module top (out, Clk, q1, q2);
input Clk;
input q1;
input q2;

output out;
wire    q1;
supply0 VSS;
  myAnd and1(q1,q2,);
  myFD    fd2(out,q1,clk);
endmodule

```

For this example, VC SpyGlass Lint generates the following message:

Data pin D on Flop top.fd2.q_reg (master RTL_FD) is
xdriven(connected to top.q1)

FlopSREX

Flip-flop set, reset or enable pin driven by an indeterminate value

Description

The *FlopSREX* tag flags flip-flop instances where any of the control pins (set, reset, or enable pin) is driven by signals with an indeterminate value.

The *FlopSREX* tag checks that there is no floating signals in the logic cone driving a control pin (set, reset, or enable pin) of a flip-flop.

This tag supports [seq_atpg](#), that is, this tag does not report a violation when the flip-flop set, reset, or enable pin is tied to a constant through the SGDC constraint.

The *FlopSREX* tag ignores a control pin (set, reset, or enable pin) of a flip-flop instance where all undriven signals connected to the control pin (ignoring combinational logic in the path) have been specified using the [supplyNetFile](#) parameter.

By default, the *FlopSREX* tag reports only one violation per source. Set the [allviol](#) parameter to *yes* or *<tag-name>* to report all messages.

Message Details

The following message appears at the location of instance *<inst-name>* of flip-flop cell *<cell-name>* where there is a pin in the logic cone of the control pin *<ctrl-pin-name>* (set, reset, or enable pin) that is connected to the net *<net-name>* of an indeterminate value:

control pin *<ctrl-pin-name>* on Flop *<inst-name>* (master *<cell-name>*) is xdriven(connected to *<net-name>*)

Severity

Warning

Examples

Consider the following example where the control pin `clr`, of the module `myFD`, connected to `top.w1` is driven with an indeterminate value:

```
module myFD (q, d, clk,rst,clr);  
output q;
```

ERC Tags

```

input d;
input clk;
input  rst;
input  clr;
reg    q;
    always @(posedge clk or posedge rst or posedge clr)
    begin
        if(rst)
            q<=1;
        else if(clr)
            q<=0;
        else
            q<=d;
    end
endmodule

```

```

module myAnd(z,in1,in2);
output z;
input in1;
input in2;
    assign z = in1 & in2;
endmodule

```

```

module top (out, Clk, q1, q2);
input  Clk;
input q1;
input q2;
wire w1;
output out;
wire q1;
supply0 VSS;
    myAnd and1(w1,q2,);
    myFD fd2(out,q1,clk,VSS,w1);
endmodule

```

For this example, VC SpyGlass Lint generates the following message:
**control pin CLR on Flop top.fd2.q_reg (master RTL_FDPC) is
 xdriven(connected to top.fd2.rtlc_N12)**

NearOverload

Load on Gate output is close to recommended maximum

Description

The *NearOverload* tag flags gate instances where the connected load to a pin is between 90% and 100% of the recommended maximum load for that pin.

Message Details

The following message appears at the location of instance *<inst-name>* of gate *<gate-name>* where the load on pin *<pin-name>* is between 90% and 100% of the recommended maximum load:

Load on pin *<pin-name>* on *<gate-type>* gate *<inst-name>* (master *<gate-name>*) exceeds 90% of the approved load

Where, *<gate-type>* can be comb-and, comb-or, flop, or latch.

Severity

Warning

Examples

Consider the following example:

```
module myOr (z, in1, in2);
output      z;
input       in1;
input       in2;

        OR2HV33      rtlc_I2 (.Z(z), .A(in1), .B(in2));
endmodule

module myAnd (z, in1, in2);
output      z;
input       in1;
input       in2;

        AND2HV33      rtlc_I2 (.Z(z), .A(in1), .B(in2));
```

ERC Tags

```

endmodule

module myFD (q, d, clk);
output    q;
input     d;
input     clk;
reg       q;

    FD1S1ANH33    rtlc_I5 (.Q(q), .D(d), .CK(clk));
endmodule

module M1 (out1, out2, out3, out4, in1, in2);
output    out1;
output    out2;
output    out3;
output    out4;
input     in1;
input     in2;
wire      w1;

    myFD    flop1 (w1, in1, in2);
    myAnd    and2_1 (out1, w1, in2);
    myAnd    and2_2 (out2, w1, in2);
    myAnd    and2_3 (out3, w1, in2);
    myAnd    and2_4 (out4, w1, in2);
endmodule

module OverLoad (out1, out2, in1, in2);
output    out1;
output    out2;
input     in1;
input     in2;
wire      w1;
wire      w2;
wire      w3;
wire      w5;

    M1      m1 (w1, w2, w3, w4, in1, in2);

```

```
myOr    or2_1 (out1, w1, w2);  
myOr    or2_2 (out2, w3, w4);  
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Load on pin Q on latch gate OverLoad.m1.flop1.rtlc_I5
(master FD1S1ANV33) exceeds 90% of the approved load@@-1

Overload

Load on Gate output exceeds recommended maximum

Description

The *Overload* tag flags gate instances where the connected load to a pin exceeds the recommended maximum load for that pin.

Message Details

The following message appears at the location of instance *<inst-name>* of gate *<gate-name>* where the load on pin *<pin-name>* exceeds the recommended maximum load by *<num>%*:

Load on pin *<pin-name>* on *<gate-type>* gate *<inst-name>* (master *<gate-name>*) exceeds recommended maximum by *<num>* percent

Where, *<gate-type>* can be comb-and, comb-or, flop, or latch.

Severity

Warning

Examples

Consider the following example:

```
module myOr (z, in1, in2);
output      z;
input       in1;
input       in2;

    OR2HV33    rtlc_I2 (.Z(z), .A(in1), .B(in2));
endmodule

module myAnd (z, in1, in2);
output      z;
input       in1;
input       in2;

    AND2HV33    rtlc_I2 (.Z(z), .A(in1), .B(in2));
endmodule
```

```
module myFD (q, d, clk);
output      q;
input       d;
input       clk;

    FD1S1ANHV33    rtlc_I5 (.Q(q), .D(d), .CK(clk));
endmodule

module M1 (out1, out2, out3, out4, in1, in2);
output      out1;
output      out2;
output      out3;
output      out4;
input       in1;
input       in2;
wire        w1;

    myFD        flop1 (w1, in1, in2);
    myAnd        and2_1 (out1, w1, in2);
    myAnd        and2_2 (out2, w1, in2);
    myAnd        and2_3 (out3, w1, in2);
    myAnd        and2_4 (out4, w1, in2);
endmodule

module OverLoad (out1, out2, in1, in2);
output      out1;
output      out2;
input       in1;
input       in2;
wire        w1;
wire        w2;
wire        w3;
wire        w4;

    M1          m1 (w1, w2, w3, w4, in1, in2);
    myOr        or2_1 (out1, w1, w2);
    myOr        or2_2 (out2, w3, w4);
```

```
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

```
Load on pin Q on latch gate OverLoad.m1.flop1.rtlc_I5 (master
FD1S1ANHV33) exceeds recommended maximum by 33.33 percent@@-1
```

Underload

Load on Gate output falls below recommended minimum

Description

The *Underload* tag flags gate instances where the connected load to a pin is less than 50% of the recommended maximum load for that pin.

The *Underload* tag does not report for unconnected pins or pins connected to dangling nets as such nets are flagged by other tag.

Message Details

The following message appears at the location of instance *<inst-name>* of gate *<gate-name>* where the load on pin *<pin-name>* is less than 50% of the recommended maximum load:

Load on pin *<pin-name>* on *<gate-type>* gate *<inst-name>* (master *<gate-name>*) fall below 50% of the approved load

Where, *<gate-type>* can be comb-and, comb-or, flop, or latch.

Severity

Warning

Examples

Consider the following example:

```
module AND2X1_TAX0 (Y, A, B);
output    Y;
input     A;
input     B;
endmodule

module myOr (Z, in1, in2);
output    Z;
input     in1;
input     in2;

    RTL_OR    rtlc_I2 (.Z(Z), .in1(in1), .in2(in2));
endmodule
```

ERC Tags

```

module myand1 (Z, in1, in2);
output      Z;
input       in1;
input       in2;

    AND2X1_TAX0      rtlc_I2 (.Y(Z), .A(in1), .B(in2));
endmodule

module e1 (out1, in1, in2);
output      out1;
input       in1;
input       in2;

    myand1      and2_1 (.Z(out1), .in1(in1), .in2(in2));
endmodule

module OverLoad (out1, out2, out3, out4, in1, in2, in3);
output      out1;
output      out2;
output      out3;
output      out4;
input       in1;
input       in2;
input       in3;
wire        w1;

    e1      Entity1 (.out1(w1), .in1(in1), .in2(in2));
    myand1   and2_2 (.Z(out1), .in1(w1), .in2(in3));
    myand1   and2_3 (.Z(out2), .in1(w1), .in2(in3));
    myand1   and2_4 (.Z(out3), .in1(w1), .in2(in3));
    myand1   and2_5 (.Z(out4), .in1(w1), .in2(in3));
endmodule

```

For this example, VC SpyGlass Lint generates the following messages:

Load on pin Y on comb-and gate OverLoad.and2_2.rtlc_I2 (master AND2X1_TAX0) fall below 50% of the approved load@@-1

Load on pin Y on comb-and gate OverLoad.and2_3.rtlc_I2 (master

AND2X1_TAX0) fall below 50% of the approved load@@-1
Load on pin Y on comb-and gate OverLoad.and2_4.rtlc_I2 (master
AND2X1_TAX0) fall below 50% of the approved load@@-1
Load on pin Y on comb-and gate OverLoad.and2_5.rtlc_I2 (master
AND2X1_TAX0) fall below 50% of the approved load@@-1

checkNetReceiver

Each internal net must have at least one receiver.

Description

The *checkNetReceiver* tag reports internal nets without receivers.

An internal net that does not have any receiver, that is, a pin with direction as input, inout, or bidirectional, is flagged.

Message Details

The following message appears for an internal net *<net-name>* that has no receiver:

```
Signal <net-name> does not have any receiver
```

Severity

Warning

Examples

Consider the following example where the wire *n1* has no receivers:

```
module top(in1, in2, out);
    input in1, in2;
    output out;

    wire n1, n2;

    LAN2 I1(.A(in1), .B(in2), .Z(out));
    LAN2 I2(.A(n2), .B(in2), .Z(n1));
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

```
Signal top.n1 does not have any receiver
```

LatchDataX

Latch data pin driven by an indeterminate value

Description

The *LatchDataX* tag flags latch instances where the data pin is driven by signals with indeterminate value.

This tag supports [seq_atpg](#), that is, this tag does not report a violation when the latch data is tied to a constant through the SGDC constraint.

The *LatchDataX* tag ignores the latch instances where all undriven signals connected to the data pin (ignoring combinational logic in the path) have been specified using the [supplyNetFile](#) parameter.

By default, the *LatchDataX* tag reports only one violation per source. Set the [allviol](#) parameter to `yes` or `<tag-name>` to report all messages.

Message Details

The following message appears at the location of instance `<inst-name>` of latch `<cell-name>` where there is a pin in the logic cone of the data pin `<data-pin-name>` that is connected to net `<net-name>` of an indeterminate value:

Data pin `<data-pin-name>` on latch `<inst-name>` (master `<cell-name>`) is `xdriven`(connected to `<net-name>`)

Severity

Warning

Examples

Consider the following example where the data pin D, of the module `myLatch`, connected to the net `LatchEnableX.w1` of an indeterminate value:

```
module myAnd (z, in1, in2);
output z;
input in1;
input in2;
    assign z = in1 & in2;
```



```

endmodule

module myLatch (Q, D, E);
output Q;
input D, E;
reg Q;
    always @(D or E or Q)
    begin
        if (E)
            Q = D;
    end
endmodule

module LatchEnableX (out1, in1, in2, in3);
output out1;
input in1;
input in2;
input in3;
wire w1;
wire w2;
    myAndand2_1 (w1, w2, );
    myLatch latch_2(out1,w1,in1);
endmodule

```

For this example, VC SpyGlass Lint generates the following message:

Data pin D on Latch LatchEnableX.latch_2.Q_reg (master RTL_LD)
is xdriven(connected to LatchEnableX.w1)

LatchEnableX

Latch enable pin driven by an indeterminate value

Description

The *LatchEnableX* tag reports latch instances where the enable pin is driven by signals with indeterminate value.

The *LatchEnableX* tag ignores the latch instances where all undriven signals connected to the enable pin (ignoring combinational logic in the path) have been specified using the [supplyNetFile](#) parameter.

This tag supports [seq_atpg](#), that is, this tag does not report a violation when the latch enable is tied to a constant through the SGDC constraint.

By default, the *LatchEnableX* tag reports only one violation per source. Set the [allviol](#) parameter to `yes` or `<tag-name>` to report all messages.

Message Details

The following message appears at the location of the instance `<inst-name>` of the latch cell `<cell-name>` where there is a pin in the logic cone of the latch enable pin `<en-pin-name>` that is connected to the net `<net-name>` of an indeterminate value:

Enable pin `<en-pin-name>` on latch `<inst-name>` (master `<cell-name>`) is xdriven(connected to `<net-name>`)

Severity

Warning

Examples

Consider the following example where the enable pin E, of the module `myLatch`, connected to the net `LatchEnableX.w1` is of an indeterminate value:

```
module myAnd (z, in1, in2);  
  output z;  
  input in1;  
  input in2;
```

ERC Tags

```

        assign z = in1 & in2;
    endmodule

    module myLatch (Q, D, E);
    output Q;
    input D, E;
    reg Q;
        always @(D or E or Q)
        begin
            if (E)
                Q = D;
        end
    endmodule

    module LatchEnableX (out1, in1, in2, in3);
    output out1;
    input in1;
    input in2;
    input in3;
    wire w1;
    wire w2;

        myAnd and2_1 (w1, w2, );
        myLatch latch_2(out1,in1,w1);
    endmodule

```

For this example, VC SpyGlass Lint generates the following message:

Enable pin E on Latch LatchEnableX.latch_2.Q_reg (master RTL_LD) is xdriven(connected to LatchEnableX.w1)

LatchEnableUndriven

Latch enable pin not driven

Description

The *LatchEnableUndriven* tag flags latch instances where the enable pin is undriven.

The *LatchEnableUndriven* tag flags a message if the enable pin of a latch is undriven, that is, if no net is connected to the enable pin or if no gate drives the enable net.

This tag supports *seq_atpg*, that is, this tag does not report a violation when the latch enable is tied to a constant through SGDC constraint.

The *LatchEnableUndriven* tag ignores the latch instances where the enable pin is directly connected to a net specified using the *supplyNetFile* parameter.

By default, the *LatchEnableUndriven* tag reports only one violation per source. Set the *allviol* parameter to *yes* or *<tag-name>* to report all messages.

Message Details

The following message appears at the location of the instance *<inst-name>* of the latch cell *<cell-name>* where the enable pin *<en-pin-name>* is undriven:

Enable pin *<en-pin-name>* on Latch *<inst-name>* (master *<cell-name>*) is undriven(connected to *<pin-name>*)

Where, *<pin-name>* is the name of the hanging net to which the enable pin is connected

Severity

Warning

Examples

Consider the following example where the enable pin *E*, of the module *myLatch*, is connected to *M1.w1* is undriven:

```
module M1(out1,in1);
```

```
output out1;
input in1;
wire w1;
  myLatch latch1(out1,in1,w1);
endmodule
```

```
module myLatch (Q, D, E);
output Q;
input D, E;
reg Q;
  always @(D or E or Q)
  begin
    if (E)
      Q = D;
  end
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Enable pin E on Latch M1.latch1.Q_reg (master RTL_LD) is undriven(connected to M1.w1)

noCombinatorialFeedBack

No combinatorial feedback loops allowed.

Description

The *noCombinatorialFeedBack* tag flags combinatorial loops.

The *noCombinatorialFeedBack* tag flags the first found combinatorial loop if there are more than one loop involving a particular cell instance in the design.

Message Details

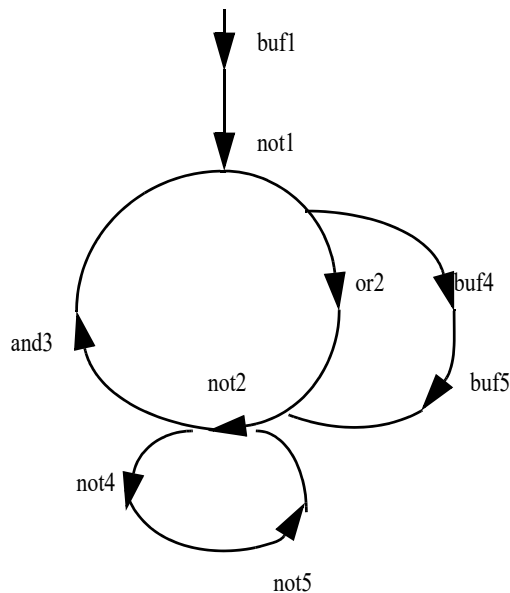
The following message appears when there is a combinatorial loop

<loop-inst-name-list>:

combinatorial feedback in instances *<loop-inst-name-list>*

Examples

Consider the following example:



While there are multiple combinatorial loops in the design, VC SpyGlass Lint generates the following message for the first found loop:

```
combinatorial feedback in instances 'delaychain3.not2-  
delaychain3.not4->delaychain3.not5->delaychain3.not2'
```

pwrnPinConnToSeqOrIOCells

PWRDN pins of cells should be connected to either IO cells or sequential elements.

Description

The pwrnPinConnToSeqOrIOCells tag flags PWRDN pins of cells that are not connected to either IO cells or sequential elements.

The pwrnPinConnToSeqOrIOCells tag requires you to specify the following:

- checkpwrpin parameter to enable the tag
- Name of the PWRDN pin using the pwrnpin parameter
- Names of the permitted IO cells using the iocells parameter

Then, the pwrnPinConnToSeqOrIOCells tag flags the following cases:

- The PWRDN pin is connected to a black box
- The PWRDN pin is not connected to an IO cell or a sequential element (flip-flop or latch)

In addition, the pwrnPinConnToSeqOrIOCells tag generates a report named pwrnreport that lists all PWRDN pin's cell names along with their connectivity. You can also access the pwrnreport report at the consolidated_report directory.

Message Details

The following message appears at an instance of the cell <cell-name> when its PWRDN pin <pin-hier-name> is unconnected:

Pin '<pin-hier-name>' (<cell-name>) is not connected to IO cell or sequential element, it is unconnected

The following message appears at an instance of the cell <cell-name> when its PWRDN pin <pin-hier-name> is connected to the pin <bb?pin?hier-name> of the black box <bb-name>:

Pin '<pin-hier-name>' (<cell-name>) is not connected to IO cell or sequential element, it is connected to black-box pin '<bb-pin-hier-name>' (<bb-name>)

The following message appears at an instance of the cell <cell-name> when its PWRDN pin <pin-hier-name> is connected to the hanging net

<net-hier-name> in the design unit <du-name>:

Pin '<pin-hier-name>' (<cell-name>) is not connected to IO cell or sequential element, it is connected to hanging net '<net-hier-name>' (<du-name>)

Severity

Error

Examples

Consider the following example:

```
module A(in1,in2,out1);
input in1,in2;
output out1;
supply1 s1;
wire w;
B u2(.out1(out1), .CWRDN(in1));
endmodule

module B(out1, CWRDN);
input CWRDN;
output out1;
endmodule

module MyTop(in1,in2,out1);
input in1,in2;
output out1;
wire w1;
wkup u1();
A u2(.in1(w1),.in2(in2),.out1(out1));
```

```
LEOF GREY_BOX(.A(in1),.B(in2),.Z(w1));  
endmodule
```

```
module wkup();  
endmodule
```

For this example, SpyGlass generates the following message:

Pin 'MyTop.u2.u2.CWRDN' (B) is not connected to IO cell or sequential element, it is connected to black-box pin 'MyTop.GREY_BOX.Z' (LEOF)@@-1

resetPinConnectedToResetNet

“Reset Pins” of cells may only be connected to signals predefined in a list of reset nets

Description

The *resetPinConnectedToResetNet* tag flags sequential cell instances where the reset pin is not connected directly or through combinational logic to any of the specified reset nets.

The *resetPinConnectedToResetNet* tag requires you to specify the hierarchical names of the reset nets to be checked in a design constraints file by using the *reset* keyword.

The *resetPinConnectedToResetNet* tag creates a file named `_erc_reset_constraint_info_` that has the hierarchical names of the reset nets specified in the design constraints file and their generated leaf-level reset nets found in the design.

Parameter(s)

RESETNET: The default value of the parameter is " ". Set the value of the parameter to a space-separated list of reset pin names to read the reset pin names specified using the SGDC or SDC commands.

Message Details

The following message appears for an instance *<inst-name>* of flip-flop cell *<cell-name>* where none of the nets specified by using the *reset* keyword in a design constraints file is connected to the reset pin *<pin-name>*:

Reset pin *<pin-name>* for instance *<inst-name>* (cell *<cell-name>*) is not connected to reset net

Examples

Consider the following example:

```
module top(a, clock, d);
  input a, clock;
  output d;
```

```
    wire rst;

    LFD3P flop1(.D(a),.CP(clock),RN(rst), .Q(d));
endmodule
```

If the net name `top.reset` is specified using the `reset` keyword in a design constraints file, VC SpyGlass LInt generates the following message:

Reset pin 'RN' for instance 'top.flop1' (Cell 'LFD3P') is not connected to reset net

setPinConnectedToSetNet

“Set Pins” of cells may only be connected to signals predefined in a list of set nets

Description

The *setPinConnectedToSetNet* tag flags sequential cell instances where the set pin is not connected directly or through combinational logic to any of the specified set nets.

The *setPinConnectedToSetNet* tag requires you to specify the hierarchical names of the set nets to be checked in a design constraints file using the [set](#) keyword.

The *setPinConnectedToSetNet* tag creates a file named `_erc_set_constraint_info_` that has the hierarchical names of the set nets specified in the design constraints file and their generated leaf-level set nets found in the design.

Parameter(s)

[SETNET](#): The default value of the parameter is " ". Set the value of the parameter to a space-separated list of set pin names to read the set pin names specified using the SGDC or SDC commands.

Message Details

The following message appears for an instance `<inst-name>` of the flip-flop cell `<cell-name>` where none of the nets specified using the `set` keyword in a design constraints file is connected to the set pin `<pin-name>`:

```
Set pin <pin-name> for instance <inst-name> ( Cell <cell-name>
) is not connected to set net
```

Examples

Consider the following example:

```
module top(a, clock, d);
    input a, clock;
    output d;
```

```
wire rst;  
  
    LFD3P flop1( .D(a), .CP(clock), .SN(rst), .Q(d) );  
endmodule
```

If the net name `top.set` is specified using the `set` keyword in a design constraints file, SpyGlass generates the following message:

Set pin 'SN' for instance 'top.flop1' (Cell 'LFD3P') is not connected to set net

clockPinsConnectedToClkNets

“Clock pins” of cells may only be connected to signals predefined in a list of clock-nets

Description

The *clockPinsConnectedToClkNets* tag flags sequential cell instances where the clock pin is not connected directly or through combinational logic to any of the specified clock nets.

The *clockPinsConnectedToClkNets* tag requires you to specify the hierarchical names of the clock nets to be checked in a design constraints file using the *clock* keyword.

The *clockPinsConnectedToClkNets* tag creates a file named `_erc_clock_constraint_info_` that has the hierarchical names of the clock nets specified in the design constraints file and their generated leaf-level clock nets found in the design.

Parameter(s)

CLOCKNET: The default value of the parameter is " ". Set the value of the parameter to a space-separated list of clock pin names to read the clock pin names specified using the SGDC or SDC commands.

Message Details

The following message appears for the instance *<inst-name>* of the flip-flop cell *<cell-name>* where none of the nets specified by using the *clock* keyword in a design constraints file is connected to the clock pin *<pin-name>*:

Clock pin *<pin-name>* for instance *<inst-name>* (cell *<cell-name>*) is not connected to clock net(*connected to <net-name>*)

Where, *<net-name>* is the name of the net to which the clock pin is actually connected in the design.

Examples

Consider the following example:

```
module top(a, clock, d);
    input a, clock;
```

```
output d;  
  
wire clk;  
  
LFD1 flop1( .D(a), .CP(clk), .Q(d) );  
endmodule
```

If the net name `top.clock` is specified using the `clock` keyword in a design constraints file, VC SpyGlass Lint generates the following message:

clock pin CP for instance `top.flop1 ( cell LFD1 )` is not connected to clock net(`connected to top.clk`)

checkOPPinConnectedToNet_a

At least one output port of a cell must be connected to a net.

Description

The *checkOPPinConnectedToNet_a* tag flags cell instances where the all outputs are either unconnected or are connected to dangling nets.

Message Details

The following message appears for an instance *<inst-name>* of cell *<cell-name>* where all output pins of the instance are either unconnected or are connected to dangling nets:

warning checkOPPinConnectedToNet_a: All the output port of instance *<inst-name>* (Cell *<cell-name>*) are unconnected or connected to a dangling net

Severity

Warning

Examples

Consider the following example:

```
module top(in1, in2, out1, out2);
  input in1, in2;
  output out1, out2;

  wire n1, n2;

  LEOF I1(.A(in1), .B(in2), .Z(n1));
  LEOF I2(.A(in1), .B(), .Z(n2));
endmodule
```

In instance I2 of cell LEOF, the inout pin B is unconnected and the output pin Z is connected to a dangling net n2. Therefore, VC SpyGlass Lint generates the following message:

warning checkOPPinConnectedToNet_a: All the output port of instance top.I2 (Cell LEOF) are unconnected or connected to a dangling net

checkOPPinConnectedToNet_b

All the output ports of the cell must be connected to a net.

Description

The *checkOPPinConnectedToNet_b* tag flags cell instances where some of the outputs are either unconnected or are connected to dangling nets.

The *checkOPPinConnectedToNet_b* tag checks if all the output ports are not connected to a net.

Message Details

The following message appears for an instance *<inst-name>* of cell *<cell-name>* where some output pins *<pin-name-list>* of the instance are either unconnected or are connected to dangling nets:

warning checkOPPinConnectedToNet_b: Following output pins of Instance *<inst-name>* (Cell *<cell-name>*) are unconnected or connected to a dangling net: *<pin-name-list>*

Severity

Warning

Examples

Consider the following example:

```
module top(in1, in2, out1, out2);
  input in1, in2;
  output out1, out2;

  wire n1, n2;

  LEOF I1(.A(in1), .B(in2), .Z(n1));
  LEOF I2(.A(in1), .B(), .Z(n2));
endmodule
```

In the instance I1 of the cell LEOF, the inout pin B is connected to the input in2 and the output pin Z is unconnected. Therefore, VC SpyGlass Lint generates the following message:

Following output pins of Instance top.I1 (Cell LEOF) are unconnected or connected to a dangling net: Z

elementsAllowedPerScanChain

List all scan chains, which contain more than n elements, as violating scan chains.

Description

The *elementsAllowedPerScanChain* tag flags scan chains with more than specified number of elements.

By default, the *elementsAllowedPerScanChain* tag stops when it encounters buffers and inverters does not allow.

Parameter(s)

- *allowBuffer*: Default value is no. Set the value of the parameter to yes to ignore buffers while traversing the scan chain.
- *allowInverter*: Default value is no. Set the value of the parameter to yes to ignore inverters while traversing the scan chain
- *allowedScanChainLength*: Default value is 1. Set the value of the parameter to a positive integer value to specify the maximum number of elements allowed in a scan chain.
- *thresholdScanChainLength*: Default value is 10000. Set the value of the parameter to a positive integer value to control the memory or time requirements for long scan chains during the SpyGlass run. If the length of a scan chain is greater than the number specified by the *thresholdScanChainLength* parameter, then only the first *thresholdScanChainLength* elements of the scan chain are reported. Make sure that the value of the *thresholdScanChainLength* parameter is reasonably large, for example, 10000. This ensures that the memory or time requirement is optimal and you can check most scan chains in the design. In any case, the value of the *thresholdScanChainLength* parameter must be more than the value of the *allowedScanChainLength* parameter.

Message Details

The following message appears when there is scan chain *<scan-inst-name-list>* of the length *<num>* with more elements than the number specified by the *allowedScanChainLength* parameter:

Found a scan chain of length <num> consisting of following instances : <scan-inst-name-list>

Examples

Consider the following example:

```
module scanchain (in1, clk1, out1);
    input in1,clk1;
    output out1;

    wire r1,r2,r3;

    LSFD2S flop1(.Q(out1), .TI(r3),.CP(clk1));
    LSFD2S flop2(.Q(r3), .TI(r2),.CP(clk1));
    LSFD2S flop3(.Q(r1), .TI(in1), .CP(clk1));
    LSFD2S flop4(.Q(r2), .TI(r1), .CP(clk1));
endmodule
```

When the value of the allowedScanChainLength parameter is set to 2, SpyGlass generates the following message:

Found a scan chain of length 4 consisting of following instances : scanchain.flop3(LSFD2S)->scanchain.flop4(LSFD2S)->scanchain.flop2(LSFD2S)->scanchain.flop1(LSFD2S)

Latch Tags

This section describes the Latch Tags in VC SpyGlass Lint.

ClockEdges

Reports design units that use both labels of the clock to drive latches

When to Use

Use this Tag to report design units in which latches are triggered by different levels of clocks.

Description

The *ClockEdges* Tag reports design units in which latches are triggered by different labels of a user-specified primary clock port, that is, if paths of opposite polarity or indeterminate polarity exist from any named clock signal to two or more latch enable inputs.

Polarity is determined by the gates through which the signal passes:

- BUF, AND, and OR gates have positive polarity.
- NOT, NAND, and NOR gates have negative polarity.
- All other gates have indeterminate polarity.

Prerequisites

The *ClockEdges* Tag requires the [latch_clock](#) parameter to be set to a list of primary port names.

Languages

Verilog and VHDL

Parameters

- [latch_clock](#): Default value is an empty string. Set this parameter to a comma-separated list of primary port names to report design units in which latches are triggered by different labels of one of these ports.
- [latch_clock_file](#): Default value is `latch_clock_edges`. Set this parameter to the name of a file in which the detailed messages are saved.
- [latch_verbose](#): Default value is `off`. Set this parameter to `on` to generate detailed messages. By default, these detailed messages are saved to a file named `latch_clock_edges` in the current directory. Use the [latch_clock_file](#) parameter to specify a different file.

Constraints

None

Messages and Suggested Fix

Message 1

The following message appears at the first line of a module or an architecture where latches are being triggered by different labels of a user-specified primary port *<name>*:

[WARNING] Do not use both edges of the clock *<name>* to drive latches

Potential Issues

This violation appears if your design contains latches that are triggered by different levels of clocks.

Consequences of Not Fixing

Latches that are triggered by different levels of clocks may lead to timing issues in your design.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the latches that use different phases of the clock, and the clock path.

Review the design to resolve the issue.

Message 2

The following message appears when an invalid or no argument is specified in the *latch_clock* parameter:

[INFO] Invalid or no argument passed to parameter *latch_clock*.
Tag checking skipped

Potential Issues

This violation appears if you specify an invalid or no argument in the *latch_clock* parameter.

Consequences of Not Fixing

If you do not specify an invalid or no argument in the *latch_clock* parameter, Tag checking will be skipped.

How to Debug and Fix

This is an informational message and requires no debug.

You need to specify a valid argument in the *latch_clock* parameter.

Example Code and/or Schematic

Verilog Example

Consider the following example where latches q1 and q2 are being triggered by different edges of the primary input clock clk:

```
module top (clk, data1, data2, rst, out1, out2);
  input clk, data1, data2, rst;
  output out1, out2;

  reg q1, q2;

  always @(clk or rst or data1)
  begin
    if (rst)
      q1 <= 1'b0;
    else if (clk)
      q1 <= data1;
    end

  wire enable = ~clk;

  always @(enable or rst or data2)
  begin
    if (rst)
      q2 <= 1'b0;
    else if (enable)
      q2 <= data2;
    end

  assign out1 = q1;
  assign out2 = q2;
endmodule
```

VHDL Example

Consider the following example where latches q1 and q2 are being triggered by different edges of the primary input clock clk:

```
library ieee;
use ieee.std_logic_1164.all

entity top is
    port (clk, rst, data : in bit;
          out1, out2 : out bit;
    end top;

architecture rtl of top is
    signal enable1 : bit;
    signal enable2 : bit;
    signal q1, q2 : bit;

begin
    enable1 <= clk;
    enable2 <= not clk;

    LD1: process (enable1, rst, data)
    begin
        if (rst = '1') then
            q1 <= '0';
        elsif ( enable1 = '1') then
            q1 <= data;
        end if;
    end process LD1;

    LD2: process (enable2, rst, data)
    begin
        if (rst = '1') then
            q2 <= '0';
        elsif ( enable2 = '1') then
            q2 <= data;
        end if;
    end process LD2;
```

Latch Tags

```
        out1 <= q1;  
        out2 <= q2;  
    end rtl;
```

Default Severity Label

Warning

Tag Group

latch

Reports and Related Files

None

LatchFeedback

Reports a latch in which a combinational feedback path exists from output pin to data or enable pin

When to Use

Use this Tag to report latches for which a feedback path exists from the latch output pin to the same latch data pin or latch enable pin. This Tag improves functionality, testability, and reliability of a design.

Description

The *LatchFeedback* Tag reports latch descriptions in which a feedback path exists from the latch output pin to the same latch data pin or latch enable pin, which means output of a latch is fed to the same latch.

In addition, this Tag checks for cascaded latch feedback. A violation is reported if a latch output is connected to the D pin of another latch whose output is connected back to the first latch, and both latches are in the same phase through common enable signals. The [set_case_analysis](#) values are considered, if specified.

Tag Exceptions

The *LatchFeedback* Tag does not report a violation in the following cases:

- When the combinational loop is through a blocked path
- When the latch enable pin is having a disable value, that is, directly connected to a constant value or through a [set_case_analysis](#) constraint value
- When the latch data pin is set to a constant value, that is, directly connected to a constant value or through a [set_case_analysis](#) constraint value

Languages

Verilog and VHDL

Parameters

- [effort_level](#): Specifies the effort level that VC SpyGlass Lint should make for checking the *LatchFeedback* Tag. Default value is 100. You can set the value of this parameter to any positive integer value.

- [*overlappingLatchLoops*](#): Enables detection of overlapping loops. Default value is yes. Set the value of the parameter to no to disable detection of overlapping loops. Violations of overlapping loops are reported, even if part of the overlapping loop does not contain a latch.
- [*latch_chain_max_length*](#): Default value is 2. Set this parameter to a positive integer value to specify/control the number of latches in the feedback path.

Constraints

[*set_case_analysis*](#) (Optional): Use this constraint to specify the case analysis conditions.

Messages and Suggested Fix

The following message appears at the first line of a design unit description where the Tag-violating latch *<name>* is described:

[WARNING] Potential feedback race for the latch with output *<name>*

Potential Issues

This violation appears when a combinational loop exists from the latch output to the data or enable pins of the same latch.

Consequences of Not Fixing

When there is a combinational feedback from the latch output to the data or enable pins of the same latch, a combinational loop is created during active high time of the enable signal. VC SpyGlass Lint DFT tools are generally unable to generate tests for combinational loop logic. In addition, timing behavior and ultimately functional behavior of logic in a combinational loop depend on the target technology because its behavior depends on gate and interconnect delays. Such designs may not be portable and may even have reliability problems if timing dependencies are strict enough to fall within manufacturing process variations.

Refer to the *CombLoop* Description in the *VC SpyGlass Lint OpenMore Tags Reference Guide* to learn more about the adverse effects of a combinational loop in a design.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the latch in which a combinational feedback path exists.

To fix the problem, search a place to break the combinational loop with a flip-flop, unless the loop is introduced by mistake.

Example Code and/or Schematic

Verilog Example

Consider the following example where there is a possible feedback loop for the latch q:

```
module top (enable, rst, data, out);
    input    enable, rst, data;
    output   out;

    reg q;
    wire daTagen;

    assign daTagen = data & q;

    always @(enable or rst or daTagen)
    begin
        if (rst)
            q <= 1'b0;
        else if(enable)
            q <= daTagen;
    end

    assign out = q;
endmodule
```

VHDL Example

Consider the following example where there is a possible feedback loop for the latch q1:

```
library ieee;
use ieee.std_logic_1164.all

entity top is
    port (output : out bit;
          enable, rst, data, gatingSig : in bit);
end top;
```

Latch Tags

```
architecture rtl of top is
    signal enaGen : bit;
    signal q1, q2 : bit;

begin
    LD1: process (enaGen, rst, data)
    begin
        if (rst = '1') then q1 <= '0';
            elsif (enaGen = '1') then q1 <= data;
        end if;
    end process LD1;

    enaGen <= q1 and gatingSig;

    LD2: process (enable, rst, q1)
    begin
        if (rst = '1') then q2 <= '0';
            elsif (enable = '1') then q2 <= q1;
        end if;
    end process LD2;

    output <= q2;
end rtl;
```

Default Severity Label

Warning

Tag Group

latch

Reports and Related Files

None

LatchGatedClock

Reports latches driven by gated or internally generated clock

When to Use

Use this Tag to report latches driven by gated or internally generated clock.

Description

The *LatchGatedClock* Tag reports latches that contain internally generated or gated clock or enable signals. This Tag supports block path analysis, that is, constant value propagation. Therefore, this Tag does not traverse on the blocked path.

Languages

Verilog and VHDL

Parameters

- *consider_sgdc_clock*: Default value is `no`. Set this parameter to `yes` to perform a check only for the clocks that are specified in the `sgdc` file.
- *checkunusedlatch*: Default value is `no`. This indicates the *LatchGatedClock* Tag does not report a violation for latches whose output is not used. Set this parameter to `yes` to report a violation for such latches.
- *strict*: Default value is `no`. This means the *LatchGatedClock* Tag reports only one violation per clock or enable net. Set this parameter to `yes` or `LatchGatedClock` to perform a strict check and report all the violations.
- *ignoreCellName*: Default value is not set and the Tag reports violations for all latches driven by gated or internally generated clocks. Set this parameter to PERL regular expression to specify the clocks gated through lib cell instances for which the violations should be ignored.
- *traverse_through_pad_pin*: Default value is 0. Set the value of the parameter to 1 to stop the fanin traversal through pin of pad cells while finding clock sources.
- *report_gated_clock_name*: The default value of this parameter is 1. Set the value of the parameter to 0 to disable reporting of the gated clock source signal name in the violation message.

Constraints

- *assume_path* (Optional): Use this constraint to specify the paths that exist between the input and output pins of black boxes. For black boxes, the *LatchGatedClock* Tag traverses further only if this constraint is specified in the SGDC file.
- *set_case_analysis* (Optional): Use this constraint to set the initial values for block path analysis. The *LatchGatedClock* Tag ignores clock or enable pin driven by a constant value that can be propagated through assignment or through this constraint.

Messages and Suggested Fix

The following message appears at the line where the latch output is set:

[WARNING] Latch with output <name> has it's clock gated or internally generated

Where, <name> refers to the name of the latch whose clock is gated or internally generated.

Potential Issues

This violation appears if your design contains latches that are driven by gated or internally generated clock.

Consequences of Not Fixing

Gating clocks create skew and timing problems, which further cause problems for ATPG tools.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the latches that contain internally generated or gated clock or enable signals, and the gated clock path.

Review the design to resolve the issue.

Example Code and/or Schematic

Verilog Example

Example 1

Consider the following example where latch `q` is being triggered by an internally generated clock `enable`.

```

module top (clk, rst, data, gatingSig, out);
    input clk, rst, data, gatingSig;
    output out;

    reg q;
    wire enable;

    assign enable = clk & gatingSig;

    always @(enable or rst or data)
        begin
            if (rst)
                q <= 1'b0;
            else if (enable)
                q <= data;
        end

    assign out = q;
endmodule

```

Example 2

Consider the following example with the ignoreCellName parameter set as "CK.*":

```

module top0 (clk, rst, data, gatingSig, out);

    input    clk, rst, data, gatingSig;
    output   out;
    reg q;
    wire enable;

    AN2 an2(clk, gatingSig, enable); //Violation, lib cell AN2
    is not
    //caught by the default regular expression value "CK*"

```

Latch Tags

```

always @(enable or rst or data)
begin
    if (rst)
        q <= 1'b0;
    else if (enable)
        q <= data;
    end
assign out = q;
endmodule

```

Example 3

Consider the following example with the ignoreCellName parameter set as "CK.*":

```

module top1 (clk, rst, data, gatingSig, out);

input    clk, rst, data, gatingSig;
output   out;
reg q;
wire enable;
CK_AN2 an2(clk, gatingSig, enable); //No Violation, lib cell CK_AN2 is
// caught by the regular expression value "CK.*"

always @(enable or rst or data)
begin
    if (rst)
        q <= 1'b0;
    else if (enable)
        q <= data;
    end
end

```

```
assign out = q;  
endmodule
```

VHDL Example

Consider the following example where latch `q` is being triggered by an internally generated clock enable.

```
library ieee;  
use ieee.std_logic_1164.all  
  
entity top is  
  port (clk : in bit;  
        rst : in bit;  
        data : in bit;  
        gatingSig : in bit;  
        output : out bit);  
end top;  
  
architecture rtl of top is  
  signal enable : bit;  
  signal q : bit;  
  
  begin  
    enable <= clk and gatingSig;  
  
    LD: process (enable, rst, data)  
    begin  
      if (rst = '1') then  
        q <= '0';  
      elsif ( enable = '1') then  
        q <= data;  
      end if;  
    end process LD;  
  
    output <= q;
```

Latch Tags

```
end rtl;
```

Default Severity Label

Warning

Tag Group

latch

Reports and Related Files

None

LatchReset

Reports design units in which the reset pins are used both synchronously and asynchronously

When to Use

Use this Tag to report design units in which a primary port is used to reset latches both synchronously and asynchronously. This Tag improves functionality (bug escape) and testability of a design.

Description

The *LatchReset* Tag reports design units in which specified primary port is used to reset latches both synchronously and asynchronously.

Synchronous reset function is triggered with respect to a clock edge or phase. A reset signal can transition well in advance. It means that the reset signal meets the required setup and hold times. However, the resetting effect is synchronized to the clock edge.

Asynchronous reset occurs immediately after the assertion of the reset signal. In this case, no clock edge or phase is required for resetting function.

This Tag supports block path analysis, that is, constant value propagation. Therefore, this Tag does not traverse on the blocked path.

Prerequisites

The *LatchReset* Tag requires a list of primary port names that can be specified by using the *latch_reset* parameter.

Languages

Verilog and VHDL

Parameters

- *Product Reports*: Default value is an empty string. Set this parameter to a comma-separated list of primary reset port names to report latch descriptions in which any of these ports is used to reset the latch both synchronously and asynchronously.
- *latch_verbose*: Default value is `off`. Set this parameter to `on` to generate the detailed messages.

- *latch_reset_file*: Default value is `latch_reset`. This indicates the detailed messages generated by the *LatchReset* Tag are saved in a file named `latch_reset`. Set this parameter to a different file name if you want to save the generated messages at a different location.

Constraints

- *assume_path* (Optional): Use this constraint to specify the paths that exist between the input pins and the output pins of black boxes. For black boxes, the *LatchReset* Tag traverses further only if this constraint is specified in the SGDC file.
- *set_case_analysis* (Optional): Use this constraint to set the initial values for block path analysis. The *LatchReset* Tag ignores latch reset pin driven by a constant value that can get propagated through assignment or through this constraint.

Messages and Suggested Fix

Message 1

The following message appears at the first line of a design unit if the primary port *<name>* is being used to reset latches both synchronously and asynchronously:

[WARNING] Reset signal *<name>* is used to reset the latches both synchronously and asynchronously

Potential Issues

This violation appears if your design contains a primary port that resets latches both synchronously and asynchronously.

Consequences of Not Fixing

If a reset signal is used in both synchronous and asynchronous manner, the integrity of scan and capture operations can be hampered during testing phase. However, such scenarios are relatively rare.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the reset pin and the latches where the reset pin is used both synchronously and asynchronously.

Review the design and determine if similar reset pins can be used in both synchronous and asynchronous manner. In addition, if a single signal is

used both synchronously and asynchronously, review the cycle partitioning intent.

Message 2

The following message appears when an invalid or no argument is specified in the *latch_reset* parameter:

[INFO] Invalid or no argument passed to parameter *latch_reset*.
Tag checking skipped

Potential Issues

This violation appears if you specify an invalid or no argument in the *latch_reset* parameter.

Consequences of Not Fixing

If you do not specify an invalid or no argument in the *latch_reset* parameter, Tag checking will be skipped.

How to Debug and Fix

This is an informational message and requires no debug.

You need to specify a valid argument in the *latch_reset* parameter.

Example Code and/or Schematic

Verilog Example

Consider the following example in which the primary input *rst* is used both as asynchronous and synchronous latch reset signals:

```
module top (enable, rst, data, out1, out2);  
    input enable, rst, data;  
    output out1, out2;  
  
    reg q1, q2;  
  
    always@(enable or rst or data)  
    begin  
        if(enable)  
            if (rst)  
                q1 <= 1'b0;  
            else  
                q1 <= data;  
    end
```



```

        end

        always@(enable or rst or data)
        begin
            if(rst)
                q2 <= 1'b0;
            else if (enable)
                q2 <= data;
            end

            assign out1 = q1;
            assign out2 = q2;
        endmodule

```

VHDL Example

Consider the following example in which the primary input `rst` is used both as asynchronous and synchronous latch reset signals:

```

library ieee;
use ieee.std_logic_1164.all

entity top is
    port (enable, rst, data : in bit;
          output1, output2 : out bit;
    end top;

architecture rtl of top is
    signal q1, q2 : bit;

    begin
        P1 : process (enable, rst, data)
        begin
            if (enable = '1') then
                if ( rst = '1') then
                    q1 <= '0';
                else
                    q1 <= data;
                end if;
            end if;
        end if;
    end if;

```

```
end process P1;

P2 : process (enable, rst, data)
begin
    if (rst = '1') then
        q2 <= '0';
    elsif (enable = '1') then
        q2 <= data;
    end if;
end process P2;

output1 <= q1;
output2 <= q2;
end rtl;
```

Default Severity Label

Warning

Tag Group

latch

Reports and Related Files

None

W122L

Variable read inside a latch always block should be inside sensitivity list

Description

The *W122L* tag flags `always` constructs modeling latches where a signal read in the construct, is missing from the sensitivity list.

Message Details

The following message appears at the first line of an `always` construct modeling a latch where a variable *<name>* is read in the construct but is missing from the sensitivity list:

variable '<name>' read within always construct is missing from the sensitivity list

Severity

Warning

Verilog Examples

Consider the following example where the signal `data` is read in the `always` construct modeling a latch but is not present in the sensitivity list:

```
module top (enable, data, out);
  input enable, data;
  output out;

  reg q;

  always @(enable)
  begin
    if (enable)
      q <= data;
  end

  assign out = q;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

```
variable 'data' read within always construct is missing from  
the sensitivity list
```

W442bL

Asynchronous reset condition can only use simple condition.

Language

Verilog

Description

The *W442bL* Tag flags complex asynchronous reset conditions.

It is recommended that asynchronous reset condition should be a simple condition, such as the following examples:

```
if (reset)
if (!reset)
if (reset==0)
if (reset==1)
```

Message Details

The following message appears at the location where a complex asynchronous reset condition is encountered:

Asynchronous reset condition should use a simple condition only

Severity

Warning

Verilog Examples

Currently Unavailable

W450L

Reports multi-bit expression used as latch enable condition

When to Use

Use this Tag to report multi-bit expressions or nets that are used as latch enable conditions.

Description

The *W450L* Tag reports multi-bit expressions or nets that are used as latch enable conditions.

Languages

Verilog

Parameters

None

Constraints

None

Messages and Suggested Fix

The following message appears at the location where the multi-bit expression *<expr>* is encountered in a latch enable condition:

[WARNING] Multi-bit expression *<expr>* used as latch enable may not be synthesizable

Potential Issues

This violation appears if your design contains multi-bit expressions or nets that are used as latch enable conditions.

Consequences of Not Fixing

Latches with multi-bit expressions or nets as enable condition may not be synthesizable.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the multi-bit expression is used as a latch enable.

Latch Tags

To fix the problem, do not use multi-bit expression as a latch enable. It is recommended to use simple nets as latch enable conditions.

Example Code and/or Schematic

Consider the following example where the multi-bit signal `enable` is used as the latch enable condition:

```
module top (enable, data, out);  
    input    [3:0] enable;  
    input    data;  
    output   out;  
    reg q;  
    always @(enable or data)  
        if (enable)  
            q <= data;  
    assign out = q;  
endmodule
```

Default Severity Label

Warning

Tag Group

latch

Reports and Related Files

None

W336L

Blocking assignment to latch output should be avoided

Language

Verilog

Description

The *W336L* tag flags blocking assignments used to assign latch outputs. When a blocking assignment is used in a sequential block, an inherent sequence of operation is implied in simulation. However, the synthesized hardware always behaves in a concurrent fashion. Hence, matching of gate-level simulations with RTL level simulations cannot be assured. In addition, the designer's intent may not be fully captured in RTL. Hence, it is best to review such violations during RTL creation phase.

Design Impact

Simulation, Synthesis, Functionality, RTL gates mismatch in some cases

Message Details

The following message appears at the location where the latch output *<name>* is assigned using blocking assignment:

```
Signal <name> is assigned using blocking assignment
```

Severity

Error

Verilog Examples

Consider the following example where the latch output *q* is assigned in a blocking manner in the always construct modeling a latch:

```
module top (enable, data, out);  
  input enable, data;  
  output out;  
  
  reg q;
```


Latch Tags

```
always @(enable or data)
begin
    if (enable)
        q = data;
    end

    assign out = q;
endmodule
```

For this example, SpyGlass generates the following message:

signal q is assigned using blocking assignment

W392aL

Do not use both edges of an asynchronous reset

Language

Verilog and VHDL

Description

The *W392aL* tag flags design units where both edges of a named reset signal are used to reset latches asynchronously.

Parameter(s)

- [*latch_reset*](#): Default value is an empty string. Set the value of the parameter to a user-specified list of primary port names to report design units where both edges of one of these ports are used to reset latches asynchronously.
- [*latch_verbose*](#): Default value is off. Set the value of the parameter to on or a comma-separated list of tags to generate detailed messages. By default, these detailed messages are saved to a file named `latch_reset` in the current directory.
- [*latch_reset_file*](#): Default value is `latch_reset`. Set the value of the parameter to name of a file in which the detailed messages are saved.

Message Details

Message 1

The following message appears at the first line of a design unit description where both edges of signal *<name>* are used to reset latches asynchronously:

Both edges of asynchronous reset signal *<name>* are used to reset latches

Message 2

The following message appears when an invalid or no argument is specified in the [*latch_reset*](#) parameter:

[INFO] Invalid or no argument passed to parameter `latch_reset`. Rule checking skipped

Severity

Warning

Verilog Examples

Consider the following example where both edges of primary input `rst` are used to reset latches asynchronously:

```
module top (enable, rst, data, out1, out2);
    output out1, out2;
    input enable, rst, data;

    reg out1, out2;

    //async reset rst used with negative edge
    always@(enable or rst or data)
    begin
        if (~rst)
            out1 <= 1'b0;
        else if (enable)
            out1 <= data;
        end

    //async reset rst used with positive edge
    always@(enable or rst or data)
    begin
        if (rst)
            out2 <= 1'b0;
        else if (enable)
            out2 <= data;
        end
    endmodule
```

When the above example is tag-checked with the primary port `rst` specified using the `latch_reset` parameter, VC SpyGlass Lint generates the following message:

Both edges of asynchronous reset signal `rst` are used to reset latches

VHDL Examples

Consider the following example where both edges of primary input `rst` are used to reset latches asynchronously:

```
library ieee;
use ieee.std_logic_1164.all

entity top is
  port (enable, rst, data : in bit;
        output1, output2 : out bit;
end top;

architecture rtl of top is
begin
  P1: process (enable, rst, data)
  begin
    if (rst = '0') then
      output1 <= '0';
    else if (enable = '1') then
      output1 <= data;
    end if;
  end process P1;

  P2: process (enable, rst, data)
  begin
    if (rst = '1') then
      output2 <= '0';
    else if (enable = '1') then
      output2 <= data;
    end if;
  end process P2;
end rtl;
```

When the above example is tag-checked with the primary port `rst` specified using the `latch_reset` parameter, VC SpyGlass Lint generates the following message:

Both edges of asynchronous reset signal `rst` are used to reset latches

W392bL

Do not use both edges of a synchronous reset

Language

Verilog and VHDL

Description

The *W392bL* flags design units where both edges of a named reset signal are used to reset latches synchronously.

Parameter(s)

- *latch_reset*: Default value is an empty string. Set the value of the parameter to a user-specified list of primary port names to report design units where both edges of one of these ports are used to reset latches asynchronously.
- *latch_verbose*: Default value is off. Set the value of the parameter to on or a comma-separated list of tags to generate detailed messages. By default, these detailed messages are saved to a file named *latch_reset* in the current directory.
- *latch_reset_file*: Default value is *latch_reset*. Set the value of the parameter to name of a file in which the detailed messages are saved.

Message Details

Message 1

The following message appears at the first line of a design unit description where both edges of signal *<name>* are used to reset latches synchronously:

Both edges of synchronous reset signal *<name>* are used to reset latches

Message 2

The following message appears when an invalid or no argument is specified in the *latch_reset* parameter:

[INFO] Invalid or no argument passed to parameter *latch_reset*.
Rule checking skipped

Severity

Warning

Verilog Examples

Consider the following example where both edges of primary input `rst` are used to reset latches synchronously:

```
module top (enable, rst, data1, data2, out1, out2);
    input    enable, rst, data1, data2;
    output   out1, out2;

    reg out1, out2;

    //sync reset with negative edge
    always @(enable or rst or data2)
    begin
        if(enable)
            if (~rst)
                out2 <= 1'b0;
            else
                out2 <= data2;
        end

    //sync reset with positive edge
    always @(enable or rst or data1)
    begin
        if(enable)
            if (rst)
                out1 <= 1'b0;
            else
                out1 <= data1;
        end
    endmodule
```

When the above example is checked with the primary port `rst` specified using the `latch_reset` parameter, VC SpyGlass Lint generates the following message:

Both edges of synchronous reset signal `rst` are used to reset

latches

VHDL Examples

Consider the following example where both edges of primary input `rst` are used to reset latches synchronously:

```
library ieee;
use ieee.std_logic_1164.all

entity top is
  port (enable, rst : in bit;
        data1, data2 : in bit;
        output1, output2 : out bit;
end top;

architecture rtl of top is
begin
  P1: process (enable, rst, data1)
  begin
    if (enable = '1') then
      if (rst = '1') then
        output1 <= '0';
      else
        output1 <= data1;
      end if;
    end if;
  end process P1;

  P2: process (enable, rst, data2)
  begin
    if (enable = '1') then
      if (rst = '0') then
        output2 <= '0';
      else
        output2 <= data2;
      end if;
    end if;
  end process P2;
```

```
end rtl;
```

When the above example is checked with the primary port `rst` specified using the `latch_reset` parameter, VC SpyGlass Lint generates the following message:

Both edges of synchronous reset signal `rst` are used to reset latches

W428L

Task should not be called inside a sequential always block

Language

Verilog

Description

The *W428L* tag flags task calls in sequential always constructs.

Message Details

The following message appears at the location where a task is called in a sequential always construct:

Task should not be called inside sequential always block

tag Severity

Warning

Verilog Examples

Consider the following example that has a task call `t1` in the sequential always construct:

```
module top (enable, rst, data, out);
    input enable, rst, data;
    output out;

    reg out;

    always @ (enable or rst or data)
        begin
            if (rst)
                out <= 1'b0;
            else if (enable)
                out <= data;
            t1;
        end
end
```

```
task t1;  
    $display(" A Task is being invoked");  
endtask  
endmodule
```

For this example, SpyGlass generates the W428L tag message.

W442aL

First statement in an asynchronous always block should be an `if` statement

Language

Verilog

Description

The *W442aL* tag flags always constructs modeling latches where the first statement is not an if statement.

Message Details

The following message appears at the first statement of an always construct modeling a latch that is not an if statement:

First statement in asynchronous always block must be an `if` statement

tag Severity

Warning

Verilog Examples

Consider the following example where the first line of the always construct modeling latch out2, is not an if statement:

```

module top (enable, rst, data, out1, out2);
    input enable, rst, data;
    output out1, out2;

    reg out1, out2;

    always @(enable or rst or data)
    begin
        out1 = ~data;
        if (rst)
            out2 <= 1'b0;
        else if (enable)

```

```
        out2 <= data;  
    end  
endmodule
```

For this example, SpyGlass generates the W442aL tag message.

W429L

Task should not be called inside a combinational always block

Language

Verilog

Description

The *W429L* tag flags task calls in combinational always constructs.

Message Details

The following message appears at the location where a task is called in a combinational always construct:

Task should not be called inside a combinational always block

Severity

Warning

Verilog Examples

Consider the following example that has a task call `t1` in the combinational always construct:

```

module top (data1, data2, out);
    input data1, data2;
    output out;

    reg tmp, out;

    always @(data1 or data2 or out)
        begin
            tmp = ~data2;
            t1 (data1, tmp, out);
        end

    task t1;
        input data1, tmp;
        output out;

```

```
        out = data1 & tmp;  
    endtask  
endmodule
```

For this example, VC SpyGlass Lint generates the W429L message.

W449L

Expression used as latch enable may not be synthesizable

Language

Verilog

Description

The *W449L* tag flags expressions used as latch enable conditions.

Latches with expressions in enable condition may not be synthesizable.

Therefore, it is recommended to use simple nets as latch enable conditions.

Message Details

The following message appears at the location where an expression *<expr>* is encountered in a latch enable condition:

Expression *<expr>* used as latch enable may not be synthesizable

Tag Severity

Warning

Verilog Examples

Consider the following example where the enable of latch *q* is an expression (*~enable*):

```
module top (enable, rst, data, out);  
    input  enable, rst, data;  
    output out;  
  
    reg temp, q;  
  
    always @(enable or rst or data)  
        if (rst)  
            temp <= 1'b0;  
        else if (enable)  
            temp <= data;
```

```
always @(enable or rst or temp)
  if (rst)
    q <= 1'b0;
  else if (~enable)
    q <= temp;

  assign out = q;
endmodule
```

For this example, SpyGlass generates the following message:

Expression (~enable) used as latch enable may not be synthesizable

GatedReset

Do not use gated or internally generated resets to reset the latches

Description

The *GatedReset* tag flags latch descriptions that use the specified primary port as latch reset after combinational transformation.

The *GatedReset* tag traverses the path from the specified port and flags a message if the traversed path has only combinational logic (ignoring buffers and inverters) and ends at the reset pin of a latch.

Parameter(s)

latch_reset: Default value is an empty string. Set the value of the parameter to the name of the reset pin for latch in the design to specify a list of primary reset port names for the *GatedReset* tag.

Message Details

Message 1

The following message appears at the first line of an always construct or a process construct describing a latch with *<sig-name>* as its data if the reset of the latch is gated:

[WARNING] Reset pin is gated for the latch with data *<sig-name>*

Message 2

The following message appears when an invalid or no argument is specified in the *latch_reset* parameter:

[INFO] Invalid or no argument passed to parameter *latch_reset*. Rule checking skipped

Severity

Warning, info

Verilog Examples

Consider the following example where tracing from the primary input *rst*, the trace path ends at the reset pin (*rstGated*) of latch *q*:

```

module top (enable, rst, data, gatingSig, out);
    input enable, rst, data, gatingSig;
    output out;

    wire rstGated;
    reg q;

    assign rstGated = rst & gatingSig;

    always @(enable or rstGated or data)
        begin
            if (rstGated)
                q <= 1'b0;
            else if (enable)
                q <= data;
        end

    assign out = q;
endmodule

```

When the above example is checked with the primary port `rst` specified using the `latch_reset` parameter, VC SpyGlass Lint generates the following message:

Reset pin is gated for the latch with data top.q

VHDL Examples

Consider the following example where tracing from the primary input `rst`, the trace path ends at the reset pin (`rstGated`) of latch output:

```

library ieee;
use ieee.std_logic_1164.all

entity top is
    port (enable, rst, data, gatingSig : in bit;
          output : out bit);
end top;

architecture rtl of top is

```

Latch Tags

```
signal rstGated : bit;
signal q : bit;

begin
    rstGated <= rst and gatingSig;

    P1: process (enable, rstGated, data)
    begin
        if (rstGated = '1') then
            output <= '0';
        elsif (enable = '1') then
            output <= data;
        end if;
    end process P1;
end rtl;
```

When the above example is checked with the primary port `rst` specified by using the `latch_reset` parameter, VC SpyGlass Lint generates the following message:

Reset pin is gated for the latch with data `top.output`

Lint Tags

This section explains the lint Tags in VC SpyGlass Lint.

AllocExpr

Identifies the allocator expressions which are not synthesizable

When to Use

Use this Tag to identify the allocator expressions which are not synthesizable by some synthesis tools.

Description

The *AllocExpr* Tag reports violation for the allocator expressions which are not properly synthesizable by some synthesis tools.

Default Weight

5

Language

VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for an allocator expression which is not synthesizable by a synthesis tool:

[Warning] Allocator expression may not be synthesizable

Potential Issues

A violation is reported when an unsynthesizable allocator expression is encountered in a design.

Consequences of Not Fixing

Using the unsynthesizable allocator expression could lead to unsynthesizable code.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the violating allocator expression is used. This expression is un-synthesizable by several commercially available synthesis tools.

No fix is required in testbenches or simulation models. However, avoid this style in the code targeted for synthesis.

Example Code and/or Schematic

Consider the following example:

```
begin

process
  variable MOD1PTR,MOD2PTR:PTR;
begin
  MOD1PTR := new MODULE;

  MOD2PTR := new MODULE'(25, 10 ns, 4, 9);
end process;
```

In the above example, the AllocExpr Tag reports a violation as MOD1PTR and MOD2PTR are allocator expressions.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

No related reports or files.

ArrayEnumIndex

Identifies the arrays using enumeration type as index

When to Use

Use this Tag to identify the arrays which are not synthesizable by some synthesis tools.

Description

The *ArrayEnumIndex* Tag reports violations for the arrays defined using enumeration type as index.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed where an array is declared with enumeration type as its index as it is not synthesizable by a synthesis tool:

[WARNING] Array defined using enumeration type as index may not be synthesizable

Potential Issues

Violation may arise when an array is declared with enumeration type as its index.

Consequences of Not Fixing

Arrays defined using enumeration type as index are not synthesizable by some synthesis tools

How to Debug and Fix

Double-click the violation message. The HDL window highlights the line where an array is declared with an enumeration type as its index. This is un-synthesizable by several commercially available synthesis tools.

No fix is required in testbenches or simulation models. However, avoid this style in the code targeted for synthesis.

Example Code and/or Schematic

Consider the following example:

```
architecture arc of ent is
type my_type is (low1,mid1,high1);
type my_array is array(my_type(low1) to my_type(high1))
of bit;
```

In the above example, the ArrayEnumIndex Tag reports a violation as the array, my_array, is declared using enumeration type as index.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

No related reports or files.

AssertStmt

Assertion statements have no significance in synthesis

Language

VHDL

Description

The AssertStmt Tag flags `assert` statements.

Assertions are for functional verification and have no meaning in synthesis. The `assert` statements are ignored by some synthesis tools.

NOTE: *Synopsys recommends use of `translate_off` and `translate_on` directives against `synthesis_off` and `synthesis_on` directives. If `synthesis_off` and `synthesis_on` directives are used within an expression, they have the potential to create incorrect logic by synopsys tools.*

Message Details

The following message appears at the location where an ASSERT statement is encountered:

Assertion statement may be ignored by `synthesis tools`

Severity

Warning

Suggested Fix

To avoid messages, enclose these statements in `synthesis_off` and `synthesis_on` pragmas.

badimplicitSM1

Identifies the sequential logic in a non-synthesizable modelling style where clock and reset cannot be inferred

When to Use

Use this Tag to identify the sequential descriptions in a module which are not synthesizable by some synthesis tools.

Description

The *badimplicitSM1* Tag reports violation for the sequential descriptions in a module where clock and reset cannot be distinguished and the module is not synthesizable by some synthesis tools.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed at the last line of a module which violates the Tag:

[ERROR] The 'if-else if' statement chain conditional expression should check for all the asynchronous reset/set signals first

Potential Issues

Violation may arise when uncommon sequential logic is used, which is unsynthesizable.

Consequences of Not Fixing

The module is not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the start of the `if-else-if` block. Scroll down the HDL Viewer window to search for the `always` block. To confirm the non-synthesizable modeling style for a sequential logic, perform one of the following steps:

- Inspect the block visually
- Scroll up the HDL Viewer window to search for the `always` construct
- Search the HDL for the `if-else` block inside the `always` construct
- Browse through the assignments and conditions used in the `if-else` block

Since the implicit modeling style is uncommon, it is most likely you made a mistake in modeling a conventional synchronous block. For a standard synchronous logic, it is neither required nor a good practice to use signals other than those present in the sensitivity list to set/reset the circuit.

Example Code and/or Schematic

In the following example code, VC SpyGlass Lint generates a violation as it cannot distinguish between the clock and the reset signals:

```
module bism1(set,reset,in1,in2,out1);

input in1,in2,reset,set;
output out1;
reg clk,out1;
always @(posedge clk or negedge set)
    if(reset)
        out1 = 0;
    else if(!set)
        out1 = 1;
    else if(in2)
        out1 = in2;
    else
        out1 = in1;
endmodule
```

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

No related reports or files.

badimplicitSM2

Identifies the implicit sequential logic in a non-synthesizable modeling style where states are not updated on the same clock phase

When to Use

Use this Tag to identify the implicit sequential descriptions in a non-synthesizable module which are not synthesizable by some synthesis tools.

Description

The *badimplicitSM2* Tag reports violations for the sequential descriptions in a module whose states are updated on different clock edges and it is not synthesizable by some synthesis tools.

The implicit sequential logics are synthesizable only if all transitions within a block switch on the same edge of the same clock. Finite-state Machines (FSMs) are not synthesizable if you attempt to switch on both edges of the same clock or same edges of different clocks.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for the implicit sequential logic which is not synthesizable by a synthesis tool :

```
[ERROR] unsynthesizable implicit sequential logic: states can only be updated on same clock phase
```

Potential Issues

A register which is updated on both edges of a clock in a given sequential block leads to ambiguous synthesis results.

Consequences of Not Fixing

Following are the consequences of not fixing the violation:

- The synthesis tool can get confused about which edge to use for updating the register.
- RTL and gate-level simulation results may not match.

How to Debug and Fix

To debug the violation, double-click the message. The HDL Viewer window highlights the line where switching on the second edge of the same clock is done. To confirm the implicit sequential logic using both clock phases, perform one of the following steps:

- Inspect the block visually
- Scroll up the HDL Viewer window to search for the `always` construct
- Search the HDL for the edge of clock used in the `if` statement

To resolve the violation, break the sequential logic into multiple sequential logic blocks, each of which can independently meet the requirement. If the register depends on both edges of the clock, describe the sequential nature separately and use the combinational logic to generate the final output.

Example Code and/or Schematic

In the following example code, VC SpyGlass Lint generates a violation as the state depends on more than one edge of the `clk` clock:

```
module test(out1,out2);  
  output out1,out2;  
  reg out1,out2,a,c,clk;  
  always  
  begin  
    @(posedge clk) out1 <= c;  
    @(negedge clk) out2 <= a;  
  end  
endmodule
```

Lint Tags

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

No related reports or files.

badimplicitSM4

Identifies the non-synthesizable implicit sequential logic where event control expressions have multiple edges

When to Use

Use this Tag to identify the sequential descriptions which are not synthesizable by some synthesis tools.

Description

The *badimplicitSM4* Tag reports violations for sequential descriptions where event control expressions use more than one edge. These descriptions are not synthesizable by some synthesis tools.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for an event control expression which uses another edge:

[ERROR] unsynthesizable implicit sequential logic: event control expression may not have more than one edge

Potential Issues

A violation is reported A register which is updated on both edges of a clock in a given sequential block leads to ambiguous synthesis results.

Consequences of Not Fixing

Following are the consequences of not fixing the violation:

Lint Tags

- The synthesis tool can get confused about which edge to use for updating the register.
- RTL and gate-level simulation results may not match.

How to Debug and Fix

To debug the violation, double-click the message. The HDL Viewer window highlights the line in which multiple event control expressions are used in a condition. Scroll down the HDL Viewer window to search for the `always` block. To confirm the implicit sequential logic where control expressions do not have more than one edge, perform one of the following steps:

- Inspect the block visually
- Scroll up the HDL Viewer window to search for the `always` construct
- Search the HDL for the edges of clock used in the `if` statement

To resolve the violation, break the sequential logic into multiple sequential logic blocks, each of which can independently meet the requirement.

Example Code and/or Schematic

In the following example code, VC SpyGlass Lint reports a violation as the event control expressions use multiple edges:

```
always
begin
    @(posedge a or negedge a) out1 <= in1;
    @(negedge a) out2 <= in1;
end
endmodule
```

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

None

BlockHeader

Identifies ports and generics in the block statement header which are not synthesizable

When to Use

Use this Tag to identify ports and generics in the block statement header which are not synthesizable by some synthesis tools.

Description

The *BlockHeader* Tag reports violations for ports and generics used in the block header statements.

Default Weight

10

Language

VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for a port or a generic in a block statement which is not synthesizable by a synthesis tool:

[WARNING] PORT/GENERIC in the block statement header may be unsynthesizable

Potential Issues

A violation is reported when ports or generics are used in the block statement.

Consequences of Not Fixing

Ports and generics used in a block statement are not synthesizable by some synthesis tools.

How to Debug and Fix

To debug the violation, double-click the message. The HDL Viewer window highlights the beginning of the block in which a port, a generic, or both are defined. The block contains the definition of the port, the generic, or both. If the block is too long, view the HDL in an editor and search forward for the port or generic definition.

No fix is required in testbenches or simulation models. Avoid this style in the code targeted for synthesis. If the register depends on both edges of the clock, describe the sequential nature separately and use the combinational logic to generate the final output.

Example Code and/or Schematic

In the following example code, the Tag reports a violation for the non-synthesizable port used in a block statement:

```
entity entBlockHeader1 is
  port(in1      : in bit;
        in2      : in bit;
        pcarry   : in bit;
        sum      : out bit;
        scarry   : out bit);
end entBlockHeader1;
architecture behavioral of entBlockHeader1 is
begin
  b1 : block \\port may be unsynthesizable
    port (a, b, c : in bit;
          d, e : out bit
        );
    port map (a=>in1, b=>in2, c=>pcarry, d=>sum, e=>scarry);
    begin
      d <= (a xor b xor c) ;
      e <= ((a and b) or (a and c) or (b and c));
    end block;
  process (in1, in2)
  begin
    sum <= in1 ;
    sum <= in2 ;
  end process ;
```

```
end behavioral;
```

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

bothedges

Identifies the variable whose both the edges are used in an event control list

When to Use

Use this Tag to identify the variable which is not synthesizable by some synthesis tools.

Description

The *bothedges* Tag reports a violation for a variable whose both the edges are used in a sensitivity list, hence making the variable not synthesizable by some synthesis tools.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for an event control list that contains both the edges of the `<var-name>` variable:

[ERROR] Both edges of the same variable (`<var-name>`) are not allowed in the event control list

Potential Issues

A violation is reported when both edges of the same variable are used in an event control list.

Consequences of Not Fixing

Synthesis tools do not allow both edges of the same variable in an event

control list.

How to Debug and Fix

To debug the violation, double-click the message. The HDL Viewer window highlights the sensitivity list of the `always` block. Check the sensitivity list. It displays both the `posedge` and the `negedge` edges of the signal mentioned in the message.

To resolve the violation, replicate the block, one switching on the positive edge and the other on the negative edge.

Example Code and/or Schematic

In the following example code, VC SpyGlass Lint generates a violation as both edges of the same variable are being used in a control list, hence making the variable non-synthesizable:

```
module test(q);  
  
    output q;  
    reg q,d,reset;  
  
    always @(posedge reset or negedge reset)  
    begin  
        if (reset != 0)  
            q = d;  
    end  
endmodule
```

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

None

BothPhase

Identifies the processes that are driven by both the edges of a clock

When to Use

Use this Tag to identify the processes that are driven by both the edges of a clock.

Description

The *BothPhase* Tag reports violations for processes that are driven by both the edges of a clock.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed at the first line of the `<process-name>` process that is driven by both the edges of the `<clk-name>` clock:

[WARNING] Clock '`<clk-name>`' driving a process ('`<process-name>`') on two different edges may not be synthesizable

Potential Issues

A violation is reported when a process is driven by both the edges of a clock.

Consequences of Not Fixing

The processes that are driven by both the edges of a clock are not

synthesizable by some synthesis tools.

How to Debug and Fix

To debug the violation, double-click the message. The HDL Viewer window highlights the process, which is driven by both the edges of a clock.

No fix is required in testbenches or simulation models. In the code targeted for synthesis, split it into two processes, one driven on the positive edge and the other on the negative edge.

Example Code and/or Schematic

The Tag checks for the following type of clock declarations which use both the edges:

Sample Code 1

```
if( clk'event and clk = '1' ) then --positive edge
    q1 <= data;
end if;
if( clk'event and clk = '0' ) then --negative edge
    q2 <= data;
end if;
```

Sample Code 2

```
if( not clk'stable and clk='1' ) then --positive edge
    q1 <= data;
end if;
if( not clk'stable and clk='0' ) then --negative edge
    q2 <= data;
end if;
```

Sample Code 3

```
wait until clk='1'; --positive edge
q1 <= data;
wait until clk='0'; --negative edge
q2 <= data;
```

Sample Code 4

Lint Tags

```
wait until clk'event and clk='1';  --positive edge
q1 <= data;
wait until clk'event and clk='0';  --negative edge
q2 <= data;
```

Sample Code 5

```
wait until rising_edge(clk); --positive edge
q1 <= data;
wait until falling_edge(clk); --negative edge
q2 <= data;
```

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

ClockStyle

A clocking style is used which may not be synthesizable

Language

VHDL

Description

The ClockStyle Tag flags un-synthesizable clocking styles used in the design.

The following table lists the clocking styles may not be synthesizable by some synthesis tools:

TABLE 1 Un-synthesizable Clocking Styles

<code>if (clk'event) then</code>
<code>elsif(clk'event) then</code>
<code>if(not clk'stable) then</code>
<code>elsif(not clk'stable) then</code>
<code>if (clk'stable) then</code>
<code>elsif(clk'stable) then</code>
<code>if (clk'stable and clk = '1') then</code>
<code>elsif(clk'stable and clk = '1') then</code>
<code>if (clk'stable and clk = '0') then</code>
<code>elsif(clk'stable and clk = '0') then</code>
<code>wait until clk'event;</code>
<code>wait until not clk'stable;</code>
<code>wait until clk'stable;</code>
<code>wait until clk'stable and clk = '1';</code>
<code>wait until clk'stable and clk = '0';</code>
<code>if (not clk'event and clk = '0') then</code>
<code>if (clk'event and not clk'stable) then</code>
<code>if (clk'event and clk'stable)</code>

TABLE 1 Un-synthesizable Clocking Styles (Continued)

<code>wait until not clk'event and clk = '0';</code>
<code>wait until clk'event and not clk'stable;</code>
<code>wait until ((clk'event and clk = '1') xor (cond = 1))</code>
<code>if ((clk'event and clk'stable) and clk= '0') then</code>
If clock is being used as multi-bit signal. For example:
<code>if (clk'event and clk = "00") then)</code>

The ClockStyle Tag checks for clocking styles based on IF, ELSIF, and WAIT constructs only.

NOTE: *Gate between clock expression and other conditions should not be other than the 'and'.*

Such descriptions may not be synthesizable by some synthesis tools.

Message Details

The following message appears at the location where an un-synthesizable clocking style is encountered:

This clocking style may not be synthesizable

Severity

Warning

Suggested Fix

No fix is required in testbenches or simulation models. In code targeted for synthesis, recode to meet synthesizability guidelines.

DisconnSpec

Identifies the disconnection specification constructs which are not synthesizable

When to Use

Use this Tag to identify the disconnection specification constructs which are not synthesizable by some synthesis tools.

Description

The *DisconnSpec* Tag reports violations for the disconnection specification constructs which are not synthesizable by some synthesis tools.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for a disconnection specification construct which is not synthesizable by a synthesis tool:

[warning] Disconnection specification may not be synthesizable

Potential Issues

A violation is reported when a disconnection specification construct is used in the design.

Consequences of Not Fixing

The disconnect specification is used to model the disable time for a signal driver in a guarded assignment. Hence, it is ignored during synthesis.

How to Debug and Fix

To debug the violation, double-click the message. The HDL Viewer window highlights the disconnection specification construct used in the design.

No fix is required in testbenches or simulation models. Avoid this style in the code targeted for synthesis.

Example Code and/or Schematic

Consider the following example:

```
library IEEE;
use ieee.std_logic_1164.all;
entity e is
end e;

architecture arc of e is
  signal a : std_logic register;
  signal b : std_logic bus;
  disconnect a :std_logic after 50 ns;
  disconnect b :std_logic after 20 ns;
begin
process
begin
if(b = '0') then
a <= '1' after 8 ns;
else
a <= null after 10 ns;
end if;
wait on b;
end process;
end arc;
```

In the above example, the DisconnSpec Tag reports a violation as the disconnection specification constructs are used in the design.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

EntityStmt

Statements in entity block may be ignored by some synthesis tools

Language

VHDL

Description

The EntityStmt Tag flags statements in entity declarations.

Statement part of entity declarations is ignored by some synthesis tools.

NOTE: *Synopsys recommends use of `translate_off` and `translate_on` directives against `synthesis_off` and `synthesis_on` directives. If `synthesis_off` and `synthesis_on` directives are used within an expression, they have the potential to create incorrect logic by synopsys tools.*

Message Details

The following message appears at the location where a statement is encountered in an entity declaration:

```
Statement inside an entity may be ignored by some synthesis  
tools
```

Severity

Warning

Suggested Fix

Avoid this style in synthesizable code or enclose these statements in `synthesis_off` and `synthesis_on` pragmas.

ForLoopWait

Identifies the WAIT statements used in FOR-loop constructs which are not synthesizable

When to Use

Use this Tag to identify the WAIT statements which are not synthesizable by some synthesis tools.

Description

The *ForLoopWait* Tag reports violations for the WAIT statements used in FOR-loop constructs.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for a WAIT statement in a FOR-loop construct which is not synthesizable by a synthesis tool:

[WARNING] WAIT statement inside a FOR-loop statement may be unsynthesizable

Potential Issues

A violation is reported when WAIT statement is used inside a FOR-loop construct.

Consequences of Not Fixing

The WAIT statements that are used inside a FOR-loop are not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location of the for loop which has offending wait statement inside the for loop.

If the for loop is small, you can scroll down to see the usage of the `wait` statement. If the for loop is too long you might view the HDL in an editor and search forward for wait statement inside the for loop to find the actual location of the `wait` statement. This is un-synthesizable by several commercially available synthesis tools.

No fix is required in testbenches or simulation models. Put event control outside loops in the code targeted for synthesis.

Example Code and/or Schematic

Consider the following example:

```
entity test is
end test;

architecture test of test is
    signal sig1 : bit;
begin

    p1: process
    begin
        for i in 0 to 10 loop
            wait on sig1;
        end loop;
    end process p1;

end test;
```

In the above example, the ForLoopWait Tag reports a violation as a `wait` statement is used inside the `for` loop.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

IncompleteType

Identifies the incomplete type declarations which are not synthesizable

When to Use

Use this Tag to identify the incomplete type declarations which are not synthesizable by some synthesis tools.

Description

The *IncompleteType* Tag reports violations for the incomplete type declarations which are not synthesizable by some synthesis tools.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for an incomplete type declaration which is not synthesizable by a synthesis tool:

[WARNING] Incomplete type declaration may not be synthesizable

Potential Issues

A violation is reported when an incomplete type declaration is used in a design.

Consequences of Not Fixing

Incomplete type declarations are not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the offending incomplete type is declared. This is un-synthesizable by several commercially available synthesis tools.

No fix is required in testbenches or simulation models. Avoid this style in the code targeted for synthesis.

Example Code and/or Schematic

Consider the following example:

```
entity e is
end e;

architecture arc of e is
type COMP;
type NET;

type COMP_PTR is access COMP;
type NET_PTR is access NET;

constant MODMAX:integer:= 100;
constant NETMAX: integer := 2500;

type COMP_LIST is array(1 to MODMAX) of COMP_PTR;
type NET_LIST is array(1 to NETMAX) of NET_PTR;

type COMPLIST_PTR is access COMP_LIST;
type NETLIST_PTR is access NET_LIST;

type COMP is
  record
    COMP_NAME : STRING(1 to 10);
    NETS      : NETLIST_PTR;
  end record;

type NET is
  record
    NET_NAME : STRING(1 to 10);
    COMPONENTS : COMPLIST_PTR;
```

Lint Tags

```
end record;
```

```
begin  
end arc;
```

In the above example, the `IncompleteType` Tag reports a violation as an incomplete type declaration is used in the design.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

infinitemloop

While/forever loop has no break control

Language

Verilog

Description

The infinitemloop Tag flags `while` or `forever` loops without event control to break the loop.

Such descriptions may not be synthesizable by some synthesis tools.

NOTE: *The offending construct is now synthesizable by almost all standard synthesis tools. Therefore, the infinitemloop Tag has been switched off and will be removed in the next release of VC SpyGlass Lint.*

NOTE: *The infinitemloop Tag is switched off and will be removed in a future release. Use the VC SpyGlass Lint SYNTH_5230 Built-In Tag that reports the while unrolling loops.*

Message Details

The following message appears at the location of a `while` or `forever` loop's terminating condition when the condition has no event control to break the loop:

`while/forever loop has no event control to break loop`

Tag Severity

Error

Suggested Fix

Fix may not be necessary if this is expected to be free-running, but you should check each such case.

Examples

In the following example, the terminating condition of the `while` loop is always false, resulting in an infinite loop:

```
`define cond 1'b1
```

Lint Tags

```
module test(in1, in2, sel, out1);
  input in1, in2, sel;
  output out1;
  reg out1;

  always@(sel or in1)
    while(`cond)
      case(sel)
        1'b0: out1 <= in1;
        1'b1: out1 <= in2;
      endcase
endmodule
```

For this example, VC SpyGlass Lint generates the `infinitemloop` Tag message.

InitPorts

Default initial value of in/out/inout port may be ignored by some synthesis tools

Language

VHDL

Description

The InitPorts Tag flags default initial value settings for input, output, and inout ports.

Such descriptions may not be synthesizable by some synthesis tools.

NOTE: *Synopsys recommends use of `translate_off/translate_on` directives against `synthesis_off/synthesis_on` directives. If `synthesis_off/synthesis_on` directives are used within an expression, they have the potential to create incorrect logic by synopsys tools.*

Message Details

The following message appears at the location where an input, output, or inout port `<port-name>` is declared with default initial value:

Default initial value of in/out/inout port '`<port-name>`' may be ignored by some synthesis tools

Severity

Warning

Suggested Fix:

If you want to set initial state of ports, use reset connections. If you need initial values for simulation, enclose those statements in the `synthesis_off/synthesis_on` pragmas. Ideally, try to avoid using such statements in synthesizable code since they may lead you to believe that the logic is functioning correctly when it actually does not reset correctly in the implementation.

IntGeneric

Identifies the non-integer type used in the declaration of a generic which is not synthesizable

When to Use

Use this Tag to identify the non-integer types used in generic declarations which are not synthesizable by some synthesis tools.

Description

The *IntGeneric* Tag reports violations for the non-integer types used in generic declarations which are not synthesizable by some synthesis tools.

Tag Exceptions

Enumeration type generics are not reported by the Tag as the synthesis process supports them.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for the *<type-name>* non-integer type, which is used in the declaration of the *<gen-name>* generic and is not synthesizable by a synthesis tool:

[WARNING] Non-integer type '*<type-name>*' used in declaration of generic '*<gen-name>*' may be unsynthesizable

Potential Issues

A violation is reported when a non-integer type is used in a generic

declaration.

Consequences of Not Fixing

Non-integer types used in a generic declaration may not be synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL window takes you to the location where the offending non-integer generic is declared. This is unsynthesizable by several commercially available synthesis tools.

No fix is required in testbenches or simulation models. Use only integer generics in the code targeted for synthesis.

Example Code and/or Schematic

Consider the following example:

```
entity test is
  generic
    gen1 : real := 10.0);
end test;
```

In the above example, the IntGeneric Tag reports a violation for gen1 as it is a non-integer generic declaration.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

LinkagePort

Identifies the linkage ports which are not synthesizable

When to Use

Use this Tag to identify the linkage ports which are not synthesizable by some synthesis tools.

Description

The *LinkagePort* Tag reports violations for ports which are of the `linkage` type and not synthesizable by some synthesis tools.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for a port of the `linkage` type which is not synthesizable by a synthesis tool:

[WARNING] Linkage port may not be synthesizable

Potential Issues

A violation is reported when a port of `linkage` type is used in the design.

Consequences of Not Fixing

Ports of `linkage` type are not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the offending port of type `linkage` is declared. This is un-

synthesizable by several commercially available synthesis tools.

No fix is required in testbenches or simulation models. Avoid this style in the code targeted for synthesis.

Example Code and/or Schematic

Consider the following example:

```
entity e is
port(
  a:  linkage bit;
  b : out bit
);
end e;
```

In the above example, the LinkagePort Tag reports a violation for the port a, as it is of type linkage.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

LoopBound

Identifies the for loop range bounds that are not locally or globally static

When to Use

Use this Tag to identify the For loop constructs which are not synthesizable by some synthesis tools.

Description

The *LoopBound* Tag reports violations for the `for` loop constructs that have locally or globally non-static bounds and are therefore not synthesizable by some synthesis tools.

Language

VHDL

Default Weight

10

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed at the first line of a FOR-loop construct that has a locally or a globally non-static range bound:

[WARNING] For loop range bounds should either be locally static or globally static

Potential Issues

A violation is reported when a for loop construct has locally or globally non-static bounds.

Consequences of Not Fixing

The synthesis process cannot be deterministic and can possibly hang if the

loop bounds are not static. In such cases, there is also a possibility of hanging of the simulation.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the `for` construct, which has locally or globally non-static bounds.

To resolve the violation, use a static lower and a static upper bound. If this is not possible, implement the loop as a multi-cycle logic.

To avoid hanging of the simulation, check the `for` loops at the RTL coding stage.

Example Code and/or Schematic

Example 1

In the following example code, VC SpyGlass Lint reports a violation as `dataIn` is non-static:

```
signal dataIn : integer ;  
for i in 1 to dataIn loop  
    temp <= temp * i ;  
end loop ;
```

Example 2

In the following example code, no violation is reported by VC SpyGlass Lint:

```
for i in 1 to 3 loop  
    temp <= temp * i ;  
end loop ;
```

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

mixedsenselist

Mixed conditions in sensitivity list may not be synthesizable (Verilog)
Edge and level conditions are mixed in if statement (VHDL)

Language

Verilog, VHDL

Description

Verilog

The mixedsenselist Tag flags mixed edge and non-edge conditions in the sensitivity list of an `always` construct.

Such conditions in sensitivity list may not be synthesizable by some synthesis tools.

VHDL

The mixedsenselist Tag flags if edge and level conditions are specified together in an `if` statement.

Message Details

Verilog

The following message appears at the location of an `always` construct where mixed edge or non-edge condition is used in the sensitivity list:

Mixed conditions in sensitivity list may not be synthesizable

VHDL

The following message appears if edge and level conditions are specified together in an `if` statement:

Edge and level conditions are mixed in 'if' statement

Tag Severity

Error (Verilog) / Warning (VHDL)

Suggested Fix

Decide if you really want to trigger the block on any change in the mixed signal. Check if your requirement can be met by mapping to either a positive-edge change or a negative-edge change. If both are required, consider duplicating the block, one triggering on each edge.

Examples (Verilog)

In the following example, the `always` construct has both an edge specification and a non-edge specification:

```
always @(posedge clock or reset)
    q = d;
```

For this example, VC SpyGlass Lint generates the `mixedsenselist` Tag message.

Examples (VHDL)

In the following example, both edge and level conditions are specified together in the `if` statement:

```
elsif ((clk'event and clk='1') and (d1='1') and
      (d2='0')) then
```

Here, clock edge `clk` is mixed with `d1` and `d2` level checking. Hence, VC SpyGlass Lint generates the `mixedsenselist` Tag message.

MultipleWait

Identifies multiple wait statements having the same clock expression which are not synthesizable

When to Use

Use this Tag to identify multiple `wait` constructs which are not synthesizable by some synthesis tools.

Description

The *MultipleWait* Tag reports violations for multiple `wait` constructs having the same clock expression which are not synthesizable by some synthesis tools.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for a `wait` construct at the `<num>` line when another `wait` construct is found for the same clock expression:

[WARNING] Multiple wait statements (first one found at line no. `<num>`) having same clock expression may not be synthesizable

Potential Issues

A violation is reported when multiple `wait` statements have the same clock expression.

Consequences of Not Fixing

Some synthesis tools support multiple `wait` statements in a process though it is not a generally accepted synthesizability practice. But even then, your formal verification tool may not support the coding style.

How to Debug and Fix

Double-click the violation message. The HDL window, highlights the location of the wait statement which has same clock expression as the previously declared wait statement (line number of the wait statement is also mentioned in the violation) inside the process block. This is un-synthesizable by several commercially available synthesis tools.

No fix is required in testbenches or simulation models. Ensure that your synthesis flow supports this style in the code targeted for synthesis. Also, use a waiver to avoid the violation if your downstream tool supports the coding style.

Example Code and/or Schematic

Consider the following example:

```
begin
  latch_behavior: process is
  begin
    wait until clk='1';
    q1<=d ;
    wait until clk='1';    --VIOLATION
    q2<=d ;
    wait until clk='1';
    q3<=d ;
  end process latch_behavior;
```

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

NoTimeOut

Identifies the timeout expression in a wait statement, which is not synthesizable

When to Use

Use this Tag to identify the timeout expressions which are not synthesizable by some synthesis tools.

Description

The *NoTimeOut* Tag reports violations for the timeout expression in a wait construct, which is not synthesizable by some synthesis tools.

The Tag reports wait constructs which are in the following format:

```
WAIT for <time-expression>;
```

where <time-expression> is the wait time.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for a timeout expression in a wait construct, which is not synthesizable by a synthesis tool:

[WARNING] Timeout expression in wait statement may not be synthesizable

Potential Issues

A violation is reported when a timeout expression is used in a wait

statement.

Consequences of Not Fixing

The timeout expressions in a wait construct are not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the location where the offending timeout expression in wait statement is used. This is un-synthesizable by several commercially available synthesis tools.

No fix is required in testbenches or simulation models. Avoid this style in the code targeted for synthesis.

Example Code and/or Schematic

Consider the following example:

```
entity test is
end test;

architecture test of test is
    signal sig1, sig2 : bit;
begin -- test

    p1: process
        begin -- process p1
            sig1 <= sig2;
            wait for 10 ns;
        end process p1;

end test;
```

In the above example, the NoTimeOut Tag reports a violation as a timeout expression is used inside a wait statement.

Default Severity Label

Warning

Tag Group

Synthesis

Lint Tags

Reports and Related Files

None

PortType

Identifies ports of unconstrained types which are not synthesizable

When to Use

Use this Tag to identify ports which are not synthesizable by some synthesis tools.

Description

The *PortType* Tag reports violations for ports of unconstrained types.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed where the *<port-name>* port of the unconstrained type, which is not synthesizable by a synthesis tool, is declared:

[WARNING] Port *<port-name>* of unconstrained type may not be synthesizable

Potential Issues

A violation is reported when a port of unconstrained type is used in the design.

Consequences of Not Fixing

Ports of unconstrained types are not synthesizable by some synthesis tools

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where unconstrained port is used. This is un-synthesizable by several commercially available synthesis tools.

No fix is required in testbenches or simulation models. Avoid this style in the code targeted for synthesis.

Example Code and/or Schematic

Consider the following example:

```
package testpack is
  type unconsIntArr is array (integer range <>) of integer;
end testpack;

use work.testpack.all;
entity test is
  port (
    p1 : in unconsIntArr);
end test;
```

In the above example, the PortType Tag reports a violation for the port, p1, as it is of unconstrained type.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

PhysicalTypes

Identifies the physical constructs which are not synthesizable

When to Use

Use this Tag to identify the physical constructs which are not synthesizable by some synthesis tools.

Description

The *PhysicalTypes* Tag reports violations for declaration of the Physical, Floating Point, File, or Access type physical constructs which are not synthesizable by some synthesis tools.

Language

VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed where an un-synthesizable physical construct `<construct-name>` is declared:

[Warning] `<construct-name>` declaration may be ignored by some synthesis tools

Potential Issues

A violation is reported when a physical construct is used in a design.

Consequences of Not Fixing

The Physical Type, Floating Point Type, File Type, and Access Type constructs are not synthesizable by most of the synthesis tools.

How to Debug and Fix

To debug the violation, double-click the message. The HDL Viewer window highlights the line where a Physical, Floating, File, or Access type object is

declared. The message specifies the type of the offending object.

No fix is required in testbenches or simulation models. Enclose these statements in the `synthesis_off`, `synthesis_on` pragmas in the code targeted for synthesis.

To create correct logic, Synopsys recommends to using the `translate_off` or the `translate_on` directive instead of the `synthesis_off` or the `synthesis_on` directive.

Example Code and/or Schematic

In the following example code, VC SpyGlass Lint reports a violation for the physical and the access type physical constructs which are non-synthesizable:

```
end ent;
architecture arc of ent is
  type current is range 0 to 1E9
  units
    nA;
  end units;  \\physical type declaration
  type MODULE is
    record
      SIZE      : integer range 20 to 200;
      i         : current;
    end record;
  type PTR is access MODULE; \\access type declaration
begin
  process
    variable MOD1PTR,MOD2PTR:PTR;
  begin
    MOD1PTR := new MODULE;
    MOD2PTR := new MODULE'(25, 10 nA);
  end process;
end arc;
```

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

PreDefAttr

Identifies the pre-defined attributes which are not synthesizable

When to Use

Use this Tag to identify the pre-defined attributes which are not synthesizable by some synthesis tools.

Description

The *PreDefAttr* Tag reports violation for the following pre-defined attributes:

VAL	SUCC	PRED	LEFTOF
RIGHTOF	DELAYED	QUIET	TRANSACTION
ACTIVE	LAST_EVENT	LAST_ACTIVE	

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for a pre-defined attribute *<attr-name>* which is not synthesizable by a synthesis tool:

[WARNING] Use of pre-defined attribute '*<attr-name>*' may not be synthesizable

Potential Issues

A violation is reported when one of the above mentioned pre-defined

attribute is used in the design.

Consequences of Not Fixing

The above mentioned pre-defined attributes are not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the offending pre-defined attribute is used. This is un-synthesizable by several commercially available synthesis tools.

No fix is required in testbenches or simulation models. Avoid these attributes in the code targeted for synthesis.

Example Code and/or Schematic

Consider the following example:

```
entity test is
end test;

architecture test of test is
    signal sig1 : integer;
begin -- test

    sig1 <= integer'VAL(0);

end test;
```

In the above example, the PreDefAttr Tag reports a violation as the predefined attribute, VAL, is used in the design.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

readclock

Unsynthesizable implicit sequential logic: clock read inside always block.

Language

Verilog

Description

The readclock Tag flags sequential descriptions where the clock signal is read inside the always construct.

Message Details

The following message appears at the location where the clock signal `<clk-name>` is read in an always construct:

```
unsynthesizable implicit sequential logic: cannot read clock
'<clk-name>' inside always block
```

Tag Severity

Warning

Suggested Fix

First determine if you really need to test the value. The fact that the block has already triggered on positive edge implies the clock value, unless you are looking for simulation X states. In that case, consider putting the clock test inside `translate_off` and `translate_on` pragmas so the construct will be ignored in synthesis.

Examples

In the following example, the clock signal `clk` is read inside the always construct:

```
always@ (posedge clk)
  if(clk == 1'b1)
    out1 <= in1 & in2;
```

For this example, VC SpyGlass Lint generates the readclock Tag message.

ResFunction

Identifies the resolution functions which are not synthesizable

When to Use

Use this Tag to identify the resolution functions which are not synthesizable by some synthesis tools.

Description

The *ResFunction* Tag reports violations for the resolution functions which are not synthesizable by some synthesis tools.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for the *<func-name>* resolution function which is not synthesizable by a synthesis tool:

[WARNING] Resolution function *<func-name>* may not be synthesizable

Potential Issues

A violation is reported when a resolution function is used in the design.

Consequences of Not Fixing

Resolution functions are not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the

location where the offending resolution function is used. This is unsynthesizable by several commercially available synthesis tools.

To resolve the violation, use the `std_logic` type instead of a resolution function.

Example Code and/or Schematic

Consider the following example:

```
entity test is
end test;

architecture test of test is

    function res_func (
        inputs : bit_vector)
        return bit is
    begin
        return '0';
    end res_func;

    signal sig1 : res_func bit;
begin
end test;
```

In the above example, the ResFunction Tag reports a violation as the resolution function is used in the design.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

SigVarInit

Identifies the initial values of signals and variables which are not synthesizable

When to Use

Use this Tag to identify the initial values which are not synthesizable by some synthesis tools.

Description

The *SigVarInit* Tag reports violations for the assignment of initial values to signals and variables, which are not synthesizable by some synthesis tools.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for the *<name>* signal or the *<name>* variable whose initial value is not synthesizable by a synthesis tool:

[WARNING] Default initial value of '*<name>*' may be ignored by some synthesis tools

Potential Issues

A violation is reported when the initial value is assigned to a signal or variable.

Consequences of Not Fixing

Assignment of initial values to signals and variables is not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the signals or variables have been declared with some initial values. This is un-synthesizable by several commercially available synthesis tools.

If you want to set an initial state, setup the reset connections. If you need initial values for the simulation, enclose such initial value statements in the `synthesis_off/synthesis_on` pragmas. Try to avoid the statements in the synthesizable code since they make you feel that the logic is functioning correctly, whereas in the implementation the logic does not function correctly.

Example Code and/or Schematic

Consider the following example:

```
architecture test of test is
    signal sig1 : integer;
begin

    p1: process (sig1)
        variable var1 : integer := 0;  --VIOLATION

    begin
        end process p1;
    end test;
```

In the above example, the SigVarInt Tag reports a violation as the variable, `var1`, is assigned an initial value.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

SynthIfStmt

Identifies the IF statements which are not synthesizable

When to Use

Use this Tag to identify the IF statements which are not synthesizable by some synthesis tools.

Description

The *SynthIfStmt* Tag reports violations for the `if`, `if-elsif`, or `if-elsif-else` constructs which are not synthesizable by some synthesis tools.

The Tag reports the following constructs, where:

- the `if` statement uses one of the following non-synthesizable attributes:

ASCENDING	IMAGE	VALUE
POS	SUCC	PRED
LEFTOF	RIGHTOF	DELAYED
QUIET	TRANSACTION	LAST_EVENT
LAST_ACTIVE	LAST_VALUE	DRIVING
DRIVING_VALUE	SIMPLE_NAME	INSTANCE_NAME
PATH_NAME		

- an asynchronous or synchronous process is followed by an `else` or an `elsif` statement

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed at the first line of an `if`, `if-elsif`, or `if-elsif-else` construct which is not synthesizable by a synthesis tool:

[WARNING] The IF-statement does not conform with any synthesizable description style

Potential Issues

A violation is reported when an IF statement does not conform with any synthesizable description style.

Consequences of Not Fixing

Such `If` and `If-else-If` statements are not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the offending `if`, `if-elsif` or `if-elsif-else` constructs are used. This is un-synthesizable by several commercially available synthesis tools.

No fix is required in testbenches or simulation models. Recode the code targeted for synthesis to meet the synthesizability guidelines.

Example Code and/or Schematic

Example 1

Consider the following example:

```
PROC_1: PROCESS(clk, rst)
begin
    if(rst = '0') then q <= '0';
        if(clk'event and clk = '1') then q <= d;
            else q <= '1';
        end if;
    end process;
```

In the above example, VC SpyGlass Lint reports a violation as the process

is followed by the `else` statement.

Example 2

Consider the following example:

```
PROC_2: PROCESS(clk, rst)
begin
    if(clk'event and clk = '1') then
        if(rst = '0') then q <= '0';
        else q <= d;
        end if;
        else q <= '1';
    end if;
end process;
```

In the above example, VC SpyGlass Lint reports a violation as the process is followed by the `else` statement.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

UserDefAttr

Identifies the user-defined attributes which are not synthesizable

When to Use

Use this Tag to identify the user-defined attributes which are not synthesizable by some synthesis tools.

Description

The *UserDefAttr* Tag reports violation for the user-defined attributes which are not synthesizable by some synthesis tools.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for a user-defined attribute:

[WARNING] Use of user-defined attribute may not be synthesizable

Potential Issues

A violation is reported when a user-defined attribute is used in the design.

Consequences of Not Fixing

User-defined attributes are not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where offending user-defined attribute is used. This is un-synthesizable by

several commercially available synthesis tools.

No fix is required in testbenches or simulation models. Avoid this style in the code targeted for synthesis.

Example Code and/or Schematic

Consider the following example:

```
architecture arc of nand_gate is
  component nand_comp
    port(in1,in2: in bit; out1 : out bit);
  end component;

  type DOUBLE_INT is
    record
      X,Y : INTEGER;
    end record;

  attribute PLACEMENT: DOUBLE_INT;
  attribute SIZE:DOUBLE_INT;
  attribute PLACEMENT of N1: label is (50,45);
  attribute SIZE of N1:label is (2,4);
  signal PERIMETER:INTEGER;
  signal A,B,Z:bit;
begin
  N1: NAND_COMP port map(A,B,Z);
  PERIMETER <= 2 * (N1'SIZE.X + N1'SIZE.Y);
end arc;
```

In the above example, the UserDefAttr Tag reports a violation as a user-defined attribute, PERIMETER, is used in the design.

Default Severity Label

Warning

Tag Group

Synthesis

Lint Tags

Reports and Related Files

None

W107

Do not make bus connections to primitive gates (and, or, xor, nand, nor, xnor)

Language

Verilog

Description

The W107 Tag flags bus connections to primitive gates.

Verilog does not provide a method to check that a connection made to the inputs of a primitive gate (and, or, etc.) is of an appropriate width.

If a wider bus is attached, the function simply expands to include the additional bits. This expansion can lead to unexpected behavior if the bus width changes as the design evolves. Thus it is safer to explicitly connect, bit-by-bit, those bits, which should be gated.

Message Details

The following message appears at the location where connection of a bus *<net-name>* to a primitive gate is encountered:

Do not make multibit bus (<net-name>) connections to gate ports. [Hierarchy: '<hier-path>']

Where, *<hier-path>* is the complete hierarchical name of the containing scope excluding subprograms.

Tag Severity

Warning

Suggested Fix

Make connections bit-by-bit. For example, use `and (b [0] , b [1] , b [2] . . )` rather than `and (b [0 : 3] )`.

Examples

Consider the following example that has multi-bit ports connected to primitive gates:

Lint Tags

```
module test(in1, in2, in3, in4, in5, in6,  
            out1, out2, out3);  
  input [3:0] in1, in2, in3, in4, in5, in6;  
  output [3:0] out1, out2, out3;  
  
  and(out1, in1, in2);  
  or(out2, in3, in4);  
  xor(out3, in5, in6);  
endmodule
```

VC SpyGlass Lint generates the W107 Tag message for each multi-bit connection (input and output) to primitive gates. Thus, nine messages are generated for the above example.

W110

Identifies a module instance port connection that has incompatible width as compared to the port definition

When to Use

Use this Tag to identify the width mismatch between a module port and the net connected to the port of that module instance.

Description

The *W110* Tag reports violations for the width mismatch between a module port and the net connected to the port of that module instance.

NOTE: *The W110 Tag supports generate-if, generate-for, and generate-case blocks.*

Calculation of width

The *W110* Tag calculates the width of port expressions on the basis of the following conditions:

- When the [nocheckoverflow](#) parameter is set to `yes` or *W110*, the width is calculated as per LRM. For the constants, natural width is considered.
- When the [nocheckoverflow](#) parameter is set to `no`, the width is calculated according to following methods:
- For the plus or the minus operator, the value based width is considered.
- For the multiplication operator, the width is calculated as the sum of operand widths.
- For the division operator, the LHS width is considered.
- For the concat operator, the width is calculated as the sum of all operands.

Language

Verilog

Default Weight

10

Parameter(s)

- [*nocheckoverflow*](#): Default value is `no`. Set this parameter to `yes` or Tag name to check the bit-width as per LRM. Other possible value is the Tag name.
- [*report_blackbox_inst*](#): Default value is `no`. In this case, the Tag does not report violation for port width mismatch for black box instances. Set the value of the parameter to `yes` to report violation for port width mismatch for black box instances.
- [*use_lrm_width*](#): Default value is `no`. Set this parameter to `yes` to consider the LRM width of integer constants, which is 32 bits.
- [*handle_shift_op*](#): Default value is `no`. In this case, no violation is reported if the shifted or non-shifted width of a shift expression (at the port connection of a module instance) matches the width of the corresponding module port definition. But the Tag does not calculate the shifted width, if the RHS of the shift expression is non-static. Set this parameter to `shift_left`, `shift_right`, `shift_both`, `no_shift`, or comma separated list of Tag names, to compare shifted or non shifted widths for left and right shift expressions.
- [*handle_zero_padding*](#): Default value is `no`. Set the value of the parameter to `yes` to perform leading zero expansion and truncation of RHS of an assignment. This is performed only if the RHS of the assignment is static.

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for the `<inst-name>` module instance when there is a bit-width mismatch between a net of `<width2>` bits, which is connected to the `<port-name>` port of `<width1>` bits:

```
[WARNING] Incompatible width for port '<port-name>'(width
<width1>) on instance '<inst-name>'(actual width <width2>)
[Hierarchy: '<hier-path>']
```

Where, `<hier-path>` is the complete hierarchical path of the containing process.

Potential Issues

Width mismatch between module port and net connected to port or when the wrong signal is connected to the terminal.

Consequences of Not Fixing

A mismatched width connection causes the following results:

- Connection of a bus which is wider than the port: Excess bits are ignored for the input bus and floated for the output bus.
- Connection of a bus which is narrower than the port: Missing bits are driven unknown for the input bus and ignored for the output bus.

How to Debug and Fix

To debug the violation, double-click the message. The HDL Viewer window highlights the module or the interface instance where a port connection has incompatible width as compared to the port definition.

The violation message shows the following information:

- Name of the port which has incompatible width in its port connection
- Name of the Module/Interface
- Name of the Module/Interface Instance
- Width of the port in the Module/Interface definition
- Width of the Connected port expression

To resolve the violation, ensure the connection of the correct number of bits across the hierarchy boundary, then explicitly ignore the bits that are not needed.

Example Code and/or Schematic

Example 1

Consider the following example:

```
inst IN1 (.in1(a[2:0]+b[2:0]));  
  
// Width of expression, a[2:0]+b[2:0], will be 4 (7+7=14, 4  
bits wide)
```

Example 2

Consider the following example:

```

inst IN1 (.in1(a[2:0]*b[2:0]));
// Width of expression, a[2:0]*b[2:0], will be 6
inst IN2 (.in1(a1[2:0]*b1[2]));
// Width of expression, a[2:0]*b1[2], will be 3

```

Example 3

Consider the following example:

```

inst IN1 (.in1(a[2:0]/b[2]));
// Width of expression, a[2:0]/b[2], will be 3

```

Example 4

Consider the following example:

```

inst IN1 (.in1({1'b0,a[2:0]}));
// Width is 3 bits (leading 0 ignored)
inst IN2 (.in1({1'b1,a[2:0]}));
// Width is 4 bits (1+3 bits)

```

Example 5

Consider the following example code:

```

module top(in1, in2, out1, out2, out3);
    input in1, in2;
    output out1, out2;
    output [1:0] out3;

    lower L1 (.i1(in1),.i2(in2),.o1(out1),.o2(out2));
    OR2 rtlc_I4 (.Z(out3), .A(in1), .B(in2));
endmodule

module lower(i1, i2, o1, o2);
    input [2:0] i1, i2;
    output o1, o2;

    reg o1, o2;

    always@(i1 or i2)
    begin
        o1 <= ~i1;
    end

```

```
        o2 <= &i2;  
    end  
endmodule
```

In the instance `L1` of module `lower`, there is a bit-width mismatch between ports `i1`, `i2` (3 bits wide) and the connected nets `in1`, `in2` (single-bit). Therefore, VC SpyGlass Lint generates the following messages:

Incompatible width for port 'i1'(width 3) on instance 'top.L1'(actual width 1) [Hierarchy: 'top']

Incompatible width for port 'i2'(width 3) on instance 'top.L1'(actual width 1)[Hierarchy: 'top']

In addition, in the instance `rtl_c_14` of ASIC cell `OR2`, there is a bit-width mismatch between port `Z` (2-bits) and the connected net `out3` (single-bit). Therefore, VC SpyGlass Lint generates the following message:

Incompatible width for port 'Z'(width 1) on instance 'top.rtlc_I4'(actual width 2)[Hierarchy: 'top']

Default Severity Label

Warning

Tag Group

Instance

Reports and Related Files

None

W110a

Use same port index bounds in component instantiation and entity declaration.

Language

VHDL

Description

The *W110a* Tag checks the mismatch, if any, in between the port index bound of component instantiation and entity declaration.

In VHDL, if there is any port width mismatch during component instantiation, it is a syntax error.

However, if a port is declared with the integer range and the range is configured through generic, it is not a syntax error. In such a case, there are chances that port range may overflow or underflow. Therefore, the purpose of this Tag is to identify situations where the port index during component instantiation does not match with the entity declaration.

The Tag also supports the following cases:

- Typed/sub-typed and aliased port.
- Positive and natural range data type, apart from integer ports.

Message Details

The *W110a* Tag reports the following violation message at component instantiation:

Incompatible index for port '<port_name>' (Range: '<port_range>') in component instantiation '<instance_name>' port '<en_port_name>' (Range: '<en_port_range>') [Hierarchy: '<hier-path>']

Arguments

- Component instance port name, <port_name>
- Component instance port range, <port_range>
- Component instance name, <instance_name>
- Entity port name, <en_port_name>
- Entity port range, <en_port_range>

- Hierarchical path of the design unit where the component is instantiated, <hier_path>

Tag Severity

Warning

Examples

Syntax Errors

The following examples describes the cases in which there is a port width mismatch during component instantiation and the *W110a* Tag considers this as syntax error:

Example 1

Consider the following example:

```
ain1 : in std_logic_vector (0 to 2)
ain  : in std_logic_vector (0 to 7)
```

Example 2

Consider the following example:

```
ain1 : in integer range (-16) to (15)
ain  : in integer range (-8) to (7)
```

In both the **Example 1** and **Example 2**, component instantiation signal `in` is assigned with `ain`. Since there is a port width mismatch for the above examples and the range is not configured through generic, the Tag considers this as a syntax error. However, if the range is configured through generic, the Tag reports a warning message in this case.

Violating Cases

The following examples describe the cases where the Tag reports a warning message because there is a port width mismatch and the range is configured through generic:

Example 1

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
```


Lint Tags

```

entity mult is
  generic (
    A: integer :=-1;
    B: integer :=1
  );
  port (ain : in integer range (A) to (B);
        z: out integer range (A) to (B)
      );
end mult;

architecture rtl of mult is
begin
  process(ain) begin
    z <= ain ;
  end process;
end rtl;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity top is
  port (ain1 : in integer range (-16) to (15);
        cout1: out integer range -(16) to (15)
      );
end top;

architecture structure of top is
  component mult
  generic (
    A: integer :=-1;
    B: integer :=1
  );
  port (ain : in integer range (A) to (B);
        z: out integer range (A) to (B)
      );
end component;

```

```
-- begin
begin
  ul: mult
    generic map(
      A=> -8,
      B=> 7
    )
    port map(ain=>ain1,z=>cout1);
end structure;
```

Example 2

Consider the following example:

```
ain1 : in integer range (-8) to (7)
ain  : in integer range (-16) to (15)
```

In the above example, the component instantiation signal `in` is assigned with `ain`.

Example 3

Consider the following example:

```
ain1 : in integer range (2) downto (0)
ain  : in integer range (15) downto (13)
```

In the above example, the component instantiation signal `in` is assigned with `ain`.

Non-Violating Cases

The following examples describe the cases where the Tag does not report any violation because there is no port width mismatch:

Example 1

Consider the following example:

```
ain1 : in integer range (0) to (4)
ain  : in integer range (4) downto (0)
```

Example 2

Consider the following example:

```
ain1 : in std_logic_vector (4 downto 0)
ain  : in std_logic_vector (0 to 4)
```

Example 3

Consider the following example:

```
ain1 : in integer range (2) downto (0)
```

```
ain : in integer range (3) downto (1)
```

In **Example 1**, **Example 2**, and **Example 3**, the component instantiation signal `in` is assigned with `ain`.

W111

Not all elements of an array are read

Language

Verilog, VHDL

Description

The W111 Tag reports a violation for arrays where all elements of the array are not read.

The W111 Tag checks for input/inout ports, internal signals, and variables.

An array with unused elements indicates that the model does not fully exercise all paths out of the array and thus may mask possible errors in index or bit-slice selection.

NOTE: *For Verilog, inside the nested `for` loops, signals of user-defined type like structures, interfaces, etc. are considered fully set if used on the left-hand side of an expression and fully read if used on the right-hand side of an expression.*

By default, the W111 Tag reports violation for array declared as input/inout port, internal signal or variable but not read. Set the value of the [ignore_in_ports](#) parameter to `yes` to ignore input and inout ports for Tag checking.

By default, this Tag does not process large arrays. Set the [handle_large_bus](#) parameter to `yes` to process large arrays (greater than 50,000) and report violation if not used correctly.

NOTE: *The W111 Tag only checks static indexes.*

Message Details

Verilog

Message 1

The following message appears at the location of an array `<array-name>` in hierarchy `<hier-path>` that is not read completely:

Not all elements of array '`<array-name>`' are 'read' [Hierarchy: '`<hier-path>`']

Where, `<array-type>` can be `signal`, `port`, or `variable`. The

<hier-path> is the complete hierarchical name of the containing scope excluding functions/tasks.

Message 2

The following message is displayed when the signal size is greater than 50,000 and the *handle_large_bus* parameter is disabled:

```
signal '<signal-name>' size too big thus not processed, use
'configure_lint_tag_parameter -tag w111 -parameter
HANDLE_LARGE_BUS -value yes' for enabling handling of these
signals
```

If the `HANDLE_LARGE_BUS` parameter is not enabled, the violation for signals of size greater than 50,000 is missed.

VHDL

The following message appears at the location of an array *<arrayHANDLE_LARGE_BUSy-name>* of type *<array-type>* in hierarchy *<hier-path>* that is not read completely. The message also displays the bits which are not read in the array:

```
Not all the elements of <array-type> '<array-name>'
'(<offending-bits>)' are read. [Hierarchy: '<hier-path>']
```

Where, *<array-type>* can be signal, port, or variable. The *<hier-path>* is the complete hierarchical name of the containing scope excluding subprograms.

Severity

Warning

Suggested Fix

Consider this Tag primarily informational.

Verilog Examples

Consider the following example:

```
module top (out,in);
    output [1:0] out;
```

```

        input [2:0] in;

        assign out=in[1:0];

    endmodule

```

In the above example, the *W111* Tag reports a violation because all bits of the array `in` are not read. The following message is reported by this example:

```

Not all elements of array 'in[2]' are 'read' [Hierarchy:
':top']

```

VHDL Examples

Consider the following example where all bits of array `a` are not read in architecture `arc_e_1` of entity `e_1`:

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity e_1 is
    port(a: in bit_vector(4 downto 0);
         b: out bit_vector(3 downto 0));
end e_1;

architecture arc_e_1 of e_1 is
    begin
        b <= a(3 downto 0);
    end arc_e_1;

```

In the above example, the *W111* Tag reports a violation as all bits of array `a` are not read in the architecture `arc_e_1` of entity `e_1`. The following message is reported by this example:

```

Not all the elements of in port 'a' (Bits: 4) are read. [Hierarchy:
':e_1(arc_e_1)']

```

W116

Identifies the unequal length operands in the bit-wise logical, arithmetic, and ternary operators

When to Use

Use this Tag to identify bit-width mismatch between operands of the bit-wise logical, arithmetic, and ternary operators.

Description

Verilog

The W116 Tag flags bit-width mismatch between operands of bit-wise logical, arithmetic, or ternary operators.

The W116 tag does not report a violation when two static expressions are present in both the LHS and the RHS.

Following is the list of operators covered under the W116 Tag:

Arithmetic Operators	
Subtraction (-)	Addition (+)
Multiplication (*)	Division (/)
Modulus (%)	
Bit-wise Operators	
bit-wise xor (^)	bit-wise negation (~)
bit-wise and (&)	bit-wise or ()
Ternary Operator (? :)	

NOTE: *If the count of operators is more than 500 in an expression, then the expression is ignored for Tag checking.*

Width Calculation for Verilog

If you set the value of the [nocheckoverflow](#) parameter to yes or W116, the W116 Tag checks the bit-width as per the LRM, as shown in the following table:

Operator	LRM width	Normal width
+, -	Max (LHS width, RHS width)	Max (LHS width, RHS width)
*	Max (LHS width, RHS width)	LHS width + RHS width
/	Max (LHS width, RHS width)	LHS width
%	Max (LHS width, RHS width)	RHS (non-static) - RHS width RHS (static) - width of (RHS Value -1)

For constants, the natural width is considered.

NOTE: *The W116 Tag does not check for expressions in signal or variable indexes, such as bit-select expression. This Tag does not report a violation for integer variable expressions.*

VHDL

The W116 Tag flags bit-width mismatch between LHS and RHS expressions of bit-wise logical, arithmetic, or relational operations.

The W116 Tag also checks the range in case of range-constrained integers, instead of calculating the bit-width.

Following is the list of operators covered under the W116 Tag:

Arithmetic Operators	
Subtraction (-)	Addition (+)
Multiplication (*)	Division (/)
modulus (mod)	Remainder (rem)
Logical Operators	
logical and (and)	logical or (or)
logical nand (nand)	logical nor (nor)
logical xor (xor)	logical xnor (xnor)
Relational Operators	
Equals (=)	Not Equals (!=)
Greater than (>)	Greater than or equal to (>=)
Less than (<)	Less than or equal to (<=)

When using integer constants, width of the constant is the natural width of

the initial value, not the default width (VHDL LRM states it to be 32 bits). For example,

```
constant c1 : integer := 9;
```

Here, the width of the constant will be $\log_2(9) + 1 = 4$, not 32.

For VHDL constant arrays, such as `Array = {1,0,3,6,0,2,3}`, when non-static index is passed for constant arrays like `Array[x]`, it's width is considered to be the maximum width among all elements of the array. For example, here the width of the `Array[x]` is 3, which is the width of the largest element (6).

Width Calculation for VHDL

If you set the value of the [nocheckoverflow](#) parameter to `yes` or `W116`, the `W116` Tag calculates the width according to the LRM (numeric_std lib) as shown in the following examples.

- For addition and subtraction, width is calculated based on the following Tags:
 - ☐ If both the operands are variables, then max width is considered.
 - ☐ If one operand is a variable and other static, then the variable width is considered.
- For multiplication, width is calculated based on the following Tags:
 - ☐ If both the operands are variable, then the RHS width is the sum of both the variables.
 - ☐ If one operand is a variable and other static, then the RHS width is `2*(Variable width)`.
- For Division width is calculated based on the following Tags:
 - ☐ If both operands are variable, then the left operand width is considered as the expression width.
 - ☐ If one operand is a variable and the other static then the width of the expression is considered.

Customizing Violation Messages

You can customize the violation messages by using the VC SpyGlass Lint's overload feature. Following message handles are available for Verilog and VHDL:

- **Verilog:** For logical and ternary operators, use the LINT_W116_ERR message handle to overload the messages. For rest of the operators, use LINT_W116_WRN.
- **VHDL:** For logical and relational operators, use the LINT_W116_ERR message handle to overload the message. For rest of the operators, use LINT_W116_WRN.

To overload the Tag, create a lint-policy-overload.pl file and modify the message details as needed. For example, to overload the message severity for logical and ternary operators, in the Tag, add the following line to the lint-policy-overload .pl file:

```
spyOverload("LANGUAGE" => "Verilog", "Tag" => "W116", "SEVERITY"  
=> "ERROR", "MESSAGELABEL" => "LINT_W116_ERR")
```

To know more about the Tag overloading feature, refer to *VC SpyGlass Lint Policy Customization Guide*.

Tag Exceptions

For **Verilog**, the Tag does not report violations for expressions where the width of a constant expression, a constant integer, or a base number is less than the width of the other operand.

For **VHDL**, following are the exceptions to the W116 Tag:

- The W116 Tag does not check in the function and procedure body as size of the arguments may depend on the actual passed in the function or procedure call.
- The W116 Tag does not report violation for expressions where the width of decimal literal or character literal or based literal is less than the width of the other operand.
- The W116 Tag does not report violation for relational operators when one side of relational operator is incremental or decremental and the size of its object is equal to size of object on other side of relation.

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

- ***check_static_value***: Default value is `no`. Set the value of the parameter to `yes` or `<Tag_list>` to report violation for cases with width mismatch, involving static expressions and non-static expressions having a static part. Other possible values are `only_const` and `only_expr`. For an expression having one operand constant and other non constant, a violation is reported only when the width of the constant operand is greater.
- ***disable_rtl_deadcode***: The default value is `no`. Set the value of the parameter to `yes` to disable violations for disabled code in loops and conditional (if condition, ternary operator) statements.
- ***enable_rtl_deadcode***: The default value is `no`. Set this parameter to `yes` to report different violations for dead codes in the RTL resulting from `if/for/while` and ternary operators that are ignored when the ***disable_rtl_deadcode*** parameter is enabled.
- ***ignore_mult_and_div***: Default value is `no`. Set the value of the parameter to `yes` to ignore violations for multiplication (*) and division (/) operations.
- ***reportconstassign***: The default value is `no`. Set the value of the parameter to `yes` to enable the W116 Tag to check for constants whose width is less than the operand.
- ***sign_extend_func_names***: The default value is "EXTEND". Set the value of the parameter comma-separated list of function names to enable the W116 Tag to recognize VHDL sign extension functions and calculate width of extend functions as per the const extension argument specified in the argument list.
- ***strict***: Default value is `no`. Therefore, the Tag behavior is as follows:
 - ☐ Ignores addition (+) and multiplication (*) operations
 - ☐ Reports violation on subtraction (-), division (/), or modulus (%) operations only if the width of the right operand is greater than the width of the left operand.
 - ☐ Set this parameter to `yes` to check for the addition and multiplication operations and to report subtraction, division, or modulus operations when there is a width mismatch between operands (both $A > B$ and

$B > A$ for operations $A-B$, A/B , and $A\%B$). You can also set this parameter to check for ternary operators

- **[use_lrm_width](#)**: Default value is `no`. Set this parameter to `yes` to consider the LRM width of integer constants, which is 32 bits. The Tag does not check the bit-width as per LRM by default.
On setting the `use_lrm_width` parameter to `yes` or `W116`, the `W116` tag does not report violations for operations between integer types when the widths are not greater than 32. If one operand is of the integer type and the other is not, then the width of that expression is considered as 32 (provided that expression's maximum value is not greater than $(2^{32})-1$).
- **[nocheckoverflow](#)**: Default value is `no`. Set the value of the parameter to `yes` or Tag name to calculate the width as per the LRM. See [Width Calculation for Verilog](#) and [Width Calculation for VHDL](#) to know more about calculating width.
- **[checkOperatorOverload](#)**: The default value is `yes`. Set the value of this parameter to `no` to report inconsistent bit-width mismatch for the overloaded operators from non-IEEE packages. This parameter is applicable for **VHDL** only.
- **[use_carry_bit](#)**: Default value is `no` and the width is taken as maximum of the two operands for a binary expression having plus and minus operators. Set this parameter to `yes` or `<Tag-name>` to get width after considering the carry bit of addition. No violation is reported, even using this parameter, for sub-expressions of a binary expression if all terms have the same width and all operators are either plus or minus. This parameter is applicable for **Verilog** only. Also, refer to the *Example 10*.
- **[check_genvar](#)**: Default value is `no` and the Tag does not report violation when the operands are genvar. When this parameter is set to `yes`, the `W116` Tag reports a violation for unequal length operands in the bit-wise logical, arithmetic, and ternary operators, although operands are genvar. This parameter is applicable for **Verilog** only.
- **[ignore_forloop_indexes](#)**: Default value is `no`. Set the value of this parameter to `yes` to ignore the `for` expressions that contain index variables of `for` loops.

- *check_counter_assignment*: Default value is no. Set this parameter to yes to report a violation for the counter type of assignments.
- *ignore_nonstatic_counter*: Default value is no. Set the value of the parameter to yes to ignore violations for non-static single-bit argument in additions and subtractions.

Constraint(s)

None

Messages and Suggested Fix

Verilog

The following message appears at the location of an operation of operator *<opr-name>* where there is a bit-width mismatch between left expression *<exprl>* of bit-width *<bit-widthl>* and right expression *<exprr>* of bit-width *<bit-widthr>*:

[WARNING] For operator (*<opr-name>*), left expression: "*<exprl>*" width *<bit-widthl>* should match right expression: "*<exprr>*" width *<bit-widthr>*. [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the complete hierarchical path of the containing scope.

Potential Issues

A violation is reported when there is a bit-width mismatch between the left expression of bit-width and right-expression of bit-width.

Consequences of Not Fixing

While working with expressions of different bit-widths may be the intended behavior, it is also a potentially error-prone design practice. For example, the addition of two words of unequal widths may indicate that you forgot to update the width of one of the buses.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the line, where, width mismatch in operators is detected.

To resolve the violation, check each case for potential messages, especially in the bitwise operators. Review the RTL code mentioned in message, this code may cause some unintended behavior. Make all arguments in such

comparisons of equal width such as by explicitly extending narrower operators in a concatenation, to see cases where upper bits are zeroed.

VHDL

The following message appears at the location of a statement where there is a bit-width mismatch between left expression `<exprl>` of bit-width `<bit-widthl>` and right expression `<exprr>` of bit-width `<bit-widthr>`:

[WARNING] Left expression: '`<exprl>`' (width `<bit-widthl>`) does not match right expression: '`<exprr>`' (width `<bit-widthr>`)
[Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the containing scope.

Potential Issues

Violation can arise when there is a bit-width mismatch between operands of the bit-wise logical, arithmetic, and ternary operators.

Consequences of Not Fixing

While working with expressions of different bit-widths may be the intended behavior, it is also a potentially error-prone design practice. For example, the addition of two words of unequal widths may indicate that you forgot to update the width of one of the buses.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the line, where, width mismatch in operators is detected.

To resolve the violation, check each case for potential messages, especially in the bitwise operators. Review the RTL code mentioned in message, this code may cause some unintended behavior. Make all arguments in such comparisons of equal width such as by explicitly extending narrower operators in a concatenation, to see cases where upper bits are zeroed.

Examples Code and/or Schematic

Example 1

Consider the following example:

```
signal a: unsigned(2 downto 0);  
signal b: unsigned(2 downto 0);
```

```
if((a + b) /= b) then
    output <= input;
end if;
```

In the above example, since both the operands are variables, max width is considered. Also, the W116 Tag does not flag a violation, if the value of the [nocheckoverflow](#) parameter is set to yes.

Example 2

Consider the following example:

```
if(a /= b + 2) then
    output <= input;
end if;
```

In the above example, since one operand is a variable and other is static, the variable width is considered. Also, the W116 Tag does not flag a violation in this case, if the value of the [nocheckoverflow](#) parameter is set to yes.

Example 3

Consider the following example:

```
signal a: unsigned(2 downto 0);
signal b: unsigned(3 downto 0);
    if((a * b) /= b) then
        b <= a*b;
    end if;
```

In the above example, since both the operands are variables, the RHS width is the sum of both the variables. Also, the W116 Tag flags a violation in this case as the width of (a*b) is 7 and the width of b is 4.

Example 4

Consider the following example:

```
if(a*2 /= b) then
    output <= input;
end if;
```

In the above example, since one operand is a variable and other static, the RHS width is 2*(Variable width). Also, the W116 Tag flags a violation in this case as the width of (a*2) is 6 and the width of b is 4.

Example 5

Consider the following example:

```
signal a: unsigned(2 downto 0);
signal b: unsigned(4 downto 0);
  if((a / b) /= b) then
    output <= input;
  end if;
```

In the above example, since both the operands are variables, the left operand width is considered as the expression width. Also, the W116 Tag flags two violations for the If condition.

Example 6

Consider the following example:

```
if(a /= b/5) then
  output <= input;
if(a /= 5/b) then
  output <= input;]
```

In the above example, since one operand is a variable and the other static, the width of the expression is considered as the variable width. Also, the W116 Tag flags a violation as the width of a is 3 and (b/5) is 5.

Example 7

Consider the following example:

```
assign out = base [4:0] - (1023>>6);
assign out = base [4:0] - 10 ;
assign out = base [4:0] - 1023;
```

In the above example, when the `check_static_value` parameter is set, the Tag reports violation only for the third expression. No violations are reported for the first two expressions because the width of a constant expression, a constant integer, or a base number is less than the width of the other operand.

Example 8

Consider the following example:

```
if(a = b + 1)
```

In the above example, the W116 Tag does not report violation, if size of a

is equal to size of b, or size of a is equal to size of $(b + 1)$.

Example 9

(Verilog)

Consider the following example code in which the arithmetic operation involves operands of different bit-widths — register a (4 bits) and register data (16 bits):

```
module test(in, out, clk);
    input [3:0] in;
    input clk;
    output [3:0] out;

    reg [3:0] out, a, b;
    reg [15:0] data;

    always @(a)
    begin
        b = a + data;
    end

    always @(posedge clk)
    begin
        out <= b + 2;
    end
endmodule
```

Here, VC SpyGlass Lint generates the following violation:

For operator (+), left expression: "a" width 4 should match right expression: "data" width 16.

Example 10

(Verilog)

Consider the following example code:

```
module test ( a, b, c, o1);
    input a,b,c;
    output o1;
    wire d, e, f, g, h, i;
```

```
    assign o1 = a & (b + c); //Violation, when parameter
    use_carry_bit is set
    assign o1 = a+b+c-d-e+f; //No violation
endmodule
```

For the above example, VC SpyGlass Lint reports the following violation when the `use_carry_bit` parameter is set:

For operator (&), left expression: "a" width 1 should match right expression: "(b + c)" width 2. [Hierarchy: ':test']

Default Severity Label

Warning

Tag Group

Expression

Reports and Related Files

No related reports or files.

W120

**A variable has been defined but is not used (Verilog)
A signal/variable has been declared but is not used (VHDL)**

Language

Verilog, VHDL

Description

Verilog

The *W120* Tag reports the variables that are declared but are not used in scope of the module where they are defined.

The Tag checks all non-port variables (wire, reg, integer, time variables, etc.) and requires that the variable must be either set, read, or both in the current scope. For vector variables, the Tag requires that each bit must be either set, read, or both in the current scope. Therefore, the Tag reports any bits of vector variables that are neither set nor read in the current scope.

While such constructs are allowed, the Tag helps you clean up your design.

NOTE: *For Verilog, inside the nested `for` loops, signals of user-defined type like structures, interfaces, etc. are considered fully set if used on the left-hand side of an expression and fully read if used on the right-hand side of an expression.*

By default, the Tag reports a violation if any bit of a variable is defined but not used in the current scope of a module. If you set the [checkfullbus](#) parameter to `yes`, the Tag reports a violation only when all bits of a variable are completely unused in the current scope of a module.

By default, this Tag does not process large arrays. Set the [handle_large_bus](#) parameter to `yes` to process large arrays (greater than 50,000) and report violation if not used correctly.

NOTE: *This Tag does not check the same variable used in different generate blocks.*

VHDL

The *W120* Tag reports signals or variables that are never used in the design.

The Tag requires that the signal or variable must be either set, read, or both. For vector signals or variables, the Tag requires that each bit must be

either set, read, or both. Therefore, the Tag reports any bits of vector signals or variables that are neither set nor read in the current scope.

While such constructs are allowed, the Tag helps you clean up your design.

For multidimensional variables, only the total count of unused bits is reported in the violation message. To see the details of the violating bits, please refer to the [SignalUsageReport](#). You can access the report from the *VC SpyGlass Lint_reports/lint* directory.

By default, the Tag reports a violation if any bit of a variable is declared but not used in the current scope of an architecture. If you set the [checkfullbus](#) parameter to *yes*, the Tag reports a violation only when all bits of a variable are completely unused in the current scope of an architecture.

By default, value [checkfullrecord](#) is set to *no*. Set this parameter to *yes* to not report violation for a record if at least one element of the record is used.

Message Details

Verilog

Message 1

The following message appears at the location where a variable *<var-name>* is declared but never used in the design:

variable '<var-name>'<no_of_bits> declared but not used
[Hierarchy: '<hier-path>']

Where,

- *<hier-path>* is the complete hierarchical name of the containing scope excluding subprograms.
- *<no_of_bits>* is the number of unused bits of multidimensional array.

Message 2

The following message is displayed when the signal size is greater than 50,000 and the [handle_large_bus](#) parameter is disabled:

signal '<signal-name>' size too big thus not processed, use
'configure_lint_tag_parameter -tag w120 -parameter
HANDLE_LARGE_BUS -value yes' for enabling handling of these

signals

If the `HANDLE_LARGE_BUS` parameter is not enabled, the violation for signals of size greater than 50,000 is missed.

VHDL

The following message appears at the location where a signal `<sig-name>` is declared that is never used in the design:

The signal '`<sig-name><bits>`' is not used [Hierarchy: '`<hier-path>`']

The following message appears at the location where a variable `<var-name>` is declared that is never used in the design:

The variable '`<var-name><bits>`' is not used [Hierarchy: '`<hier-path>`']

Where, `<bits>` refers to unused bits of vector variable/signal . For multidimensional array, `<bits>` is the total number of unused bits.

Also, the following message is generated, when the `W120` Tag flags a violation for vector or multidimensional signal:

Please refer to 'SignalUsageReport.rpt' for details of violating bits

In the Console, click on the above violation message to view the [SignalUsageReport](#).

Tag Severity

Warning

Suggested Fix

Check all such cases. If you see a variable reported that you think should be used, suspect a possible typing error.

Examples (Verilog)

Example 1

Consider the following example:

```
module W120_mod2(in1, clk, out1);
    input [2:0] in1;
```

```
input clk;
output [2:0] out1;
reg [2:0] out1;

wire [4:0] net;

always@(posedge clk)
    #20 out1 <= in1 && net[2:0];

endmodule
```

In this example, not all bits of the net are used in the `always` block. Therefore, by default, the *W120* Tag reports a violation. However, if you set the *checkfullbus* parameter to `yes`, the Tag does not report any violation.

Example 2

Consider the following example:

```
module W120_top(i1, clock, o1);
    inout [2:0] i1;
    input clock;
    output [2:0] o1;

    W120_mod my_mod(i1, clock, o1);

endmodule

module W120_mod(in1, clk, out1);
    input [2:0] in1;
    input clk;
    output [2:0] out1;
    reg [2:0] out1;

    wire [2:0] net;

    always@(posedge clk)
        #20 out1 <= in1;

endmodule
```

In this example, all bits of the net are completely unused in the always block. Therefore, by default, the *W120* Tag reports a violation. The Tag will also report a violation even if you set the *checkfullbus* parameter to yes.

Examples (VHDL)

Example 1

Consider the following example:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity entW120 is
  port(a: in std_logic;
        b: in std_logic;
        z: out std_logic
  );
end entW120;

architecture behav of entW120 is
  signal sig1: std_logic_vector(2 DOWNT0 0);
begin
  process(a, b)
  begin
    z <= a or sig1(0);
  end process;
end behav;
```

In this example, not all bits of the sig1 signal are used in the process block. Therefore, by default, the *W120* Tag reports a violation. However, if you set the *checkfullbus* parameter to yes, the Tag does not report any violation.

W121

A variable name collides with and may shadow another variable

Language

Verilog

Description

The W121 Tag flags object names that are not unique within the current scope.

The W121 Tag flags an object name that is same as the name of another object of different object type located within the same scope.

The W121 Tag checks for name case variants within a specific scope, that is, the boundaries of a module, a UDP, a function, a task, or a named block. If a scope is contained in another scope (say, a named `always` construct in a module), the names in the containing scope (module) are also checked against the names in the contained scope (named `always` construct).

Names are not checked across parallel scopes.

NOTE: *The W121 Tag supports generate-if, generate-for, and generate-case blocks.*

Parameters

- **`waiver_compat`:** Default value is `no`. If you set the value of this parameter to `yes` or `<Tag-name>`, it ensures that the Tag does not generate the line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.
- **`ignore_genvar`:** Default value is `no` and the W121 Tag reports violations for variables shadowed by genvar variables. Set the value of this parameter to `yes` to not report violation for such cases.
- **`ignore_scope_names`:** Default value is `no` and the W121 Tag checks for scope names. Set this parameter to `yes` to ignore Tag checking on scope names.
- **`ignore_macro_to_nonmacro`:** Default value is `no`. Set the value of the parameter to `yes` to ignore name matching for macro to non-macro names.

- *limit_task_function_scope*: Default value is no. Set the value of the parameter to yes to ignore Tag checking if a signal is declared with function/task scope as well as with module scope.

Message Details

The following message appears at the location where an object name *<name>* is declared that is same as the name of another object of different object type located within the same scope:

Name '*<name>*' is not unique - may shadow another variable
(Previously used at line no. *<line-num>* in *<file-name>*)

Where *<line-num>* is the line number where the variable of similar name has been defined earlier and *<file-name>* is the file in which the variable of similar name has been defined earlier.

Tag Severity

Warning

Suggested Fix

In general, this is considered as loose coding style. It is best to ensure that variables within a given scope have unique names. Take cases reported by this Tag and change names so the Tag is not violated.

Examples

The following example shows some cases of name clashes:

```
`define M1 1
module test (in1, in2, clk, rst, out1, out2);
    input in1, in2, clk, rst;
    output out1, out2;
    reg out1, out2;
    wire test;
    assign test = rst ? in1 : in2;
    always @( clk)
        begin : P2
            out1 = P1(in1 , rst);
            M1( in1, in2, out2);
```

```

    end
    always@(posedge clk)
    begin: block1
        if (rst)
            begin: block1
                reg out1;
                out1 = P1(in2, rst);
                M1( in1, `M1, out2);
            end
        else
            begin
                out1 = 0;
                out2 = 1;
            end
        end
    end
    task M1;
        input in, sel;
        output out;
        out = in & sel;
    endtask
    function P1;
        input in, sel;
        P1 = in & sel;
    endfunction
endmodule

```

The wire `test` has a name clash with the module `test`. Thus, VC SpyGlass Lint generates the following message:

Name 'test' is not unique - may shadow another variable
(Previously used at line no. 2 in top.v)

There are two blocks named `block1` in the same `always` construct. Thus, VC SpyGlass Lint generates the following message:

Name 'block1' is not unique - may shadow another variable
(Previously used at line no. 14 in top.v)

The register `out1` declared in the `always` construct has a name clash with a register with same name declared outside the construct. Thus, VC SpyGlass Lint generates the following message:

Lint Tags

Name 'out1' is not unique - may shadow another variable
(Previously used at line no. 10 in top.v)

The task M1 has a name clash with parameter M1. Thus, VC SpyGlass Lint generates the following message:

Name 'M1' is not unique - may shadow another variable
(Previously used at line no. 19 in top.v)

W122

A signal is read inside a combinational always block but is not included in the sensitivity list (Verilog)
A signal is read inside a combinational process but is not included in the sensitivity list (VHDL)

When to Use

Use this Tag to identify the signals that are read inside a combinational `always` block or combinational process but are not included in the sensitivity list.

Description

Verilog

The *W122* Tag reports violations for the signals that are read in a combinational `always` construct but are not present in the sensitivity list of the construct.

The Tag does not check the `for` loop indices because they do not represent real signals.

The Tag reports violations for global signals or variables that are read in the tasks or functions called from the `always` blocks that have wildcard (`@*`) sensitivity list. The wildcard infers sensitivity to only those signals that are directly referenced in the `always` block. It does not infer sensitivity to signals that are externally referenced in a function or a task that is called from the `always` block. Therefore, the wildcard is sensitive only to those signals that are passed to a function or a task.

NOTE: *The Tag supports the multi-line functionality, generate-if block, and generate-for block.*

VHDL

The *W122* Tag reports violations for the signals, which are read in a combinational process but are not present in the sensitivity list of the process.

NOTE: *The Tag supports the multi-line functionality.*

Tag Exceptions

- The Tag does not report violations for signals which are assigned constant values outside the `always` block using continuous assignment statement and are missing from the sensitivity list. The Tag also does

not report violations for the variables which are assigned values in tasks and then are read in the same `always` construct. These variables are connected to the output of a task.

- The Tag will not report any violation for delay variable used inside the `always` blocks, which are not read inside the sensitivity list.

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

Verilog

- [*allow_clk_in_condition*](#): Default value is `no`. If you set this parameter to `yes`, the Tag does not traverse a combinational block present inside a sequential block and does not report a violation in such cases.
- [*simplesense*](#): Default value is `no`. Set the value of this parameter to `yes` to enable the Tag to report the signals that are read in an `always` construct and being set in blocking assignment statement but are not present in the sensitivity list.
- [*strict*](#): Default value is `no`. Set this parameter to `yes` to report memories and delay variables that are read in the `always` construct but are not present in the sensitivity list.
- [*report_all_messages*](#): Default value is `no`. Set this parameter to `yes` or Tag name(s) to report the bit information in the violation messages.

VHDL

- [*considerInoutAsOutput*](#): Default value is `no`. Set the value of the parameter to `yes` to consider the inout port as output port.
- [*strict*](#): Default value is `no`. This indicates the *W122* Tag reports a violation for a signal that is read in the `process` block and initialized to a value outside the `process` block but is not completely present in the sensitivity list. If you set this parameter to `yes`, the *W122* Tag does not report a violation in such cases.

NOTE: *The W122 Tag does not report a violation for a signal that is assigned a value*

outside the process block by using the concurrent assignment statement.

- ***ignoreSeqProcess***: Default value is no. Set this parameter to yes to ignore Tag checking inside the sequential process blocks.

Constraint(s)

None

Messages and Suggested Fix

Verilog

The following message is displayed for the *<name>* signal or the *<name>* variable which is read in the `always` construct but is not present in the sensitivity list:

[WARNING] The signal/variable '*<name>*' (or some of its bits) read in the block is not in the sensitivity list [Hierarchy: '*<hier-path>*']

If the [*report_all_messages*](#) parameter is set, then the violation message is modified, as follows:

[WARNING] Signal/variable '*<name>*', that is read in this block, is not present in sensitivity list [Hier: *<hier-path>*]

Where, *<hier-path>* is the complete hierarchical path of the signal.

Potential Issues

A violation is reported for a signal or a variable, which is read in the `always` block but is not present in the sensitivity list.

Consequences of Not Fixing

Signals missing from the sensitivity list can lead to a mismatch between the pre- and post-synthesis simulations. The simulation only evaluates the changes in the combinational logic when the signals in the sensitivity list change before synthesis. The synthesis process generates a logic which reads all the required values, whether they are present in the sensitivity list or not. Effectively, the missing signals are added to the sensitivity list. Therefore, the post-synthesis simulation results are different from the pre-synthesis simulation results.

How to Debug and Fix

To debug the violation, double-click the message. The HDL Viewer window highlights the line in which the specified signal or variable is read in the denominational `always` block. This signal is missing in the sensitivity list of the denominational `always` block. Confirm the same in one of the following ways:

- Scroll up the HDL window till you reach the `always` block and then check the sensitivity list.
- When the `always` block is very long, view the HDL in an editor and search backwards for the nearest `always` block, and then check the sensitivity list.
To ensure that the pre-synthesis simulation results match the post-synthesis simulation results, add the missing signals to the sensitivity list of the combinational `always` construct.

VHDL

The following message is displayed for the `<construct-name>` construct of the `<construct-type>` type which is read in the process block but is not present in the sensitivity list:

[WARNING] The `<construct-type>` '`<construct-name>`' (or some of its bits) read in the process is not in the process sensitivity list. [Hierarchy: '`<hier-path>`']

Where, `<construct-type>` is a port, signal, or variable and `<hier-path>` is the complete hierarchical path of the containing process.

Potential Issues

A violation is reported for a construct, which is read in the process block but is not present in the sensitivity list.

Consequences of Not Fixing

Signals missing from the sensitivity list can lead to a mismatch between the pre- and post-synthesis simulations. The simulation only evaluates the changes in the combinational logic when the signals in the sensitivity list change before synthesis. The synthesis process generates a logic which reads all the required values, whether they are present in the sensitivity list or not. Effectively, the missing signals are added to the sensitivity list. Therefore, the post-synthesis simulation results are different from the pre-synthesis simulation results.

How to Debug and Fix

To debug the violation, double-click the violation message. The HDL Viewer window highlights the line, where, the specified signal is read in the denominational `process` block. This signal is missing in the sensitivity list of the denominational `process` block. You can confirm the same in any one of the following ways:

- Scroll up the HDL window till you reach the process and check the sensitivity list.
- If the `process` block is a very long block, view the HDL in an editor, and, search backwards for the nearest `process` block and then check the sensitivity list.

To resolve the violation, add the reported signals to the sensitivity list.

Example Code and/or Schematic

Verilog

Example 1

Consider the following example:

```
module mod_NotInSensOM_Test1 ( in1, in2, out1 );
input in1, in2;
output out1;
reg out1;

    always @(in1)
    begin
        out1 = in1&in2;
    end

endmodule
```

The *W122* Tag reports a violation for the above example because the `in2` variable, which is read in the combinational `always` block, is not present in the sensitivity list.

Example 2

Consider the following example:

```
module chip (input wire [ 7:0] a, b,
            input wire [15:0] max_prod,
```


Lint Tags

```

input wire [ 8:0] max_sum,
input wire error,
output logic [ 8:0] sum_out,
output logic [15:0] mult_out);

function [8:0] mult (input [7:0] m, n);
    mult = 0;
    if (!error) // error is an external signal
        mult = m * n;
        if (mult > max_prod)
            //max_prod is an external signal
            mult = max_prod;
endfunction
task sum (input [7:0] p, q, output [8:0] sum_out);
    sum_out = 0;
    if (!error) // error is an external signal
        sum_out = p + q;
        if (sum_out > max_sum)
            // max_sum is an external signal
            sum_out = max_sum;
endtask

always @* begin
    // @* will only be sensitive to a and b
    sum(a, b, sum_out);
    // @* will not be sensitive to max_prod,
    // max_sum or error
    mult_out = mult(a, b);
end
endmodule

```

In the above example, the *W122* Tag reports a violation because the global signals or variables, which are read in the tasks or functions called from the always blocks, have wildcard (@*) sensitivity list.

Example 3

Consider the following example:

```
module test (clk,in,in1,out);
```

```

input clk,in,in1;
output out;
reg out;

always@(posedge clk)
begin
if (clk)
    out = in ^ in1;
end
endmodule

```

In the above example, the *W122* Tag, by default, reports violations for signals *in* and *in1*. If you set the [allow_clk_in_condition](#) parameter to *yes*, the Tag does not report any violation.

VHDL

Example 1

Consider the following example where the signal *rst* is read in the process block *test* (in architecture *arc* of entity *ent*) but is not present in the sensitivity list of the process block:

```

library IEEE;
use IEEE.std_logic_1164.all;

test: process(clk)
begin
    if (rst = '0') then
        out1 <= "0000";
    elsif (clk'event and clk = '1') then
        out1 <= in1;
    end if;
end process test;

```

For this example, VC SpyGlass Lint generates the following message:

```

The in port 'rst' (or some of its bits) read in the process is
not in the process sensitivity list. [Hierarchy:
':ent(arc):test:']

```

To resolve the violation, rewrite the design to include the signal *rst* in the sensitivity list as follows:

```
test: process(rst, clk)
```

Example 2

Consider the following example:

```
entity test is
port (d1 : in std_logic_vector (2 downto 0);
      d2  : in std_logic;
      o1  : out std_logic_vector (2 downto 0));
end test;
architecture behav of test is
begin
  process(d1(2), d2)
  begin
    for i in 1 to 1 loop
      o1(1) <= d1(i+1) and d2;
    end loop;
  end process;
end behav;
```

In the above example, the signal `d1` is read in the process block and is also present in the sensitivity list. However, note that the *W122* Tag evaluates only single-level binary operations, such as `(d1(i+1))`. Any level more than one for binary operations are not evaluated (for example, `d1(1+i+1)`).

Example 3

In the following example, the *W122* Tag reports a violation for the `SIG1` signal because it is initialized to a value outside the process block:

```
entity ent24 is
  port(A : in BIT_VECTOR(0 to 3); Z : out BIT_VECTOR(0 to 3));
end ent24;
architecture arch24 of ent24 is
  signal SIG : BIT_VECTOR(0 to 3);
  signal C : BIT_VECTOR(0 to 3);
  signal SIG1 : BIT_VECTOR(0 to 3) := "0010";
begin
  SIG <= (OTHERS => '0');
  process (A(0 to 1), SIG(0 to 1), SIG1(0 to 1))
```

```
begin
  for NUM in 0 to 3 loop
    C(NUM) <= SIG(NUM);
    Z (NUM) <= SIG1 (NUM) ;
  end loop;
end process;
end arch24;
```

The *W122* Tag does not report a violation for the *SIG* signal because it is assigned a value outside the `process` block by using the concurrent assignment statement.

Default Severity Label

Warning

Tag Group

Simulation

Reports and Related Files

No related reports or files.

W123

Identifies the signals and variables that are read but not set

When to Use

Use this Tag to identify signals and variables that are read but not set.

Description

Verilog

The *W123* Tag reports violations for the variables that are read but not set in the design.

The Tag determines the non-port value within a module which is read but never assigned a value. The Tag also checks for all the `reg`, `wire`, and `integer` variables that are declared inside the module.

The *W123* Tag in addition to static conditions, simple non-static conditions involving FOR-loop index are also evaluated.

NOTE: *Inside the nested `for` loops, signals of user-defined type like structures, interfaces, etc. are considered fully `set` if used on the left-hand side of an expression and fully `read` if used on the right-hand side of an expression.*

For multidimensional variables, only the total count of violating bits that are read but not set, is reported in the violation message. To see the details of the violating bits, refer to the [SignalUsageReport](#) report in the *VC SpyGlass Lint_reports/lint* directory.

A gate of a CMOS transistor needs to be at a defined state and electrical level of logic 1 or 0, for the transistor to function as an on-off switch. If the gate of a transistor is left floating, then the corresponding transistor can get into a `limbo` state of partially on.

In the context of a logic gate, such a partially-on transistor can lead to indeterminate logic value and analog electrical state on the output of that gate.

For an inverter, if the input is left floating, both the `n` and `p` transistors can be partially on creating a steady state path from VDD to ground.

In such cases, the output of a transistor would be in a non-digital state, that is, somewhere between logic 1 and 0. This state would then propagate to all the input pins of gates in the fan-out cone of the inverter. Those

gates, in turn would produce a non-digital output and would propagate further. Thus potentially large number of nodes in a chip would be in the indeterminate non-digital state.

Also, the steady state path created between VDD and ground through the inverter transistors leads to an excessive current draw from VDD. When many nodes are in this indeterminate state, a large amount of current is drawn from VDD leading to a burn-out or latch-up of the chip.

VHDL

The *W123* Tag reports violations for signals or variables that are read but not set in the design. The Tag evaluates only the static conditions.

For multidimensional variables, only the total count of offending bits that are read but not set, is reported in the violation message. To see the details of the violating bits, refer to the *SignalUsageReport* report in the report from the *VC SpyGlass Lint_reports/lint* directory.

Tag Exceptions

For VHDL, following are the exceptions to the *W123* Tag:

- The Tag does not report violations for variables read inside dead code (conditional or selected signal assignment).
- The Tag does not consider assignments inside a conditional statement when the condition is false. In VHDL, only the static conditions are evaluated. In Verilog, in addition to static conditions, simple non-static conditions involving FOR-loop index are also evaluated.

NOTE: *The Tag does not evaluate the complex IF-conditions involving more than one loop variable. The Tag checks both the assignments inside the if and the else constructs.*

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

- ***ignoreModuleInstance***: Default value is `no`. Set this parameter to `yes` to ignore the unset variables that are used only in the instance port mapping.
- ***handle_large_bus***: Default value is `no` and this Tag does not process large arrays. Set the parameter to `yes` to process large arrays (greater than 50,000) and report violation if not used correctly.
- ***traverse_function***: Default value is `no`. Set this parameter to `yes` to consider the inputs as read when being read inside a function. This parameter is applicable to **VHDL** only.
- ***checkfullrecord***: Default value is `no`. Set this parameter to `yes` to not report violation for a VHDL record if at least one element of the record is read and set.
- ***ignore_hier_scope_var***: Default value is `no` and the *W123* Tag reports all user-defined data types. Set the value of this parameter to `yes` to not report any violation for the user-defined data types. This parameter is applicable for Verilog only.
- ***dump_array_bits***: Default value is `no`. Set this parameter to `yes`, to enable the *W123* Tag to include multi-dimensional signals/bits information in the violation message. This parameter is applicable to **VHDL** only.
- ***report_struct_name_only***: By default, the value of the parameter is `no`. Set the value of the parameter to `yes` to enable the *W123* Tag to report one violation for a struct variable.

Constraint(s)

None

Messages and Suggested Fix

Verilog

Message 1

The following message is displayed when the `<var-name>` variable that is never set in the design, is read for the first time:

[WARNING] variable '`<var-name>`'`<no_of_bits>` read but never set

[Hierarchy: '<hier-path>']

Where,

- *<hier-path>* is the complete hierarchical name of the containing scope excluding subprograms.
- *<no_of_bits>* is the number of offending bits of multidimensional array, which are read but not set.

Potential Issues

A violation is reported when a variable is read but never set.

Consequences of Not Fixing

It is a poor design practice to set values across hierarchy boundaries by using hierarchical names. Reading uninitialized value may result in undriven nets post synthesis.

How to Debug and Fix

To debug the violation, double-click the message. The HDL Viewer window highlights the line in which the variable is read. To check whether the variable is set, search the variable name in the source file and check all the usages of the variable in the corresponding module or architecture. To view the list of signal bits that are read but not set, open the [SignalUsageReport](#) section from the Tools -> Report menu or double-click the violation message.

To resolve the violation, review the RTL to determine why the variable is not set. Also, check for the typing error in the variable name.

Message 2

The following message is displayed when the signal size is greater than 50,000 and the [handle_large_bus](#) parameter is disabled:

[INFO] Signal '<signal-name>' size too big thus not processed, use 'configure_lint_tag_parameter -tag W123 -parameter HANDLE_LARGE_BUS -value yes' for enabling handling of these signals

Potential Issues

If the `HANDLE_LARGE_BUS` parameter is not enabled, the violation for signals of size greater than 50,000 is missed.

Consequences of Not Fixing

None

How to Debug and Fix

None

Message 3

The following message is displayed when the [report_struct_name_only](#) parameter is set to yes:

```
[INFO] variable '<variable_name>' (total bits
<violation_bit_count>)' read but never set.[Hierarchy:
'<hierarchy>']"
```

Potential Issues

The violation message is reported when the [report_struct_name_only](#) parameter is set to yes

Consequences of Not Fixing

None

How to Debug and Fix

None

VHDL

Message 1

The following message is displayed for the `<sig-name>` signal that is read but not set in the design:

```
[WARNING] The signal '<sig-name>' <bits> is read but not set
[Hierarchy: '<hier-path>']
```

Potential Issues

A violation is reported when a signal is read but never set.

Consequences of Not Fixing

Reading uninitialized value may result in undriven nets post synthesis.

How to Debug and Fix

To debug the violation, double-click the message. The HDL Viewer window highlights the line in which the signal is read. To check whether the signal is set, search the signal name in the source file and check all the usages of the signal in the corresponding module or architecture. To view the list of signal bits that are read but not set, open the [SignalUsageReport](#) section

from the Tools -> Report menu or double-click the violation message. To resolve the violation, review the RTL to determine why the signal is not set. Also, check for the typing error in the signal name.

Message 2

The following message is displayed at the location of the declaration of a variable `<var-name>` that is read but not set in the design:

[WARNING] The variable '`<var-name>`' `<bits>` is read but not set
[Hierarchy: '`<hier-path>`']

Where, `<bits>` are the vector bits, which are read but not set.

Potential Issues

A violation is reported when a variable is read but never set.

Consequences of Not Fixing

Reading uninitialized value may result in undriven nets post synthesis.

How to Debug and Fix

To debug the violation, double-click the message. The HDL Viewer window highlights the line in which the variable is read. To check whether the variable is set, search the variable name in the source file and check all the usages of the variable in the corresponding module or architecture. To view the list of variable bits that are read but not set, open the [SignalUsageReport](#) section from the Tools -> Report menu or double-click the violation message.

To resolve the violation, review the RTL to determine why the variable is not set. Also, check for the typing error in the variable name.

Message 3

The following message is generated, when the *W123* Tag reports a violation for vector or multidimensional signal:

[INFO] Please refer to 'SignalUsageReport.rpt' for details of violating bits

Potential Issues

A violation is reported when a vector or a multi-dimensional signal is read but not set in the design.

Consequences of Not Fixing

Reading an uninitialized value may result in undriven nets post synthesis.

How to Debug and Fix

To debug the violation, double-click the message. The HDL Viewer window highlights the line in which the signal is read. To check whether the signal is set, search the signal name in the source file and check all the usages of the signal in the corresponding module or architecture. To view the list of signal bits that are read but not set, open the [SignalUsageReport](#) section from the Tools -> Report menu or double-click the violation message.

To resolve the violation, review the RTL to determine why the signal is not set. Also, check for the typing error in the signal name.

Example Code and/or Schematic**Example 1**

In the following example code, VC SpyGlass Lint reports a violation for the `net[2:0]` variable, which is read but never set:

```
module test(in1, clk, out1);
    input [2:0] in1;
    input clk;
    output [2:0] out1;
    reg [2:0] out1;

    wire [2:0] net;

    always@(posedge clk)
        out1 <= in1 && net;

endmodule
```

Example 2

Consider the following example:

```
always @(posedge clk) begin
    for (k = 0; k < 3; k = k + 1)
        begin
            if (k > 3)
                sig2[k] <= sig1[k];
            else
                sig2[k] <= 1'b0;
```

```

    end
end

```

The W123 Tag does not report a violation for the above example as a signal is assigned inside the IF conditional statement, which is false.

Example 3

Consider the following **VHDL** example:

```

library IEEE;
use IEEE.std_logic_1164.all;

architecture rtl of test is
  signal match : std_logic;
  signal input1 : std_logic;

  function fnout(
    input2    : in std_logic;
    match1    : in std_logic
  )
  return std_logic is
  variable match2 : std_logic;
  begin
    match2 := match1; -- match used inside the function
  return match2;
  end fnout;

begin

  y <= fnout(input1,match);

end architecture;

```

In the above example, as the [traverse_function](#) parameter is set to yes, the input1 signal is not being read inside the function block, so it is not considered as read while the match signal is being read inside the function block, so it is considered as read.

Lint Tags

Default Severity Label

Warning, Info

Tag Group

Usage

Reports and Related Files

[*SignalUsageReport*](#)

W127

Delay values should not contain X (unknown value) or Z (high-impedance state)

Language

Verilog

Description

The W127 Tag flags delay values containing X (unknown value) or Z (high-impedance state).

Though not a syntax error, such descriptions are meaningless as they describe delay for an indeterminate period of time.

Message Details

The following message appears at the location where a delay value containing X (unknown value) or Z (high-impedance state) is encountered:

Delay value <value> should not contain X or Z

where, <value> is a delay expression.

Tag Severity

Warning

Suggested Fix:

Change delay value to an integer value.

Examples:

Consider the following example:

```
out1 = # (8'h1x) in1;  
out2 = # 5 in2;  
out3 = #(4'b10z1) in3;
```

In the above example, the W127 Tag reports two violations because the delay values contain X (unknown value) or Z (high-impedance state).

The following messages are reported by this example:

Lint Tags

Delay value '8'h1x' should not contain X or Z

Delay value '4'b10z1' should not contain X or Z

W126

Do not use non-integer delays

Language

Verilog

Description

The W126 Tag flags non-integer delay values.

It is recommended to control delay precision using the `'timescale` directive.

A floating-point delay value will at best be truncated to the current timescale and will not track changes in timescale values.

NOTE: *The W126 Tag does not flag negative integer delay values. See the [W128 Tag](#) to catch negative delay values.*

Message Details

The following message appears at the location where a non-integer delay value is encountered:

```
Do not use non-integer delay value <value>
```

where, *<value>* is a delay expression.

Tag Severity

Warning

Suggested Fix

Change all delay values to integer values and use `'timescale` to control delay accuracy.

W162

Extension of bits in constant integer conversion

Language

Verilog

Description

The W162 Tag flags constant integer assignments to signals when the width of the signal is wider than the width of the constant integer.

When assigning a constant integer value to a LHS operand, the width specification for RHS should match LHS operand width. If the signal is wider than the constant integer, the extra bits are padded with zeros. If the signal is narrower than the constant integer, the extra high-order non-zero bits are discarded.

The W162 Tag also checks for those non-constant assignments where extension is due to the constant present in the expression.

```
reg [4:0] a;  
reg [1:0] b;  
a = 10 + b;
```

NOTE: The W162 Tag supports *generate-if*, *generate-for*, and *generate-case* blocks.

Message Details

The following message appears at the location where a constant integer is assigned to a signal that is wider than the constant integer:

Extension of bits in constant integer conversion is erroneous
[Hierarchy: '<hier-path>']

Where, *<hier-path>* is the complete hierarchical path of the signal.

Tag Severity

Warning

Suggested Fix

No fix is required in general, as long as your are OK with zero-extension.

W164c

LHS width is greater than RHS width of assignment (Extension)

Language

Verilog

Description

The *W164c* Tag flags assignments in which the LHS width is greater than the (implied) width of the RHS expression.

The *W164c* Tag is same as the *W164b* Tag except the following two conditions:

- The *W164c* Tag checks for all assignments (including those assignments containing no wire or reg objects) and the *W164b* Tag checks all assignments only when the *strict* Tag parameter is set.
- The *W164c* Tag, unlike *W164b* Tag, does not report a violation, if the RHS is an unsigned unbased constant.

By default, the *W164c* Tag considers the natural width of integer constants, which is $\log_2(N)+1$. Set the value of the *use_lrm_width* Tag parameter to yes to consider the LRM width of integer constants, which is 32 bits.

When a signed signal is divided by another signed signal, the *W164c* Tag calculates the width as follows:

$\langle \text{width of numerator} \rangle + 1$

Consider the following example:

```
wire signed [2:0] a_3_sign, b_3_sign, c_sign_div;
assign c_sign_div = a_3_sign / b_3_sign;
```

The *W164c* Tag reports a violation for the above assignment when the RHS width is 4. This calculation is valid only for simple division expressions.

By default, the *W164c* Tag reports violation for disabled code in loops and conditional (if condition, ternary operator) statements. Set the value of the *disable_rtl_deadcode* parameter to yes to disable violations for disabled code in loops and conditional (if condition, ternary operator) statements.

By default, the width is calculated considering the best fit width of an expression. That is the width in which maximum value of an expression can be accommodated.

When you set the value of the [nocheckoverflow](#) parameter to `yes` then width is calculated according to the LRM and the natural width is considered for constants.

The behavior of the Tag is explained by the cases below.

■ For Constant Expressions

- ☐ For constant integer expressions, the width is calculated based on the value of expression:

```
out[5:0] = 2 + 15 ;    //RHS Width = 5 (Value = 17),
                        violation
out[10:0] = 100 << 4 ; //RHS Width = 11, NoViolation
out[6:0] = 100 >> 2 ;  //RHS Width = 5
```

- ☐ If constant is not a part of sub-expression then the specified width is considered:

```
out[11:0] = 12'b0 ;    //RHS Width = 12, No violation
```

- ☐ If the width of a based number is not specified then the natural width is considered:

```
out[11:0] = 'h344;     //RHS Width = 10, violation
```

For above cases, behavior of Tag remains same when the [nocheckoverflow](#) parameter is set to `yes`.

■ For Arithmetic Operators

- ☐ For addition operator, the width is calculated as the width in which maximum value of expression is accommodated. Consider the following example:

```
out[3:0] = in1[1:0] + in2[1:0] + in3[1:0]; //RHS Width
4, No violation
Max value RHS 3 + 3 + 3 = 9
RHS Width = 4
```

```
out[2:0] = in1[1:0] + (3/3) ;    //RHS Width 3, No
violation
Max Value RHS = 3 + 1 = 4
RHS Width = 3
```

```
out[6:0] = in1[5:0] + 5'b1010;  //RHS Width 7, No
```

```
violation
out[4:0] = 15 + 3'b111;           //RHS width 5, No
violation
```

- ❑ For the subtraction operator, the width is calculated in a similar way as it is done for addition:

```
out[4:0] = in1[3:0] - in1[1:0]; //RHS Width 5, No
violation
out[4:0] = in1[3:0] - 4'b1010; //RHS Width 5, No
violation
```

- ❑ For the multiplication operator, the width is calculated as follows:

When both operands are variable, then the RHS width is the sum of the width of both operands:

```
out[5:0] = in1[1:0] * in2[1:0] * in3[1:0]; //RHS Width
6, No violation
out[4:0] = in1[2:0] * in1[1:0];           //RHS Width 5,
No violation
```

When one operand is static and other is variable, then the RHS width is calculated considering maximum value of the expression:

```
out[7:0] = in1[3:0] * 4'b1;           //RHS Width 4, Max value
= 15*1 = 15, Violation
out[7:0] = in1[3:0] * 4'b10;         //RHS Width 5, Max value
= 15*2 = 30, Violation
out[7:0] = in1[3:0] * 4'b1010;       //RHS Width 8, Max value
= 15*10 = 150, No violation
out[6:0] = in1[3:0] * 4;              //RHS Width 6, Max value
= 15*4 = 60, violation
```

- ❑ For the division operator, the width of the RHS is assumed as the width of left operand:

```
out[2:0] = in1[2:0]/in1[1:0]; //RHS Width 3, No
violation
out[2:0] = in1[1:0]/in1[2:0]; //RHS Width 2, Violation
out[3:0] = in1[3:0]/4'b10;     //RHS Width 4, No
violation
out[3:0] = 4/in1[3:0];         //RHS Width 3,violation
```

- ❑ When the [nocheckoverflow](#) parameter is set to yes, the width is assumed to be the same as the width of term having maximum width. Consider the following example:

```
out[3:0] = in1[1:0] + in2[1:0] + in3[1:0]; //RHS Width
2, violation
out[6:0] = in1[5:0] + 5'b1010; //RHS Width 6, Violation
out[4:0] = 15 + 3'b111;          //RHS width 4, Violation
out[4:0] = in1[3:0] - in1[1:0]; //RHS Width 4, Violation
out[4:0] = in1[3:0] - 4'b1010; //RHS Width 4, Violation
out[4:0] = in1[2:0] * in1[1:0]; //RHS Width 3, Violation
out[7:0] = in1[3:0] * 4'b10;    //RHS Width 4, Violation
out[2:0] = in1[1:0]/in1[2:0];   //RHS Width 3, No
violation
out[3:0] = 4/in1[3:0];          //RHS Width 4, No
violation
```

■ For Shift Operator

- ❑ For the right shift, if left operand width is matching the LHS width then no violation is reported. Consider the following example:

```
out[3:0] = in1[3:0] >> in ;          //RHS Width 4, No
violation
out[5:0] = in1[4:0] >> 2 ;           //RHS Width 5,
violation
out[31:0] = 4 >> in ;                //RHS Width 32, No
violation
out[4:0] = in1[4:0] >> in2[1:0] ; //RHS Width 5, No
violation
out[5:0] = in1[4:0] >> in2[1:0] ; //RHS Width 5,
violation
```

- ❑ For left shift, if shifted or left operand width is matching LHS width then no violation is reported. Consider the following example:

```
out[5:0] = 4'b0001 << in ; //RHS Width 4, Violation
out[31:0] = 4 << in ;      //RHS Width 32, No
```

```

violation
out[5:0] = in1[4:0] << 1 ;           //RHS Width 6, No
violation
out[6:0] = in1[4:0] << 2 ;           //RHS Width 7, No
violation
out[4:0] = in1[4:0] << 2 ;           //RHS Width 5, No
violation
out[7:0] = in1[4:0] << in2[2:0] ; //RHS Width 5,
Violation

```

- ☐ When parameter [nocheckoverflow](#) is set to `yes` then width of left operand is considered for both left shift and right shift operations.

- For Self-Determined Expression

- ☐ For self-determined expressions, the width is calculated as per the LRM:

```

wire a,b,c;
assign a = {b+c};    //LHS: 1, RHS: 1, No violation
assign out[2:0] = {1'b1,b+c}; //LHS: 3, RHS: 2,
Violation

```

- ☐ Behavior of the Tag remains the same when the [nocheckoverflow](#) parameter is set to `yes`

- For Conditional Operator

The width of conditional operator is calculated as follows:

- ☐ A violation is reported if there is a width mismatch either in left expression or in right expression. Consider the following example:

```

out[2:0] = in1[2] ? in2[0] : in3[2:0] ; //RHS Width 1,
Violation

out[2:0] = in1[2] ? in2[2:0] : in3[1:0]; //RHS Width 2,
Violation

out[0]    = in1[2] ? in2[0] : in3[2] ; //RHS Width 1, No
violation

```

- ☐ Behavior remains same when the [nocheckoverflow](#) parameter is set to `yes`.

- For Power Operator

The width of power operator is calculated as follows:

- ❑ If the RHS expression of the power operator is static, then the width of the expression is calculated as per the following formula:

Expression width = LHS expression width * RHS expression value

Consider the following example:

```
out[23:0] = in1[1:0] ** 12; //RHS Width 24, No violation
```

- ❑ If the RHS expression of power operator is non static then the LHS expression width is reported. Consider the following example:

```
out[1:0] = in1[1:0] ** in2[3:0]; //RHS Width 2, No violation
```

- ❑ When the [nocheckoverflow](#) parameter is set to yes, the LHS expression width is reported. Consider the following example:

```
out[23:0] = in1[1:0] ** 12; //RHS Width 2, violation
out[1:0] = in1[1:0] ** in2[3:0]; //RHS width 2, No violation
```

■ For Unbased Unsized Constants

- ❑ No violation is reported if the RHS is unbased unsized constant:

```
assign out[3:0] = 3; //No-Violation
assign out[2:0] = 2'd3; //RHS Width 2, Violation
```

- ❑ Behavior remains same when the [nocheckoverflow](#) parameter is set to yes.

■ For concatenation operator, when the RHS expression is concatenated with zero bits:

- ❑ No violation is reported when the width of the LHS expression lies between the original width of the RHS expression and the width after adding zero concatenated bits. Here, the original width is the width without considering the zero concatenation.
- ❑ When the [nocheckoverflow](#) parameter is set to yes, a violation is reported when LHS width is greater than the RHS width after adding zero concatenated bits.

Message Details

The following message appears at the location where the width `<widthl>` of LHS of an assignment is greater than the width `<widthr>` of the RHS:

LHS width '`<widthl>`' is greater than RHS width '`<widthr>`'
[Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the signal.

Tag Severity

Warning

Suggested Fix

It is more readable to explicitly extend as necessary, rather than relying on default behavior. In case of counters specified as integers with range, fix may not be required as you may have put the check to avoid overflow. This is the normal practice to use integer range for counters.

W242

**This Tag has been deprecated (Verilog)
A function is calling itself; that is, it is recursive (VHDL)**

Language

Verilog, VHDL

Description

Verilog

The W242 Tag has been deprecated. The recursive functions are now synthesizable. In addition, a synthesis error SYNTH_5369 is flagged if a function is unrollable.

VHDL

The W242 Tag flags recursive functions.

Recursive functions have no physical equivalent and may not be synthesized.

Message Details

VHDL

The following message appears at the location of a function declaration *<func-name>* that is recursive:

The function '*<func-name>*' is recursive thus may not be synthesizable;

Tag Severity

Warning (VHDL)

Suggested Fix

Find some other way to implement the function. In general, whatever can be written recursively can be replaced by a loop.

W328

Truncation in constant conversion, without loss of data

Language

Verilog

Description

The W328 Tag flags constant assignments where the left expression is narrower than the right expression but the extra bits in the right expressions are all zeros.

The W328 Tag flags cases where a value with defined width is being assigned to a vector with smaller width. However the truncated bits are all zeros and no data is lost. While these cases are not directly error conditions, they can become error conditions if the constant changes.

Message Details

The following message appears at the location of constant assignment where the left expression width `<widthl>` is narrower than the right expression width `<widthr>` but the truncated bits are all zeros:

Non-significant bits of constant are truncated during assignment (lhs width `<widthl>`, rhs width `<widthr>`) [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the containing scope.

Tag Severity

Warning

Suggested Fix

Consider adjusting the constant so truncation will not occur.

W341

Constant will be 0-extended

Language

Verilog

Description

The W341 Tag flags constant assignments where the size specification of the assigned value is wider than the value specification and the high-order bit/byte is zero. As a result, the constants will be 0-extended.

No Tag checking is done for unused macro definitions and unused parameters.

By default, the W341 Tag considers all type definitions for Tag checking. Set the value of the [ignore_typedefs](#) parameter to yes to ignore all type definitions for Tag checking.

Message Details

The following message appears at the location of the constant assignment where the constant `<const-name>` is extended by zeros to match the size specification:

```
Constant <const-name> is extended by 0; value has fewer bits  
than size specification
```

Tag Severity

Warning

Suggested Fix

It is best to use the correct width and value specification, rather than depending on default extension.

Examples

Consider the following examples:

Specified Value	Is Equivalent to...	Whether Message
8'b0x	8'b00000000x	Yes as the value is 0-extended
9'h0x	9'b000000xxxx	Yes as the value is 0-extended
8'h0x	8'b00000xxxx	No as the value is not 0-extended

W446

Output port signal is being read (within the module)

Language

Verilog

Description

The W446 Tag flags output ports that are read in the module where they are set.

Such models could lead to an unintended feedback path from the instantiating module in the post-synthesis simulation while this issue is not apparent in the pre-synthesis simulation. Such models are also not recommended for some test tools that need to handle inout ports specially (by attaching bus-holders, for example).

NOTE: *Inside the nested `for` loops, signals of user-defined type like structures, interfaces, etc. are considered fully set if used on the left-hand side of an expression and fully read if used on the right-hand side of an expression.*

By default, this Tag does not process large arrays. Set the [handle_large_bus](#) parameter to `yes` to process large arrays (greater than 50,000) and report violation if not used correctly.

The Tag reports violation for all the scenarios by default. Set the value of the [flag_only_instance_ports](#) parameter to `yes` to report scenarios where output port is connected to the input instance pin.

Message Details

The following message appears at the location where an output port `<port-name>` is declared in a module when the port is being both set and read in the module:

Output port '`<port-name>`' is being read inside module.
[Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the hierarchical path of the containing scope.

Tag Severity

Warning

Suggested Fix

If the port really should be an inout port, change the declaration to inout. Otherwise, buffer the signal before the port to guard against possible feedback.

Examples

Consider the following example:

```
module top (out,feedback,in1, in2);  
    input in1,in2;  
    output out;  
    output feedback;  
  
    assign out=in1 & in2 ;  
    assign feedback=out | in2;  
  
endmodule
```

In the above example, the *W446* Tag reports a violation because the output *out* is read in the design instead of setting it. The following message is reported by this example:

Output port 'out' is being read inside module.[Hierarchy:
' :top']

W456

A signal is included in the sensitivity list of a combinational always block but not all of its bits are read in that block (Verilog)
A signal is included in the sensitivity list of a process but not all of its bits are read in that block (VHDL)

Language

Verilog, VHDL

Description

Verilog

The W456 Tag flags signals that are in the sensitivity list of a combinational always construct but are not completely read in the construct.

While such signals do not affect the functionality, they slow down the design simulation. Also, the logic related to a signal in the sensitivity list may have been accidentally removed or not added, which results in wrong functionality. Many simulators re-evaluate the always construct on each change in a redundant signal, even though the evaluation itself is redundant.

By default, the W456 Tag does not report messages for constants present in sensitivity lists. Set the [strict](#) Tag parameter to flag constants also.

By default, the W456 Tag does not report bit information with the messages. Set the [report_all_messages](#) parameter to `yes` or Tag name(s) to report the bit information in the violation messages.

NOTE: *The W456 Tag supports generate-if and generate-for block.*

VHDL

The W456 Tag flags signals that are in the sensitivity list of a process construct but are not completely read in the construct.

While such signals do not affect the functionality, they slow down the design simulation. Also, the logic related to a signal in the sensitivity list may have been accidentally removed or not added, which results in wrong functionality. Many simulators re-evaluate the process construct on each change in a redundant signal, even though the evaluation itself is redundant.

In combinational process constructs, any signal (as a whole or some of its bits) that is not read in the process construct but included in the sensitivity

list, is flagged.

In sequential process constructs, signals other than the clock and the reset signals are not required in the sensitivity list even if they are read in that process construct and hence, are flagged.

Message Details

Verilog

The following message appears at the location of the sensitivity list of a combinational always construct where signal/variable *<name>* is present in the sensitivity list of the always construct but is not completely read in the always construct:

[WARNING] Either all or some of the bits of the signal/variable '*<name>*' are not required in sensitivity list, since they are not read in always block [Hierarchy: '*<hier-path>*']

If the [report_all_messages](#) parameter is set, then the violation message is modified, as follows:

[WARNING] signal/variable '*<name>*' is not required in sensitivity list, since it is not read in always block [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the complete hierarchical path of the containing scope.

VHDL

The following message appears at the location of the sensitivity list of a combinational process construct where a signal or input port *<name>* is present in the sensitivity list of the process construct but is not completely read in the process construct:

[WARNING] The *<type>* '*<name>*' may not be required in sensitivity list as not all of its bits are read in process [Hierarchy: '*<hier-path>*']

Where, *<type>* can be signal or inport and *<hier-path>* is the complete hierarchical path of the containing scope.

The following message appears at the location of the sensitivity list of a sequential process construct where a signal or input port *<name>* is present

but is not required in the sensitivity list of the process construct:

The `<type> ' <name>'` is not required in sensitivity list of the sequential process [Hierarchy: '`<hier-path>`']

Where, `<type>` can be signal or in port and `<hier-path>` is the complete hierarchical path of the containing scope.

Tag Severity

Warning

VHDL Examples

Consider the following example of a sequential process construct:

```
library IEEE;
use IEEE.std_logic_1164.all;

test: process(clk, rst, in1)
begin
    if (rst = '0') then
        out1 <= "0000";
    elsif (clk'event and clk = '1') then
        out1 <= in1;
    end if;
end process test;
```

While signal `in1` is being read in the sequential process construct, it is not the clock signal or the reset signal and hence, is not required in the sensitivity list.

Suggested Fix

Verilog

This is partly a question of taste. If you specify the whole bus and some bits are not read in the block, you may get a warning that the sensitivity list is over-specified. In these cases, it is not possible to satisfy both requirements simultaneously, so you must decide which requirement you want to ignore. From an implementation point of view, it is safest to ignore this Tag.

VHDL

Remove variables from the sensitivity list if they are not required. This will help improve simulation performance. It will have no effect on synthesis behavior.

W488

A bus variable appears in the sensitivity list but not all bits of the bus are read in the contained block (Verilog)
An array signal appears in the sensitivity list but not all bits of the array are read in the process (VHDL)

Language

Verilog, VHDL

Description

Verilog

The W488 Tag flags the bus variables that appear in the sensitivity list but not all bits of that bus are read in the contained block.

While such designs are allowed, they may impact simulation performance. Many simulators re-evaluate the block on each change in each bit, even though the evaluation is redundant.

NOTE: *The W488 Tag supports generate-if and generate-for block.*

VHDL

The W488 Tag flags those arrays that appear in the sensitivity list but all elements of the arrays are not read in the process.

While such designs are allowed, they may impact simulation performance. Many simulators re-evaluate the process on each change in each element of an array in the sensitivity list, even though the evaluation is redundant.

Parameter(s)

[*traverse_fanin_for_src_net*](#): Default value is 0. Set this parameter to 1 to stop the fanin of synchronous reset to detect flattened source reset.

Message Details

Verilog

The following message appears at the location of the contained block's sensitivity list that contains a bus/memory variable, *<var-name>*, and all the bits/words of that variable are not read in the contained block:

Bus/Memory '*<var-name>*' (or some of its bits/words) which is not read in always block is not required in the sensitivity list [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the complete hierarchical name of the containing scope excluding subprograms.

VHDL

The following message appears at the location of a process' sensitivity list that has an array *<array-name>* of type *<array-type>* and all array elements are not read in the process scope (hierarchy *<hier-path>*):

Array *<array-type>* '*<array-name>*' is in the sensitivity list but not all its bits are read inside the process [Hierarchy: '*<hier-path>*']

The *<array-type>* can be signal, port, or variable and *<hier-path>* is the complete hierarchical name of the containing scope excluding subprograms.

Severity

Warning

Suggested Fix

Verilog

There is a trade-off in fixing this Tag versus [W122](#), [W456](#) and [W456a](#) Tags. It is recommended to clear W122, W456 and W456a Tags and waive or ignore the W488 Tag.

VHDL

Remove variables from the sensitivity list if they are not required. This will help improve simulation performance without effecting synthesis behavior.

W494a

Input port is not used

Language

VHDL

Description

The W494a Tag flags input ports that are never read in the design.

While such ports are allowed, the W494a Tag helps you clean up your design.

By default, the Tag reports violation for the completely unused input ports. Set the value of the [strict](#) parameter to yes to also report violation for partially unused ports.

Message Details

The following message appears at the location where an input port `<port-name>` is declared that is never set in the design:

The in port '`<port-name>`' is not used

Severity

Warning

Suggested Fix

Determine if the input port is really not required. If not, consider removing the port.

W494b

Output port is not used

Language

VHDL

Description

The W494b Tag flags output ports that are never set in the design.

While such ports are allowed, the W494b Tag helps you clean up your design.

By default, the Tag reports violation for the completely unused output ports. Set the value of the [strict](#) parameter to *yes* to also report violation for partially unused ports.

Message Details

The following message appears at the location where an output port `<port-name>` is declared that is never set in the design:

The out port '`<port-name>`' is not used

Severity

Warning

Suggested Fix

Determine if the output port is really not required. If not, consider removing the port.

W500

A connection to a clock port is not a simple name

Language

VHDL

Description

The W500 Tag flags clock ports of component instances that use bus, indexed-name, sliced-name, selected-name, expression, or concatenation. It is recommended to use simple names as clocks.

Message Details

Following message appears at the location where the connection to clock port *<clk-name>* of an instantiation is not a simple name:
Connection to *<clock>* port '*<clk-name>*' is not a simple name

Severity

Warning

Examples (VHDL)

In the following examples, the connection to clock port `clk` is not a simple name:

```

library IEEE;
use IEEE.std_logic_1164.all;

-- Bit-select connected to clock port
INST: test port map
    (d => s1, clk => input(0), q => output);

-- Expressions connected to clock port
INST: test port map
    (d => input(0), clk => (c and d), q => output);

-- Constants connected to clock port

```

```
INST: test port map  
      (d=> input(0), clk=>('0' XOR '1'), q=> output);
```


W526

Use case statements rather than if/else, where feasible, if performance is important

Language

Verilog, VHDL

Description

Verilog

The W526 Tag flags those if-else constructs that should be changed to case constructs to improve simulation performance.

The W526 Tag assumes the following:

- If the condition expressions of a nested if-else construct test the same signal *n* times, the nesting level of the construct is assumed to be *n*.
- A signal is assumed to be tested if the signal and only the signal appears on the LHS of a comparison operation (`==`, `<=`, `>=`, `!=` etc.).

The W526 Tag flags an if-else construct only if the nesting level is more than 3, and the same value is being tested in each branch. You can change the nesting level (default limit of 3) to any required value by changing the Tag definition given in the PERL Tagdeck file.

A chain of if-else statements on the same test signal is inferred as a priority encoder. The same logic build using case statement is inferred as a multiplexer, which is likely to simulate faster than a priority encoder.

Using a case construct instead of if-else construct is meaningful only if the same value is being tested in each branch of the if-else construct.

VHDL

The W526 Tag flags those if-then-else constructs that should be changed to case constructs to improve simulation performance.

A chain of if-then-else statements on the same test signal is inferred as a priority encoder. The same logic build using case statement is inferred as a multiplexer, which is likely to simulate faster than a priority encoder.

Using a case construct instead of if-then-else construct is meaningful only if the same value is being tested in each branch of the if-then-else construct.

The W526 Tag flags an if-then-else construct only if the nesting level is more

than 3, and the same value is being tested in each branch.

Message Details

Verilog

The following message appears at the first line of a Tag-violating if-else construct:

Case statement is recommended over deeply nested 'if' statement

VHDL

The following message appears at the first line of an if-then-else construct for condition clause *<name>* if the number of branches is more than 3:

Deeply nested if statement (condition '*<name>*') could be rewritten as a case statement

Tag Severity

Guideline

Suggested Fix

Generally it is best to recode deeply nested if statements as case statements unless you actually need a priority encoder or the logic is not in a performance-critical part of the circuit.

Examples (Verilog)

Consider the following if-else construct that has four levels of nesting as the same signal *sel* is being compared four times in the conditional expressions:

```
if (sel == 4'h0 )
    out = (in1 + int1) - P1;
else if (sel == 4'h1 )
    out = in2 + (int2 - P2);
else if (sel == 4'h2 )
    out = ((in3 + int1) - `M1) + (in2 + (int2 - P2));
else if (sel == 4'h3 )
    out = (data1 + (int2 - `M2));
else
```

```
out = int2;
```

For this example, SpyGlass generates the W526 Tag message.

Examples (VHDL)

Consider the following example if-then-else construct that has more than 3 branches and the same value (x) is being tested in each branch:

```
library IEEE;
use IEEE.std_logic_1164.all;

if ( x = "000" ) then y <= a and b;
elsif ( x = "001" ) then y <= a or b;
elsif ( x = "010" ) then y <= a nand b;
elsif ( x = "011" ) then y <= a xor b;
else y <= '0';
end if;
```

For this example, SpyGlass generates the following message:

Deeply nested if statement (condition 'x') could be rewritten as a case statement

To fix this violation, you can rewrite the above example using the case construct as follows:

```
case (x) is
  when "000" => y <= a and b;
  when "001" => y <= a or b;
  when "010" => y <= a nand b;
  when "011" => y <= a xor b;
  default   => y <= '0';
end case;
```

W529

``ifdef` is not supported by all tools

Language

Verilog

Description

The W529 Tag flags ``ifdef` compiler directives used in the design.

Some synthesis tools do not support preprocessor conditional directives.

Message Details

The following message appears at the location where a ``ifdef` compiler directive is encountered:

Compiler directive ``ifdef` is not supported by all synthesis tools

Tag Severity

Warning

Suggested Fix

Avoid using these directives in synthesizable logic.

W128

Avoid using negative delays

Language

Verilog, VHDL

Description

The W128 Tag flags negative delay values.

A negative delay involves a step backward in time which is at least confusing (and therefore error-prone) and may cause efficiency problems in some simulators. At most, negative delays should be used only to model special behavior in cell libraries.

Message Details

The following message appears at the location where a negative delay value is encountered:

Negative delay <value> specified

where, <value> is a delay expression.

Tag Severity

Warning

Suggested Fix

Try to re-code the logic and use either zero or positive delays only.

W129

Variable delay values should be avoided

Language

Verilog

Description

The W129 Tag flags non-constant delay values.

While it is not a fundamental restriction, debugging in the presence of variable delays dramatically increases complexity and the opportunity for error.

Message Details

The following message appears at the location where a non-constant delay value is encountered:

variable delay values <value> should be avoided
where, <value> is a delay expression.

Tag Severity

Warning

Suggested Fix

If possible, avoid using variable delay values.

Examples

Consider the following example:

```
module top (out,in);  
    input in;  
    output reg out;  
    integer i=0;  
    always @(in)  
    begin  
        out = #i in;  
    end
```

Lint Tags

```
endmodule
```

In the above example, the *W129* Tag reports a violation because the delay value *i* is not a constant. The following message is reported by this example:

```
variable delay value 'i' should be avoided
```

W143

Macro has been redefined

Language

Verilog

Description

The W143 Tag flags macros redefinitions in the same file.

Redefining a macro in the same file in which it was originally defined, is likely to cause confusion and limit readability.

The W143 Tag also flags a violation if a macro is defined in a file, and then the same file is included in a separate file where the macro is redefined.

Consider an example in which a macro, M1, is defined in a file, F1, and F1 is included in a separate file, F2. If you redefine the macro, M1, in F2, this Tag will report violation. These apparent errors can be ignored.

Parameters

[waiver_compat](#): Default value is no. If you set the value of this parameter to yes or <Tag-name>, it ensures that the Tag does not generate the line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.

Message Details

The following message appears, at the location where a macro *<macro-name>* is redefined, providing details of the line number *<line-num>* and the file name *<file-name>* in which the macro was previously defined:

```
Redefining macro '<macro-name>'. Previously defined at line  
'<line-num>' in file '<file-name>'
```

Tag Severity

Warning

Suggested Fix

If the macro is redefined with a different value, rewrite the code to use a different macro in the second case. Redefining the same macro with a different value can make the code error-prone.

W146

Use named-association rather than positional association to connect to an instance

Language

Verilog, VHDL

Description

The W146 Tag flags module or component instantiations where the port association is by position.

Using positional association significantly increases the possibility of mis-connects, some of which may prove very subtle and difficult to find. Using named association completely eliminates this possibility. Also, other users do not have to cross-reference to the definition of the master of an instance to understand signal connections.

By default, the W146 Tag does not report violation for parameters in instantiation, which are positional association. Set the value of the [check_param_association](#) parameter to `yes` to check for parameters in instantiation.

Message Details

Verilog

The following message appears at the location of a module instantiation where the port association is by position:

Explicit named association is recommended in instance references

VHDL

The following message appears at the location of a component instantiation where the port association is by position:

Explicit named association is recommended in instance port/
generic map

Tag Severity

Guideline

Suggested Fix

In Verilog designs, if this is a top-level testbench, there is nothing to fix. Otherwise you can only read to and write from this design unit through global variables, which is unsynthesizable. May sometimes be OK for simulation monitors (though would still be considered by many to be poor coding style).

For VHDL designs, rewrite all instances using named-association connections.

W154

Do not declare nets implicitly

Language

Verilog

Description

The W154 tag flags implicit net declarations.

While implicit net declarations may be convenient for handling unconnected (don't care) outputs, they can also result through misspelling of net names which are intended to be connected.

Running the W154 tag ensures that all implicit declarations are flagged. if you require all nets to be declared, the w154 tag effectively becomes a spell-check for net connections; any message found represents a true error.

Parameter(s)

[*report_port_net*](#): Default value is no. Set the value of the parameter to yes to report nets/ports declared without a type.

Message Details

The following message appears at the location where a net *<net-name>* is used that is not explicitly declared:

declare net '*<net-name>*' explicitly

Tag Severity

Guideline

Suggested Fix

Explicitly declare all nets found by this Tag (after checking they are not a result of spelling errors).

W156

Do not connect buses in reverse order

Language

Verilog, VHDL

Description

Verilog

The W156 Tag flags reverse connected buses.

Making reversed connections (for example, `15 : 0` connected to `0 : 15`) is legal but bad design practice and may represent an error.

One exception (which can be handled with a waiver) is in making a `big-endian/little-endian` choice in connecting to a processor bus port.

VHDL

The W156 Tag flags reverse connected buses.

Making reversed connections (for example, `15 downto 0` connected to `0 to 15`) is legal but bad design practice and may represent an error.

One exception (which can be handled with a waiver) is in making a `big-endian/little-endian` choice in connecting to a processor bus port.

For a constant declaration that has its subtype indication as an unconstrained array, the direction of the index constraint is the same direction of the index subtype definition used in the declaration of the unconstrained array. For example,

```
type MYBIT_VEC is array (natural range <>) of BIT;
...
constant myconst : MYBIT_VEC := "1011";
```

Here, the direction of `myconst` is same as that of `natural` that is `to`.

By default, the W156 Tag does not report the reverse connections involving static expressions. Set the *strict* Tag parameter to report such reverse connections also.

Parameter(s)

- [*report_all_connections*](#): Default value is `no`. Set the value of this parameter to `yes` to report a violation for all reverse order buses in the same module.
- [*ignore_concat_expr*](#): Default value is `no`. Set the value of this parameter to `yes` to not report violations for the reverse connections which is an expression that concat operation is involved.

Message Details

Verilog

The following message appears at the location of a module instantiation where the port association of bus *<sig-name>* is in reverse order:

Bus net '*<sig-name>*' is connected in reverse. [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the complete hierarchical name of the containing scope excluding subprograms.

VHDL

The following message appears at the location where the port association of bus *<sig-name>* is in reverse order:

<sig-name> contains bus connection in reverse order

Tag Severity

Warning

Suggested Fix

Where possible, connect in the same order as defined. In what should be localized big-endian/little-endian hookup cases, use a waiver.

Examples (Verilog)

In the following example, the module instance `lower1` has the bit-slice `sel[0:1]` is connected in reverse order than the port definition in the parent module `lower`:

Lint Tags

```
module test(i1, i2, o1, o2);
  input [3:0] i1, i2;
  output [0:3] o1, o2;

  wire [0:5] sel;

  lower lower1(i1, i2, sel[0:1], o1, o2);
endmodule

module lower(in1, in2, sel, out1, out2);
  input [3:0] in1, in2;
  inout [1:0] sel;
  output [0:3] out1, out2;

  reg [0:3] out1, out2;

  always@(sel)
    case(sel)
      2'b00: out1 <= in1;
      2'b01: out1 <= in2;
      2'b10: out2 <= in1;
      2'b11: out2 <= in2;
    endcase
endmodule
```

For this example, VC SpyGlass Lint generates the following message:
Bus net 'sel' is connected in reverse. [Hierarchy: ':test']

W159

Condition contains a constant expression

Language

Verilog

Description

The W159 Tag flags control expressions that evaluate to a constant.

The W159 Tag flags constant control expressions used in `if` statement conditions, loop conditions, and conditional statements.

Constant control expression or sub-expression may result in an always ON or always OFF logic inference. A condition with a constant expression is usually a mistake, except where the condition represents a hard-wire configuration parameter.

By default, the value of the [*ignore_cond_having_identifier*](#) is set to `no` and this Tag considers constants defined by parameters, localparams or constant integers, for Tag checking. Set the value of this parameter to `yes` to not report violation for such constants.

Message Details

The following message appears at the location where the control expression `<expr>` evaluates to a constant:

Constant expression `<expr>` in condition [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the containing scope.

Tag Severity

Warning

Suggested Fix

There may not need to be a fix as long as you are aware that portion of your logic may be permanently disabled/optimized out as a result

Examples

Following examples show constant control expressions where VC SpyGlass Lint generates the W159 Tag message:

```
if(1'b1)
...
`define cond 1'b1
...
while(`cond)
...
out1 <= 1'b0 ? in1: in2;
```

W163

Truncation of bits in constant integer conversion

Language

Verilog

Description

The W163 Tag flags constant integer assignments to signals when the width of the signal is narrower than the width of the constant integer.

When assigning a constant integer value to a LHS operand, the width specification for RHS should match LHS operand width. If the signal is wider than the constant integer, the extra bits are padded with zeros. If the signal is narrower than the constant integer, the extra high-order non-zero bits are discarded.

Message Details

The following message appears at the location where a constant integer (width `<widthr>`) is assigned to a signal (width `<widthl>`) that is narrower than the constant integer:

Significant bits of constant will be truncated (lhs width `<widthl>`, rhs width `<widthr>`) [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the signal.

Tag Severity

Warning

Suggested Fix

Adjust the constant so you assign a smaller constant which will not be truncated.

W164a

Identifies assignments in which the LHS width is less than the RHS width

When to Use

Use this Tag to identify assignments in which the LHS width is less than the RHS width.

Description

Verilog

The tag reports violations for assignments in which the width of LHS is less than the (implied) width of the RHS expression.

The Tag determines the assignment type of a binary expression as per the following table:

Condition	Assignment Type of a Binary Expression
LHS < RHS	Expression type of RHS
LHS > RHS	Expression type of LHS
LHS = RHS	The expression type is same as that of the LHS or the RHS type, which can be one of the following: reg, wire, real, realtime, time, integer, static, constant integer, Boolean

Width Calculation for Verilog

By default, the width is calculated considering the best fit width of an expression. That is the width in which maximum value of an expression can be accommodated. When you set the value of the [nocheckoverflow](#) parameter to `yes` then the width is calculated according to the LRM and natural width is considered for constants.

The behavior of the Tag is explained by the following cases.

- For Constant Expressions (with the value of the [nocheckoverflow](#) parameter set to `no`):
 - ☐ For Constant integer expressions, the width is calculated based on the value of the expression:

```

out[3:0] = 2 + 15 ;    //RHS Width = 5 (Value = 17)
out[6:0] = 100 << 4 ; //RHS Width = 11
out[4:0] = 100 >> 2 ; //RHS Width = 5

```

- ❑ If constant is not a part of the sub-expression then the specified width is considered:

```
out[11:0] = 12'b0 ; //RHS Width = 12
```

- ❑ If the width of a based number is not specified then the natural width is considered:

```
out[8:0] = 'h344;      //RHS Width = 10
```

For above cases, behavior of the Tag remains the same when [nocheckoverflow](#) parameter is set to `yes`.

■ For Arithmetic Operators

- ❑ The width addition operator (with the value of the [nocheckoverflow](#) parameter set to `no`), is calculated as a width where maximum value of an expression is accommodated. Consider the following example:

```
out[1:0] = in1[1:0] + in2[1:0] + in3[1:0]; //RHS
Width 4
```

Max value RHS $3 + 3 + 3 = 9$

RHS Width = 4

```
out[2:0] = in1[1:0] + (3/3) ; //RHS Width 3
```

Max Value RHS = $3 + 1 = 4$

RHS Width = 3

```
out[5:0] = in1[5:0] + 5'b1010; //RHS Width 7
```

```
out[4:0] = 15 + 3'b111;      //RHS width 5
```

- ❑ For the subtraction operator (with the value of the [nocheckoverflow](#) parameter set to `no`), the width of the expression is calculated based on the minimum and maximum values of the expression:

```
out[3:0] = in1[3:0] - in2[1:0]; //Min value is -3 (width is 3), Max
value is 15 (width is 4), Therefore, RHS Width 4
```

```
out[3:0] = in1[3:0] - 4'b1010; //Min value is -10 (width 5), Max
value is 5 (width is 3), Therefore, RHS Width 5
```

- ❑ For the multiplication operator (with the value of the *nocheckoverflow* parameter set to no), the width is calculated as follows:

When both operands are variable, then the RHS width is the sum of the width of both operands:

```
out[6:0] = in1[1:0] * in2[1:0] * in3[1:0]; //RHS Width 6
out[2:0] = in1[2:0] * in1[1:0];           //RHS Width 5
```

When one operand is static and other is variable, then the RHS width is calculated considering maximum value of the expression:

```
out[3:0] = in1[3:0] * 4'b1; //RHS Width 4, Max value = 15*1 = 15
out[3:0] = in1[3:0] * 4'b10; //RHS Width 5, Max
value = 15*2 = 30
out[7:0] = in1[3:0] * 4'b1010; //RHS Width 8, Max value = 15*10 =
150
out[3:0] = in1[3:0] * 4; //RHS Width 6, Max
value = 15*4 = 60
```

- ❑ For the division operator (with the value of the *nocheckoverflow* parameter set to no), the width of the RHS is assumed as the width of left operand:

```
out[2:0] = in1[2:0]/in1[1:0]; //RHS Width 3
out[1:0] = in1[1:0]/in1[2:0]; //RHS Width 2
out[3:0] = in1[3:0]/4'b10; //RHS Width 4
out[2:0] = 4/in1[3:0]; //RHS Width 3
```

- ❑ When the *nocheckoverflow* parameter is set to yes, the width is assumed to be the same as the width of term having maximum width. Consider the following example:

```
out[1:0] = in1[1:0] + in2[1:0] + in3[1:0]; //RHS Width = 2
out[5:0] = in1[5:0] + 5'b1010; //RHS Width 6
out[2:0] = 15 + 3'b111; //RHS width 4
out[3:0] = in1[3:0] - in1[1:0]; //RHS Width 4
out[2:0] = in1[3:0] - 4'b1010; //RHS Width 4
out[2:0] = in1[2:0] * in1[1:0]; //RHS Width 3
out[3:0] = in1[3:0] * 4'b10; //RHS Width 4
out[1:0] = in1[1:0]/in1[2:0]; //RHS Width 3
```

```
out[2:0] = 4/in1[3:0];           //RHS Width 4
```

■ For Shift Operators

- For the right shift (with the value of the *nocheckoverflow* parameter set to no), if left operand width is matching the LHS width then no violation is reported. Consider the following example:

```
out[3:0] = in1[3:0] >> in ;      //RHS Width 4
out[2:0] = in1[4:0] >> 2 ;        //RHS Width 5
out[31:0] = 4 >> in ;            //RHS Width 32
out[4:0] = in1[4:0] >> in2[1:0] ; //RHS Width 5
out[3:0] = in1[4:0] >> in2[1:0] ; //RHS Width 5
```

- For the left shift (with the value of the *nocheckoverflow* parameter set to no), if shifted or left operand width is matching the LHS width, no violation is reported. Consider the following example:

```
out[2:0] = 4'b0001 << in ;       //RHS Width 4
out[2:0] = 4 << in ;             //RHS Width 32
out[4:0] = in1[3:0] << 1 ;        //RHS Width 5
out[6:0] = in1[4:0] << 2 ;        //RHS Width 7
out[3:0] = in1[4:0] << in2[2:0] ; //RHS Width 5
```

- When the *nocheckoverflow* parameter is set to yes, the width of the left operand is considered for both left shift and right shift operations.

■ For Self-Determined Expressions

- The width of a self-determined expression (with the value of the *nocheckoverflow* parameter set to no), is calculated according to the LRM. Consider the following example:

```
wire a,b,c;
assign a = {b+c};    //LHS: 1, RHS: 1
assign b = {1'b1,b+c}; //LHS: 1, RHS: 2
```

- The behavior of the Tag remains the same when *nocheckoverflow* parameter is set to yes.

■ For Conditional Operators

The width of a conditional operator (with the value of the *nocheckoverflow* parameter set to no), is calculated as follows:

- ❑ A violation is reported if there is width mismatch either in the left expression or in the right expression. Consider the following example:

```
out[0] = in1[2] ? in2[0] : in3[2:0] ;    //RHS Width 3
out[1:0] = in1[2] ? in2[2:0] : in3[1:0]; //RHS Width 3
out[0] = in1[2] ? in2[0] : in3[2] ;    //RHS Width 1
```

- ❑ The behavior of the Tag remains same when *nocheckoverflow* parameter is set to yes.

■ For Power Operators

The width of a power operator (with the value of the *nocheckoverflow* parameter set to no), is calculated as follows:

- ❑ If the RHS expression of the power operator is static, then the width of the expression is calculated as per the following formula:

Expression width = LHS expression width * RHS expression value

Consider the following example:

```
out[1:0] = in1[1:0] ** 12;    //RHS Width 24
```

- ❑ If the RHS expression of a power operator is non-static then the LHS expression width is reported:

```
out[1:0] = in1[1:0] ** in2[3:0]; //RHS Width 2
```

- ❑ When the *nocheckoverflow* parameter is set to yes then the LHS expression width is reported:

```
out[1:0] = in1[1:0] ** 12;    //RHS Width 2
out[1:0] = in1[1:0] ** in2[3:0]; //RHS width 2
```

■ For concatenation operator (with the value of the *nocheckoverflow*

parameter set to no), when the RHS expression is concatenated with zero bits:

- ❑ No violation is reported when the width of the LHS expression lies between the original width of the RHS expression and the width after adding zero concatenated bits. Here, the original width is the width without considering the zero concatenation.

- ❑ When the [nocheckoverflow](#) parameter is set to yes, a violation is reported when the LHS width is less than the RHS width after adding zero concatenated bits.

NOTE: *The Tag does not report violation if the RHS is numeric constant zero or any expression multiplied by a numeric constant zero or a based number zero. For example, violation will not be reported if the RHS is either of 0, 4 * 0, 4 * 1'b0, w1 * 4'b0, or (w1+w2) * 0.*

While comparing width of assign statements inside the `for` loops, width of the `for` loop variable is limited to the minimum width, which is enough to occupy values of the `for` loops range (from initial value to condition value). This limiting happens when only the following conditions are true, otherwise the actual loop variable width is considered for comparisons:

- Loop initial value and condition value should be static expressions
- Loop variable condition should be a simple one with one comparison operator

```
i >= <static expr>
<static_expr> < j
```

This does not work for the following examples:

```
(i < <static expr>) && (j < <static expr>)
(i < <static expr>) || (j < <static expr>)
```

- Same loop variable/variable expression with same bit selection as used in `for`-loop condition should be present in RHS of assignment statement

Example 1

```
for (integer j=31; j>=0; j=j-1)
wr_domain_sel[4:0] = j; // No violation because width
(width sufficient to occupy loop range) of loop variable
j is 5
```

Example 2

```
reg j [31:0] j;
for (j[15:0]=31; j[15:0]>=0; j[15:0]=j[15:0]-1)
wr_domain_sel[4:0] = j[15:0]; // No Violation because
```


width (width sufficient to occupy loop range) of loop variable `j[15:0]` is 5

Will not work for following examples

```
reg [31:0]
  for (j[7:0]=31; j[7:0]>=0; j[7:0]=j[7:0]-1)
    wr_domain_sel[4:0] = j[15:0]; // Violation because
    j[15:0] is not a loop variable and so its width (actual
    width) is 16
```

VHDL

The *W164a* Tag reports violations for assignments in which the width of LHS is less than the (implied) width of the RHS expression.

The *W164a* tag does not report violations for assignments in which the width of the LHS lies between the zero padded and non-zero padded width, provided the RHS of the assignment is an arithmetic expression on the two zero padded concatenation expressions.

For VHDL designs, the Tag does not check the function or procedure bodies as the size of the arguments depends on the actual values passed.

In case of all integer types where ranges are specified, the Tag works in a different manner. Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

signal A1 : integer range -100 to 100 ;
signal A2 : integer range -100 to 100 ;
signal A3 : integer range -199 to 199 ;
signal A4 : integer range -200 to 200 ;
A3 <= A1 + A2 ; -- violation
A4 <= A1 + A2 ; -- No violation
```

In the first assignment, the range of the RHS expression `A1+A2` is from -200 to 200 but the range of the LHS expression `A3` is from -199 to 199, which is less than the range of the RHS expression. Hence, the Tag reports a violation.

In the second assignment, the range of the RHS expression A1+A2 is from -200 to 200, which is equal to the range of the LHS expression A4. Hence, the Tag does not report a violation.

NOTE: *If any operand in the RHS has a non-integer type of range, then the bit-width is compared as per the package, not the range.*

Assignment Types

The Tag handles the different assignment types as follows:

- `<Integer_range> <= <Integer_range> OP`
`<Integer_range>`

The RHS range is evaluated and then compared with the LHS range.

- `<Non_integer_range> <= <Non_integer_range> OP`
`<Non_integer_range>`

The bit-width for the RHS and the LHS is calculated as per the package.

- `<Non_integer_range> <= <Integer_range> OP`
`<Integer_range>`

The RHS range is evaluated and converted to bit-width, which is then compared with the bit-width of the LHS.

- `<Integer_range> <= <Non_integer_range> OP`
`<Non_integer_range>`
- `<Non_integer_range> <= <Integer_range> OP`
`<Non_integer_range>`
- `<Integer_range> <= <Integer_range> OP`
`<Non_integer_range>`

The bit-width of the RHS is calculated from the package and then compared with the bit-width of the LHS.

An integer type without the range specified is treated as `<Non_integer_range>` and bit-width is calculated instead of range.

Width Calculation for VHDL

Set the [nocheckoverflow](#) parameter to yes or *W164a* to calculate the width according to the LRM (numeric_std lib) as per the following cases:

- For the addition and subtraction operators, the width is calculated based on the following Tags:
 - If both the operands are variables, then the max width is taken.
 - If one operand is a variable and the other is a static, then the width of the variable is taken.
- For the multiplication operator, the width is calculated based on the following Tags:
 - If both the operands are variables, then the RHS width is the sum of both the variables.
 - If one operand is a variable and the other is a static, then the RHS width is $2 * (\text{Variable width})$.
- For the division operator, the width is calculated based on the following Tags:
 - If both the operands are variables, then the RHS width is the width of the left operand.
 - If one operand is a variable and the other is a static, then the RHS width is the width of the variable.
- For concatenation operator, no violation is reported when the RHS expression is concatenated with zero bits and the width of the LHS expression lies between original width of the RHS expression and width after adding zero concatenated bits. Here, original width is the width without considering the zero concatenation.

NOTE: For constant arrays, the declared width is considered when the [nocheckoverflow](#) parameter is set to yes. Whereas by default, the maximum width among the initialized elements is considered. For more, refer to the VHDL [Example 11](#).

NOTE: The *W164a* Tag supports generate-if and generate-for blocks.

Language

Verilog, VHDL

Parameter(s)

- [check_lrm_and_natural_width](#): Default value is no. Set the value of the parameter to yes to check for both LRM and natural widths before reporting violations.
- [check_static_natural_width](#): Default value is no. Set the value of the parameter to yes to enable the specified tag to check for natural width

of a static expression even when the value of the *nocheckoverflow* parameter is set to yes.

- ***check_static_value***: Default value is no. Set the value of the parameter to yes or <Tag_list> to report violation for all cases with width mismatch, involving static expressions and non-static expressions having a static part. You can also set the value of the parameter to only_static to ignore violations for expressions, which do not have a static part. Other possible values are only_const and only_expr.
- ***disable_rtl_deadcode***: The default value is no. Set the value of the parameter to yes to disable violations for disabled code in loops and conditional (if condition, ternary operator) statements.
- ***ignoreInteger***: By default, the value of the parameter is no. Set the value of the parameter to yes to not report a violation for integer index.
- ***sign_extend_func_names***: The default value is "EXTEND". Set the value of the parameter comma-separated list of function names to enable the W164a Tag to recognize VHDL sign extension functions and calculate width of extend functions as per the const extension argument specified in the argument list.
- ***strict***: Default value is no. For **Verilog**, by default, the tag reports violation if the RHS expression contains wire or reg. Set this parameter to yes to report all assignments. Also, the tag reports integer port signals in the RHS expression, in this case.

For **VHDL**,

- ☐ Set the value of the parameter to yes to report violation for addition and subtraction expression, if any one operand is a constant and its width is lesser.
- ☐ Set the value of the parameter to no to report violations for the following scenarios:
 - ◆ Maximum range of the RHS is greater than the maximum range of the LHS.
 - ◆ Minimum range of the RHS is lesser than the minimum range of the LHS.
 - ◆ RHS value exceeds the maximum LHS range or if it is lesser than the minimum LHS range.

- [*use_lrm_width*](#): Default value is `no`. Set this parameter to `yes` to consider the LRM width of integer constants, which is 32 bits.
- [*use_carry_bit*](#): By default, when the `nocheckoverflow` parameter is set `no`, the width of addition and subtraction is considered as the width of the maximum value that the expression can hold. Set the value of this parameter (with default value as `W164a`) to `no` to calculate the width as the LRM, that is the maximum width of the operand.
- [*nocheckoverflow*](#): Default value is `no`. Set this parameter to `yes` or Tag name to check the bit-width as per LRM. Other possible value is the Tag name.
- [*checkOperatorOverload*](#): Default value is `yes`. Set this parameter to `no` to evaluate width of the expression without considering overloaded operators. This parameter is applicable for **VHDL** only.
- [*check_counter_assignment*](#): Default value is `no`. Set this parameter to `yes` to report a violation for the counter type of assignments. This parameter is applicable for both Verilog and VHDL in the *W164a* Tag.

NOTE: For details about the cases that are considered as counter cases in Verilog, see the [*check_counter_assignment*](#) parameter section.

- [*check_unsign_overflow*](#): Default value is `no`. This indicates the Tag suppresses the overflow when sign extension is used for unsigned signals in addition or subtraction operation. If you set this parameter to `yes` or `W164a`, the Tag does not suppress the overflow and reports a violation when sign extension is used for unsigned signals in addition or subtraction operation.
- [*check_concat_max_width*](#): Default value is `no`. In this case, no violation is reported when the width of the LHS expression is present between the width of the RHS expression without considering zero concatenated bits and the width of the RHS after adding zero concatenated bits. If you set this parameter to `yes`, the RHS width is considered as the width after adding zero concatenated bits. That is, the violation is reported if the LHS width does not match the RHS width after adding zero concatenated bits. This parameter is applicable for **Verilog** only.
- [*concat_width_nf*](#): Default value is `no`. Set the value of the parameter to `yes` to specify if the `W164a` Tag should use new algorithm to calculate the width of concatenations ignoring the self-determined nature of concatenation items.

- [*handle_shift_op*](#): Default value is `no`. In this case, no violation is reported if the shifted or non-shifted width of a shift expression matches the LHS width of an assignment. But the Tag does not calculate the shifted width, if the RHS of the shift expression is non-static. Set this parameter to `shift_left`, `shift_right`, `shift_both`, `no_shift`, `no_shift_forced`, or comma-separated list of Tag names, to compare shifted or non shifted widths for left and right shift expressions.
- [*datapath_or_control*](#): Default value is `no`. In this case, and the Tag reports violations for all type of signals. Set this parameter to `datapath`, `control`, or `all` to specify the type of signals to be checked.
- [*force_handle_shift_op*](#): Default value is `no`. Set this parameter to `yes` to enable the Tag to honor [*handle_shift_op*](#) parameter value even if the [*nocheckoverflow*](#) parameter is set to `yes`.
- [*control_sig_detection_nf*](#): Default value is `no`. If you set this parameter to `yes`, the Tag treats a signal to be a control signal only if it is used in a conditional operator or control binary operator within the scope of currently processing statement. The Tag does not consider the signal to be a control signal even if it is used as a control signal in other places of the module (that is, the statements other than the current statement).
- [*handle_lrm_param_in_shift*](#): Default value is `no`. Set this parameter to `yes` to enable the *W164a* Tag to honor the [*use_lrm_width*](#) parameter in the width calculations of shift expressions.
- [*handle_zero_padding*](#): Default value is `no`. Set the value of the parameter to `yes` or tag name to perform leading zero expansion and truncation of RHS and LHS of an assignment. This is performed only if the RHS and LHS of the assignment are static. In addition, if the parameter is set to `yes`, the tag performs zero expansion or truncation if the RHS of the assignment is a concatenation and the first element is padded with zero.
- [*process_complete_condop*](#): Default value is `no`. Set the value of the parameter to `yes` to enable the Tag checking on both operands of the `condop` assignment.
- [*treat_concat_assign_separately*](#): Default value is `no`. Set the value of the parameter to `yes` to use this parameter to report violation for each bucket assignment in unpacked array separately. For a packed array, a violation is reported for the whole array.

- [*check_natural_width_of_div*](#): Default value is no. Set the value of the parameter to yes to calculate the natural width of the division operator when it's divisor is static, regardless of the value of the *nocheckoverflow* parameter.
- [*check_natural_width_of_modulo*](#): Default value is no. Set the value of the parameter to yes to calculate the natural width of the modulo operator, regardless of the value of the *nocheckoverflow* parameter.
- [*check_natural_width_of_multiplication*](#): Default value is no. Set the parameter to yes, to calculate the width of the multiplication expressions as per the natural width even if the *nocheckoverflow* parameter is set to yes. Set the parameter to a comma/ space-separated list of tags to enable (the parameter) for the tags specified in the list.
- [*check_natural_width_for_static*](#): Default value is no. By default, the tag reports violations for static LHS depending on the [*check_static_value*](#) parameter. Set the value of the parameter to yes to enable the tag to not report violations for such cases.

Example:

```
abc[199:49] = 200'b0; // no violation as value 0 can be
accommodated in 151 bit LHS.
```

```
def[2:0] = 4'1000; // violation as value 8 can't
be accommodated in 3 bit LHS
```

- [*report_semicolon*](#): Default value is no. By default, multiple violations may be reported on same assignments due to different hierarchies. Set the value of the parameter to yes to report only one violation per assignment.
- [*consider_sub_as_add*](#): By default, when the *nocheckoverflow* parameter is set to no, the *W164a* tag considers the width of expression like $b - 1$ as the width of b when there is no underflow due to -ve operator. However, this behavior is limited to a simple -ve operation, where this is a part of arithmetic expression other than $+/-$. Set the value of the parameter to yes to enable the tag to calculate the width of $a-b$ / $b-1$ as $a+b$ / $b+1$ respectively.
- [*consider_lrm_for_div*](#): Default value is no. Set the value of the parameter to yes to enable the tag to calculate the width of the division to be the maximum width of the operands (LRM width) when *nocheckoverflow* parameter is set to no.

- **`ignore_nonstatic_leftshift`**: Default value is `no`. Set this parameter to `yes` enable the tag to ignore assignments where the RHS is a left shift operator and both the operands are non-static.
- **`report_only_from_one_hierarchy`**: By default, the value of the parameter is `no`, and the tag reports violations for multiple hierarchies. Set the value of the parameter to `yes` to suppress the hierarchy field and report only one violation per assignment.
- **`ignore_counter_with_same_width`**: Default value is `no`. Set the value of the parameter to `yes` to not report violations for cases where any constant value added/subtracted from different signal on both LHS and RHS with the same width is treated as a counter.

Example 1:

`a = b + 1` where width of both `a` and `b` are same.

Example 2:

```
logic [5:0] a;
logic [5:0] a_nxt;
a_nxt = a + 3; // no violation
```

Example 3:

```
logic [6:0] a;
logic [5:0] a_nxt;
a_nxt = a + 3; // violation as width of a & a_nxt
are different
```

Example 4:

If `ignore_counter_with_same_width` is set to `only_by_one`, then the tag will suppress violations for cases where there is increment/decrement only by 1 (constant's natural width should be 1).

```
wire [2:0] a,b;
assign a = b + 1; // no violation
assign a = b + 1'b1; // no violation
assign a = b + 1'd1; // no violation
```

Example 5:


```

logic [24:0] a, b, c;
assign a = {2'b01, c[22:0]};
assign b = ~a + 25'h1; // no violation

```

Constraint(s)

None

Messages and Suggested Fix

Verilog

The following message is displayed for the `<lexpr>` LHS expression of an assignment of width `<widthl>`, which is less than the `<rexpr>` RHS expression of width `<widthr>`:

[WARNING] LHS: '`<lexpr>`' width `<widthl>` is less than RHS: `<rexpr>` width `<widthr>` in assignment [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the signal.

If the [datapath_or_control](#) parameter is set `datapath`, `control`, or `all`, the following message is reported:

[WARNING] LHS `<control | datapath>` signal: '`<LHS-expression>`' width `<LHS-width>` is less than RHS: '`<RHS-expression>`' width `<RHS-width>` in assignment [Hierarchy: '`<hier-path>`']

Potential Issues

A violation is reported when the LHS width is less than the RHS width.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, click [How to Debug and Fix](#).

VHDL

Message 1

The following message is displayed for the `<lexpr>` LHS expression of an

assignment of width `<widthl>`, which is less than the `<rexpr>` RHS expression of width `<widthr>`:

[WARNING] LHS: '`<exprl>`' (width `<widthl>`) is less than RHS: '`<exprr>`' (width `<widthr>`) in assignment [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the signal.

Potential Issues

A violation is reported when the LHS width is less than the RHS width in an assignment.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, click [How to Debug and Fix](#).

Message 2

The following message is displayed for the `<rrange>` range of the `<exprr>` RHS expression, which is not within the `<lrange>` range of LHS expression `<exprl>`:

[WARNING] Range of RHS '`<exprr>`' (`<rrange>`) is not within the range of LHS '`<exprl>`' (`<lrange>`)

Potential Issues

A violation is reported when the range of the RHS expression is less than the range of the LHS expression.

Consequences of Not Fixing

This may result in overflow and loss of data.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the line where the width mismatch is detected.

To resolve the violation, explicitly truncate as necessary since it is more readable, instead of relying on the default behavior. Examine the logic to ensure that the truncation does not affect the behavior of the design.

In case of counters specified as integers with range, a fix is not required since the user puts the check for avoiding overflow. It is considered a normal practice to use integer ranges for counters.

Example Code and/or Schematic

Verilog

Example 1

Consider the following example code in which the *strict* parameter reports a violation:

```
wire a,b;
assign a = b + 4 ; //Violation reported with strict
```

Example 2

Consider the following example in which no violation is reported:

```
A [12:0] = { 3'b000, b[9:0] } ;
```

In this case, the LHS width is 13. This LHS width lies between original RHS width 10, which is calculated without considering leading zeros and RHS width 13, which is calculated after considering leading zeros. Therefore, no violation is reported.

Example 3

Consider another example, in which no violation is reported:

```
A [11:0] = { 3'b000, b[9:0] } ;
```

In this case, the LHS width is 12. This LHS width lies between original RHS width 10, which is calculated without considering leading zeros and RHS width 13, which is calculated after considering leading zeros. Therefore, no violation is reported.

Example 4

Consider another example, in which a violation is reported:

```
A [7:0] = { 3'b000, b[9:0] } ;
```

In this case, the LHS width is 8. This LHS width does not lie between original RHS width 10, which is calculated without considering leading zeros and RHS width 13, which is calculated after considering leading zeros. Therefore, a violation is reported.

Example 5

Consider the following example, in which violations are reported for a concatenation operator:

```
module test3(d,clk,rst,q1, q2, q3, q4, q5);
input d,clk,rst;
output reg [4:0]q1;
output reg [9:0]q2;
output reg [7:0]q3;
output reg [12:0]q4;
output reg [4:0]q5;
wire [4:0]datain;

always @(posedge clk)
begin
    if(rst)
        begin
            q1 = {5'b00000, datain};
            q2 = {5'b00000, datain};
            q3 = {5'b00000, datain};
            q4 = {5'b00000, datain};
            q5 = {5'b00100, datain};

            q2 = {5'b00000, 3'b000, datain};
            q3 = {5'b00000, 3'b000, datain};

            q2 = {5'b00010, 3'b000, datain};
            q2 = {5'b00000, 3'b010, datain};
        end
    end
endmodule
```

The *W164a* Tag reports the following violations for the above example:

LHS: 'q5' width 5 is less than RHS: '{5'b00100, datain}' width 10 in assignment [Hierarchy: ':test3']

LHS: 'q2' width 10 is less than RHS: '{5'b00010, 3'b000, datain}' width 13 in assignment [Hierarchy: ':test3']

Example 6

Consider the following example:

```
assign out2 [8:0] = {3'b001,in1,8'b00001111};
```

In the above example, the Tag preserves all leading zeros in case of out2 assignment, which is 3'b001. Therefore, the width of the assignment is calculated as 12 bits.

Example 7

Consider the following example:

```
a[3:0] <= b[5:2] + 1;
```

The W164a Tag does not report a violation for the above example as it is a case of counter, where addition is with 1.

Example 8

The following example demonstrates width calculation for the *W164a* Tag when the *handle_shift_op* parameter is set to *shift_both*:

```
wire [3:0] w1, w2, w3, w4;
assign w1[2:0] = w2 << w3;           //RHS Width is 19
assign w1 = 3'b110 << w2;           //RHS Width is 18
assign w1[2:0] = w2 << (w3 + w4);    //RHS Width is 19
assign w1[2:0] = w2 << (w3 + w4 + 5'd0); //RHS Width is
34
assign w1[2:0] = w2 << (w3 - w4);    //RHS Width is 19
assign w1[2:0] = w2 << {w3[0],w4[0]}; //RHS Width is 7
assign w1[2:0] = w2 << {w3[0],1'b0}; //RHS Width is 6
assign w1 = 10'h001 << w3[1:0];      //RHS Width is 10
assign w1 = {1'b0, w2[2:0]} << w3[0]; //RHS Width is 4
```

Example 9

The following example demonstrates width calculation for the *W164a* Tag when the *handle_shift_op* parameter is set to *shift_both*:

```
module m1();
wire [40:0]a, b, c;

assign a[3:0] = b[3:0] << 1;         //RHS width = 5
assign a[3:0] = b[3:0] << c[1];     //RHS width = 5
assign a[3:0] = 11 << c[0];         //RHS width = 5
assign a[31:0] = 11 << c[0];        //RHS width = 5
```

```

assign a[3:0] = b[3:0] >> 1;           //RHS width = 3
assign a[2:0] = b[3:0] >> c[1];       //RHS width = 4
assign a[2:0] = 11 >> c[1];          //RHS width = 4
endmodule

```

Example 10

The following command reports the counter type of assignments for the *W164a* Tag when the *check_static_value* and *check_counter_assignment* parameters are set to yes:

```
assign w1[3:0] = w2[3:0] + 1;
```

Example 11

Consider the following example in which the *check_unsign_overflow* parameter is set to yes:

```

wire [5:0] a,b;
wire [6:0] c;
assign c = {a[5],a} + {b[5],b};

```

The *W164a* Tag reports a violation for the unsigned expression.

Example 12

Consider the following example in which the *W164a* Tag reports violations for all three assignments when the *check_static_value* parameter is set to yes:

```

wire [4:0] AAA, BBB, CCC;
assign CCC = 6'b100111;
assign CCC = 120;
assign CCC = (AAA + 1) - BBB;

```

VHDL

Example 1

Consider the following example code in which the *strict* parameter reports a violation:

```

library ieee;
use ieee.std_logic_1164.all

a (3 downto 0) <= b ( 3 downto 0) + 4;

```

```
a (3 downto 0) <= 4 + b (3 downto 0);
```

Example 2

Consider the following example:

signal SIG1, SIG3	:Unsigned (3 downto 0)
signal SIG2	:Unsigned (2 downto 0)
signal SIG4	:Unsigned (4 downto 0)
SIG3 < =SIG1+SIG2 //no violation (left width 4 right shift 4)	

In the above example, since both the operands are variables, then the max width is taken. Also, the W164a Tag does not report a violation in this case as the LHS width is equal to the RHS width, which is 4.

Example 3

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity test is
  port (
    sig1 :      unsigned( 3 downto 0);
    sig4 :      unsigned( 4 downto 0);
    sig3 :      out unsigned( 3 downto 0)
  );
end test;

architecture test_arc of test is
begin
  sig3 <= sig1 + 17; --No violation if parameter
nocheckoverflow is set then width(left4,right4)
  sig3 <= sig4 + 17; --Violation if parameter
nocheckoverflow is set then width(left4,right5)
```

```
end test_arc;
```

Example 4

Consider the following example:

signal SIG1	:Unsigned (3 downto 0)
signal SIG2	:Unsigned (2 downto 0)
signal SIG3	:Unsigned (1 downto 0)

**SIG1 <=SIG2*SIG3 //violation (left width 4
right width 5)**

In the above example, the Tag reports violation for the expression as there is a mismatch between LHS and RHS since LHS is 4 and RHS is 5.

Example 5

Consider the following example:

signal SIG1	:Unsigned (3 downto 0)
signal SIG2	:Unsigned (2 downto 0)
signal SIG3	:Unsigned (1 downto 0)

**SIG1 <=SIG2*SIG3 //violation (left width 4
right width 5)**

In the above example, since both the operands are variables, the RHS width is the sum of both the variables. Therefore, the LHS width is 4 and RHS width is 5. As a result, the Tag reports a violation in this case.

Example 6

Consider the following example:

SIG1 <= SIG3 * 17 //no violation (width is 4 on both sides)	
SIG1 <= SIG2 * 17 //violation (left 4, right 6)	

In the above example, one operand is a variable and the other variable is

static. Therefore, the RHS width is 2*(Variable width), that is, LHS width is 4 and the RHS width is 6. As a result, the Tag reports a violation in this case.

Example 7

Consider the following example:

signal SIG1	:Unsigned (3 downto 0)
signal SIG2	:Unsigned (3 downto 0)
signal SIG3	:Unsigned (1 downto 0)
signal SIG4	:Unsigned (4 downto 0)
SIG1 <=SIG2/SIG3 //No violation (left width 4 right width 4)	

In the above example, both the operands are variables. Therefore, the RHS width is the left operand width, that is, the width of both LHS and RHS is 4. As a result, the Tag does not report a violation in this case.

Example 8

Consider the following example:

SIG1 <= SIG2 / 17 //no violation (width is 4 on both sides)	
SIG1 <= SIG4 / 17 //violation (left 4, right 5)	
SIG1 <=17 / SIG2 //no violation (left 4, right 4)	

In the above example, one operand is a variable and the other variable is static. Therefore, the RHS width is variable width. The Tag reports a violation for the second expression as LHS width is 4 and the RHS width is 5. Also, the Tag does not report a violation for first and third expression as the LHS and RHS width for both the expressions is 4.

Example 9

Consider the following example:

a(3 downto 0) <= b(5 downto 2) + 1;

In the above example, the W164a Tag does not report a violation as it is a case of counter, where addition is with 1.

Example 10

The following command reports the counter type of assignments for the *W164a* Tag when the *check_counter_assignment* and *strict* parameters are set to yes:

```
w1(3 downto 0) <= w2(3 downto 0) + 1;
```

Example 11

Consider the following example:

```
architecture rtl of rgx_pbe_gammacmp_lut is
    type aa_type is array(3 downto 0) of std_logic_vector(11
        downto 0);
    constant const_gamma_table : aa_type := (
        X"1ff",
        X"ff",
        X"2ff",
        X"00f"
    );

    signal t : std_logic_vector( 9 downto 0);
    signal a : std_logic_vector( 13 downto 0);
begin
    t <= const_gamma_table(to_integer(unsigned(a)));
end rtl;
```

In the above example, by default, no violation is reported because the width of RHS and LHS is the same (10), as for RHS, maximum width among the initialized elements of the array (that is the width of *x"2ff"*) is considered. When the *nocheckoverflow* parameter is set to *yes*, for RHS, the declared width of an element of array is considered (12), hence a violation is reported.

Default Severity Label

Warning

Tag Group

Lint_Elab_Tags

Lint Tags

Reports and Related Files

None

W164a_a

LHS width is less than RHS width of assignment (Hard Mismatch)

When to Use

Use this tag to identify assignments in which the LHS width is less than the RHS width (hard mismatch).

Description

A hard mismatch happens when the width of the RHS of the assignment is greater than the width of the LHS of the assignment. In the tool, the assignment statements that are violated by the *W164a* tag irrespective of the value of the *nocheckoverflow* parameter are categorized as hard mismatches. In other words, in the assignment, if both the LRM width and the natural width of the RHS of the assignment is greater than the width of the LHS, it is considered as a hard mismatch. But for multiplication operators, the violation is categorized as a hard mismatch if the natural width is greater than the LHS width and not the LRM width. For more details, see the following examples.

Example 1:

```
module test();  
wire [3:0] a;  
wire [7:0]b;  
  
assign a = b; //both natural and LRM width is 8.  
  
endmodule
```

In the above example, the violation for `a = b` is treated as a hard mismatch as assigning an 8-bit variable to a 4-bit variable results in a data loss of the most significant bits of variable `b`.

Example 2:

```
module test();  
wire [3:0] a;  
wire [4:0]b,c;  
  
assign a = b + c; //both natural and LRM width is 5  
  
endmodule
```

In the above example, the violation for $a = b + c$ is treated as a hard mismatch as assigning a 5-bit addition to a 4-bit variable results in a data loss of the most significant bits of the addition of $a + b$. Violation for tag *W164a* is reported even if *nocheckoverflow* is set to yes.

Example 3:

```
module test();
wire [3:0] a, b, c;
assign a = b + c; //natural width is 5 and LRM width is 4.
endmodule
```

In the above example, the addition of two 4-bit variables needs 5 bits to accommodate the carry bit of the addition. When *nocheckoverflow* is set to yes, carry bit is discarded and no violation is reported when it is assigned to a 4-bit variable. This violation is not treated as a hard mismatch. Data loss is for missing the carry bit of the addition and this violation is covered in the *W164a_b* tag.

Example 4:

```
module test();
wire [3:0] a, b, c;
assign a = b * c; //natural width is 8 and LRM width s 4
and violation when check_natural_width_of_multiplication
is set to yes
endmodule
```

In the above example, the multiplication of two 4-bit variables needs 8 bits to accommodate the maximum value of $a * b$. When *nocheckoverflow* is set to yes, according to LRM, the width would be the maximum width of its operands and in this example it is 4. By default this violation is categorized in tag *W164a_b*, but if you set *check_natural_width_of_multiplication*, then we consider this multiplication violation as a hard mismatch because unlike in the addition operator where carry bit is discarded, this potentially discards multiple bits that results in a major data loss.

Example 5:

```
module test();
```

```
wire [3:0] a, b;
wire [4:0] c ;
assign a = c / b; //natural width is 5 and LRM width is 4.
endmodule
```

Example 6:

```
module test();
wire [3:0] a, b;
assign a = b << 2; //natural width 6, LRM width is 4.
```

For a shift operator, the parameter value [handle_shift_op](#) is honored and the violation is considered as hard mismatch if the width calculated according to parameter value [handle_shift_op](#) is greater than the width of LHS of the assignment. For this tag, a violation is reported if the [handle_shift_op](#) parameter value is `shift_left` or `shift_both`.

The *W164a_a* tag reports violations for assignments in which the width of the LHS is less than the (implied) width of the RHS expression. This mismatch may result in truncation of some bits in the assignment.

The assignment type of a binary expression is determined in the following order:

- If LHS or RHS is reg then expression type is reg
- If LHS or RHS is wire then expression type is wire
- If LHS or RHS is real then expression type is real
- If LHS or RHS is real-time then expression type is real-time
- If LHS or RHS is time then expression type is time
- If LHS or RHS is integer then expression type is integer
- If LHS or RHS is static then expression type is static
- If LHS or RHS is constant integer then expression type is constant integer
- If LHS or RHS is boolean then expression type is boolean

Width Calculation

By default, the width is calculated considering the best fit width of an

expression. That is the width in which maximum value of an expression can be accommodated. When you set the value of the parameter to `yes` then the width is calculated according to the LRM `nocheckoverflow` and natural width is considered for constants.

The behavior of the tag is explained by the following cases.

- For Constant Expressions (with the value of the `nocheckoverflow` parameter set to `no`):

- For Constant integer expressions, the width is calculated based on the value of the expression:

```
out[3:0] = 2 + 15 ;    //Violation, RHS Width = 5 (Value
                       = 17))
out[6:0] = 100 << 4 ; //Violation, RHS Width = 11
out[4:0] = 100 >> 2 ; //RHS Width = 5
```

- If constant is not a part of the sub-expression then the specified width is considered:

```
out[11:0] = 12'b0 ; //RHS Width = 12
```

- If the width of a based number is not specified then the natural width is considered:

```
out[8:0] = 'h344;      //RHS Width = 10
```

For above cases, behavior of the tag remains the same when `nocheckoverflow` parameter is set to `yes`.

- For Arithmetic Operators

- The width addition operator (with the value of the `nocheckoverflow` parameter set to `no`), is calculated as a width where maximum value of an expression is accommodated. Consider the following example:

```
out[2:0] = in1[1:0] + (3/3) ;
Max value RHS 3 + 3 + 3 = 9
RHS Width = 4
```

```
out[2:0] = in1[1:0] + (3/3) ; //RHS Width 3
Max Value RHS = 3 + 1 = 4
RHS Width = 3
```

```
out[5:0] = in1[5:0] + 7'b1110010; //RHS Width 8
out[4:0] = 15 + 3'b111;           //RHS width 5
```

- ❑ For the subtraction operator (with the value of the *nocheckoverflow* parameter set to *no*), the width is calculated in a similar way as it is done for addition:

```
out[3:0] = in1[3:0] - in1[1:0]; //RHS Width 5
out[4:0] = 15 + 3'b111 //RHS Width 5
```

- ❑ For the multiplication operator (with the value of the *nocheckoverflow* parameter set to *no* and the *check_natural_width_of_multiplication* is *yes*), the width is calculated as follows:

When both operands are variable, then the RHS width is the sum of the width of both operands:

```
out[6:0] = in1[1:0] * in2[1:0] * in3[1:0]; //RHS Width 6
out[2:0] = in1[2:0] * in1[1:0]; //RHS Width 5
```

When one operand is static and other is variable, then the RHS width is calculated considering maximum value of the expression:

```
out[3:0] = in1[3:0] * 4'b1; //RHS Width 4, Max value = 15*1 = 15
out[3:0] = in1[3:0] * 4'b10; //RHS Width 5, Max
value = 15*2 = 30
out[7:0] = in1[3:0] * 4'b1010; //RHS Width 8, Max value = 15*10 =
150
out[3:0] = in1[3:0] * 4; //RHS Width 6, Max
value = 15*4 = 60
```

- ❑ For the division operator (with the value of the *nocheckoverflow* parameter set to *no*), the width of the RHS is assumed as the width of left operand:

```
out[2:0] = in1[2:0]/in1[1:0]; //RHS Width 3
out[1:0] = in1[1:0]/in1[2:0]; //RHS Width 2
out[3:0] = in1[3:0]/4'b10; //RHS Width 4
out[2:0] = 4/in1[3:0]; //RHS Width 3
```

- ❑ When the *nocheckoverflow* parameter is set to *yes*, the width is assumed to be the same as the width of term having maximum width. Consider the following example:

```
out[1:0] = in1[1:0] + in2[1:0] + in3[1:0]; //RHS Width = 2
out[5:0] = in1[5:0] + 5'b1010; //RHS Width 6
out[2:0] = 15 + 3'b111; //RHS width 4
out[3:0] = in1[3:0] - in1[1:0]; //RHS Width 4
```



```

out[2:0] = in1[3:0] - 4'b1010; //RHS Width 4
out[2:0] = in1[2:0] * in1[1:0]; //RHS Width 3
out[3:0] = in1[3:0] * 4'b10; //RHS Width 4
out[1:0] = in1[1:0]/in1[2:0]; //RHS Width 3
out[2:0] = 4/in1[3:0]; //RHS Width 4

```

■ For Shift Operators:

- For the right shift (with the value of the *nocheckoverflow* parameter set to no), if left operand width is matching the LHS width then no violation is reported. Consider the following example:

```

out[3:0] = in1[3:0] >> in ; //RHS Width 4
out[2:0] = in1[4:0] >> 2 ; //RHS Width 5
out[31:0] = 4 >> in ; //RHS Width 32
out[4:0] = in1[4:0] >> in2[1:0] ; //RHS Width 5
out[3:0] = in1[4:0] >> in2[1:0] ; //RHS Width 5

```

- For the left shift (with the value of the *nocheckoverflow* parameter set to no), if shifted or left operand width is matching the LHS width, no violation is reported. Consider the following example:

```

out[2:0] = 4'b0001 << in ; //RHS Width 4
out[2:0] = 4 << in ; //RHS Width 32
out[4:0] = in1[3:0] << 1 ; //RHS Width 5
out[6:0] = in1[4:0] << 2 ; //RHS Width 7
out[3:0] = in1[4:0] << in2[2:0] ; //RHS Width 5

```

- When the *nocheckoverflow* parameter is set to yes, the width of the left operand is considered for both left shift and right shift operations.

■ For Self-Determined Expressions

- The width of a self-determined expression (with the value of the *nocheckoverflow* parameter set to no), is calculated according to the LRM. Consider the following example:

```

wire a,b,c;
assign a = {b+c}; //LHS: 1, RHS: 1
assign b = {1'b1,b+c}; //LHS: 1, RHS: 2

```

- The behavior of the tag remains the same when *nocheckoverflow* parameter is set to yes.

■ For Conditional Operators

The width of a conditional operator (with the value of the *nocheckoverflow* parameter set to *no*), is calculated as follows:

- ❑ A violation is reported if there is width mismatch either in the left expression or in the right expression. Consider the following example:

```
out[0] = in1[2] ? in2[0] : in3[2:0] ;    //RHS Width 3
out[1:0] = in1[2] ? in2[2:0] : in3[1:0]; //RHS Width 3
out[0] = in1[2] ? in2[0] : in3[2] ;    //RHS Width 1
```

- ❑ The behavior of the tag remains same when *nocheckoverflow* parameter is set to *yes*.

■ For Power Operators

The width of a power operator (with the value of the *nocheckoverflow* parameter set to *no*), is calculated as follows:

- ❑ If the RHS expression of the power operator is static, then the width of the expression is calculated as per the following formula:

Expression width = LHS expression width * RHS expression value

Consider the following example:

```
out[1:0] = in1[1:0] ** 12;    //RHS Width 24
```

- ❑ If the RHS expression of a power operator is non-static then the LHS expression width is reported:

```
out[1:0] = in1[1:0] ** in2[3:0]; //RHS Width 2
```

- ❑ When the *nocheckoverflow* parameter is set to *yes* then the LHS expression width is reported:

```
out[1:0] = in1[1:0] ** 12;    //RHS Width 2
out[1:0] = in1[1:0] ** in2[3:0]; //RHS width 2
```

Parameter(s)

- *check_assign_pattern*: Default value is *no*. Set the value of the parameter to *yes* to check assign pattern statements where the elements are assigned names or indexes.

- **[check_static_value](#)**: Default value is no. Set the value of the parameter to yes or <rule_list> to report violation for all cases with width mismatch, involving static expressions and non-static expressions having a static part. You can also set the value of the parameter to only_static to ignore violations for expressions, which do not have a static part. Other possible values are only_const and only_expr.
- **[disable_rtl_deadcode](#)**: The default value is no. Set the value of the parameter to yes to disable violations for disabled code in loops and conditional (if condition, ternary operator) statements.
- **[enable_rtl_deadcode](#)**: The default value is no. Set this parameter to yes to report different violations for dead codes in the RTL resulting from if/for/while and ternary operators that are ignored when the [disable_rtl_deadcode](#) parameter is enabled.
- **[sign_extend_func_names](#)**: The default value is "EXTEND". Set the value of the parameter comma-separated list of function names to enable the W164a tag to recognize VHDL sign extension functions and calculate width of extend functions as per the const extension argument specified in the argument list.
- **[checkOperatorOverload](#)**: Default value is yes. Set this parameter to no to evaluate width of the expression without considering overloaded operators. This parameter is applicable for VHDL only.
- **[check_counter_assignment](#)**: Default value is no. Set this parameter to yes to report a violation for the counter type of assignments. This parameter is applicable for both Verilog and VHDL in the W164a tag.
For details about the cases that are considered as counter cases in Verilog, see the [check_counter_assignment](#) parameter section.
- **[check_unsign_overflow](#)**: Default value is no. This indicates the tag suppresses the overflow when sign extension is used for unsigned signals in addition or subtraction operation. If you set this parameter to yes or W164a, the tag does not suppress the overflow and reports a violation when sign extension is used for unsigned signals in addition or subtraction operation.
- **[handle_shift_op](#)**: Default value is no. In this case, no violation is reported if the shifted or non-shifted width of a shift expression matches the LHS width of an assignment. But the tag does not calculate the shifted width, if the RHS of the shift expression is non-static. Set this parameter to shift_left, shift_right, shift_both, no_shift, no_shift_forced,

or comma-separated list of tag names, to compare shifted or non shifted widths for left and right shift expressions.

- [*handle_lrm_param_in_shift*](#): Default value is `no`. Set this parameter to `yes` to enable the *W164a_a* Tag to honor the [*use_lrm_width*](#) parameter in the width calculations of shift expressions.
- [*check_concat_max_width*](#): Default value is `no`. In this case, no violation is reported when the width of the LHS expression is present between the width of the RHS expression without considering zero concatenated bits and the width of the RHS after adding zero concatenated bits. If you set this parameter to `yes`, the RHS width is considered as the width after adding zero concatenated bits. That is, the violation is reported if the LHS width does not match the RHS width after adding zero concatenated bits. This parameter is applicable for **Verilog** only.
- [*datapath_or_control*](#): Default value is `no`. In this case, and the tag reports violations for all type of signals. Set this parameter to `datapath`, `control`, or `all` to specify the type of signals to be checked.
- [*control_sig_detection_nf*](#): Default value is `no`. If you set this parameter to `yes`, the tag treats a signal to be a control signal only if it is used in a conditional operator or control binary operator within the scope of currently processing statement. The tag does not consider the signal to be a control signal even if it is used as a control signal in other places of the module (that is, the statements other than the current statement).
- [*handle_zero_padding*](#): Default value is `no`. Set the value of the parameter to `yes` to perform leading zero expansion and truncation of RHS of an assignment.
- [*force_handle_shift_op*](#): Default value is `no`. Set this parameter to `yes` to enable the Tag to honor [*handle_shift_op*](#) parameter value even if the [*nocheckoverflow*](#) parameter is set to `yes`.
- [*concat_width_nf*](#): Default value is `no`. Set the value of the parameter to `yes` to specify if the *W164a* tag should use new algorithm to calculate the width of concatenations ignoring the self-determined nature of concatenation items.
- [*nocheckoverflow*](#): Default value is `no`. Set this parameter to `yes` or tag name to check the bit-width as per LRM. Other possible value is the tag name.
- [*treat_concat_assign_separately*](#): Default value is `no`. Set the value of the parameter to `yes` to use this parameter to report violation for each

bucket assignment in unpacked array separately. For a packed array, a violation is reported for the whole array.

- **`process_complete_condop`**: Default value is `no`. Set the value of the parameter to `yes` to enable the tag checking on both operands of the `condop` assignment.
- **`check_lrm_and_natural_width`**: Default value is `no`. Set the value of the parameter to `yes` to calculate both `lrm` and `natural` width before reporting violation for the assignment statement. If any of these width matches with `RHS`, then no violation is reported for the assignment.
- **`strict`**: Default value is `no` and the tag reports violation if the `RHS` expression contains `wire` or `reg`. Set this parameter to `yes` to report all assignments. Also, the tag reports integer port signals in the `RHS` expression, in this case.
- **`use_lrm_width`**: Default value is `no`. Set this parameter to `yes` to consider the `LRM` width of integer constants, which is 32 bits.
- **`check_natural_width_of_multiplication`**: Default value is `no`. Set the parameter to `yes`, to calculate the width of the multiplication expressions as per the `natural` width even if the `nocheckoverflow` parameter is set to `yes`. Set the parameter to a comma/ space-separated list of tags to enable (the parameter) for the tags specified in the list.
- **`report_nonblocking_in_error`**: Default value is `no`. Set the value of the parameter to `yes` to report a violation (with `Error` severity) for nonblocking assignments.
- **`ignore_parameter_assignment`**: Default value is `no`. Set this parameter to `yes` or comma/space-separated list of tag names to enable the tag to not report any violation for parameter and local parameter assignments.
- **`ignore_counter_with_same_width`**: Default value is `no`. Set the value of the parameter to `yes` to not report violations for cases where any constant value added/subtracted from different signal on both `LHS` and `RHS` with the same width is treated as a counter.

Example 1:

`a = b + 1` where width of both `a` and `b` are same.

Example 2:

```
logic [5:0] a;
```

```
logic [5:0]a_nxt;
a_nxt = a + 3; // no violation
```

Example 3:

```
logic [6:0]a;
logic [5:0]a_nxt;
a_nxt = a + 3; // violation as width of a & a_nxt
are different
```

Example 4:

If `ignore_counter_with_same_width` is set to `only_by_one`, then the tag will suppress violations for cases where there is increment/decrement only by 1 (constant's natural width should be 1).

```
wire [2:0] a,b;
assign a = b + 1; // no violation
assign a = b + 1'b1; // no violation
assign a = b + 1'd1; // no violation
```

Example 5:

```
logic [24:0] a, b, c;
assign a = {2'b01, c[22:0]};
assign b = ~a + 25'h1; // no violation
```

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for the `<lexpr>` LHS expression of an assignment of width `<widthl>`, which is less than the `<rexpr>` RHS expression of width `<widthr>`:

[WARNING] LHS: '`<lexpr>`' width `<widthl>` is less than RHS: `<rexpr>` width `<widthr>` in assignment [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the signal.

If the [datapath_or_control](#) parameter is set `datapath`, `control`, or `all`, the following message is reported:

[WARNING] LHS <control | datapath> signal: '<LHS-expression>' width <LHS-width> is less than RHS: '<RHS-expression>' width <RHS-width> in assignment [Hierarchy: '<hier-path>']

Potential Issues

A violation is reported when the LHS width is definitely less than the RHS width.

Consequences of Not Fixing

This may result in overflow and loss of data.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the line where the width mismatch is detected.

To resolve the violation, explicitly truncate as necessary since it is more readable, instead of relying on the default behavior. Examine the logic to ensure that the truncation does not affect the behavior of the design.

In case of counters specified as integers with range, a fix is not required since the user puts the check for avoiding overflow. It is considered a normal practice to use integer ranges for counters.

Example Code and/or Schematic

Consider the following example:

```
wire [4:0] AAA, BBB, CCC;
assign CCC = 6'b100111;
assign CCC = 120;
assign CCC = (AAA + 100) - BBB;
```

For the above example, by default, no violation is reported. However, when you set the value of the `check_static_value` parameter to `yes`, the `W164a_a` tag reports a violation.

Default Severity Label

Warning

Reports and Related Files

None

W164a_b

LHS width is less than RHS width of assignment (Soft Mismatch)

When to Use

Use this tag to identify assignments in which the LHS width is less than the RHS width (soft mismatch).

Description

A soft mismatch happens when the width of the RHS of the assignment is greater than the width of the LHS of the assignment for some of the values in the RHS of the assignment. In the tool, the assignment statements that are violated by the *W164a* tag when the *nocheckoverflow* parameter is set no, but are not violated when the *nocheckoverflow* is set to yes, are categorized as soft mismatches. In other words, a violation is categorized as a soft mismatch, if the LHS of the assignment matches with LRM width but not with the natural width of the RHS of the assignment. For more details, see the following examples.

Example 1:

```
module test();
wire [3:0] a, b, c;
assign a = b + c; //natural width is 5 and LRM width is 4.
endmodule
```

In the above example, the addition of two 4-bit variables needs 5 bits to accommodate the carry bit of the addition. When *nocheckoverflow* is set to yes, carry bit is discarded and no violation is reported when it is assigned to a 4-bit variable. This violation is not treated as a soft mismatch. Data loss is for missing the carry bit of the addition.

NOTE: Any other violations that are not reported by the *W164a_a* tag but reported by the *W164a* tag are considered as soft mismatches.

The *W164a_b* tag reports violations for assignments in which the width of LHS is less than the (implied) width of the RHS expression. This mismatch

may result in truncation of some bits in the assignment.

The assignment type of a binary expression is determined in following order:

- If LHS or RHS is reg then expression type is reg
- If LHS or RHS is wire then expression type is wire
- If LHS or RHS is real then expression type is real
- If LHS or RHS is real-time then expression type is real-time
- If LHS or RHS is time then expression type is time
- If LHS or RHS is integer then expression type is integer
- If LHS or RHS is static then expression type is static
- If LHS or RHS is constant integer then expression type is constant integer
- If LHS or RHS is boolean then expression type is boolean

Width Calculation

By default, the width is calculated considering the best fit width of an expression. That is the width in which maximum value of an expression can be accommodated. When you set the value of the [nocheckoverflow](#) parameter to `yes` then the width is calculated according to the LRM and natural width is considered for constants.

The behavior of the tag is explained by the following cases.

- For Arithmetic Operators
 - The width addition operator (with the value of the [nocheckoverflow](#) parameter set to `no`), is calculated as a width where maximum value of an expression is accommodated. Consider the following example:

```
out[1:0] = in1[1:0] + in2[1:0] + in3[1:0]; //RHS
Width 4
```

```
Max value RHS 3 + 3 + 3 = 9
RHS Width = 4
```

```
out[2:0] = in1[1:0] + (3/3); //RHS Width 3
Max Value RHS = 3 + 1 = 4
RHS Width = 3
```

```
out[5:0] = in1[5:0] + 5'b1010; //RHS Width 7
out[4:0] = 15 + 3'b111;      //RHS width 5
```

- ❑ For the subtraction operator (with the value of the *nocheckoverflow* parameter set to *no*), the width is calculated in a similar way as it is done for addition:

```
out[3:0] = in1[3:0] - in1[1:0]; //RHS Width 5
out[4:0] = in1[3:0] - 4'b1010; //RHS Width 5
```

- ❑ For the division operator (with the value of the *nocheckoverflow* parameter set to *no*), the width of the RHS is assumed as the width of left operand:

```
out[2:0] = in1[2:0]/in1[1:0]; //RHS Width 3
out[1:0] = in1[1:0]/in1[2:0]; //RHS Width 2
out[3:0] = in1[3:0]/4'b10;   //RHS Width 4
out[2:0] = 4/in1[3:0];       //RHS Width 3
```

- ❑ When the *nocheckoverflow* parameter is set to *yes*, the width is assumed to be the same as the width of term having maximum width. Consider the following example:

```
out[1:0] = in1[1:0] + in2[1:0] + in3[1:0]; //RHS Width = 2
out[5:0] = in1[5:0] + 5'b1010; //RHS Width 6

out[2:0] = 15 + 3'b111;           //RHS width 4
out[3:0] = in1[3:0] - in1[1:0]; //RHS Width 4

out[2:0] = in1[3:0] - 4'b1010; //RHS Width 4

out[2:0] = in1[2:0] * in1[1:0]; //RHS Width 3
out[3:0] = in1[3:0] * 4'b10;   //RHS Width 4
out[1:0] = in1[1:0]/in1[2:0];  //RHS Width 3
out[2:0] = 4/in1[3:0];         //RHS Width 4
```

■ For Self-Determined Expressions

- ❑ The width of a self-determined expression (with the value of the *nocheckoverflow* parameter set to *no*), is calculated according to the LRM. Consider the following example:

```
wire a,b,c;
assign a = {b+c}; //LHS: 1, RHS: 1
assign b = {1'b1,b+c}; //LHS: 1, RHS: 2
```

- ❑ The behavior of the tag remains the same when *nocheckoverflow* parameter is set to *yes*.
- For Conditional Operators

The width of a conditional operator (with the value of the *nocheckoverflow* parameter set to *no*), is calculated as follows:

 - ❑ A violation is reported if there is width mismatch either in the left expression or in the right expression. Consider the following example:


```
out[0] = in1[2] ? in2[0] : in3[2:0] ;    //RHS Width 3
out[1:0] = in1[2] ? in2[2:0] : in3[1:0]; //RHS Width 3
out[0] = in1[2] ? in2[0] : in3[2] ;    //RHS Width 1
```
 - ❑ The behavior of the tag remains same when *nocheckoverflow* parameter is set to *yes*.

Parameter(s)

- *check_assign_pattern*: Default value is *no*. Set the value of the parameter to *yes* to check assign pattern statements where the elements are assigned names or indexes.
- *strict*: Default value is *no* and the tag reports violation if the RHS expression contains *wire* or *reg*. Set this parameter to *yes* to report all assignments. Also, the tag reports integer port signals in the RHS expression, in this case.
- *check_static_value*: Default value is *no*. Set the value of the parameter to *yes* or *<rule_list>* to report violation for all cases with width mismatch, involving static expressions and non-static expressions having a static part. You can also set the value of the parameter to *only_static* to ignore violations for expressions, which do not have a static part. Other possible values are *only_const* and *only_expr*.
- *check_counter_assignment*: Default value is *no*. Set this parameter to *yes* to report a violation for the counter type of assignments.
- *use_lrm_width*: Default value is *no*. Set this parameter to *yes* to consider the LRM width of integer constants, which is 32 bits.
- *handle_shift_op*: Default value is *no*. In this case, no violation is reported if the shifted or non-shifted width of a shift expression matches the LHS width of an assignment. But the tag does not calculate the shifted width,

if the RHS of the shift expression is non-static. Set this parameter to `shift_left`, `shift_right`, `shift_both`, `no_shift`, `no_shift_forced`, or comma-separated list of tag names, to compare shifted or non shifted widths for left and right shift expressions.

- **`handle_lrm_param_in_shift`**: Default value is `no`. Set this parameter to `yes` to enable the `W164a` tag to honor the `use_lrm_width` parameter in the width calculations of shift expressions.
- **`nocheckoverflow`**: Default value is `no`. Set this parameter to `yes` or tag name to check the bit-width as per LRM. Other possible value is the tag name.
- **`enable_rtl_deadcode`**: The default value is `no`. Set this parameter to `yes` to report different violations for dead codes in the RTL resulting from `if/for/while` and ternary operators that are ignored when the `disable_rtl_deadcode` parameter is enabled.
- **`check_concat_max_width`**: Default value is `no`. In this case, no violation is reported when the width of the LHS expression is present between the width of the RHS expression without considering zero concatenated bits and the width of the RHS after adding zero concatenated bits. If you set this parameter to `yes`, the RHS width is considered as the width after adding zero concatenated bits. That is, the violation is reported if the LHS width does not match the RHS width after adding zero concatenated bits. This parameter is applicable for **Verilog** only.
- **`datapath_or_control`**: Default value is `no`. In this case, and the tag reports violations for all type of signals. Set this parameter to `datapath`, `control`, or `all` to specify the type of signals to be checked.
- **`control_sig_detection_nf`**: Default value is `no`. If you set this parameter to `yes`, the tag treats a signal to be a control signal only if it is used in a conditional operator or control binary operator within the scope of currently processing statement. The tag does not consider the signal to be a control signal even if it is used as a control signal in other places of the module (that is, the statements other than the current statement).
- **`handle_zero_padding`**: Default value is `no`. Set the value of the parameter to `yes` to perform leading zero expansion and truncation of RHS of an assignment.
- **`force_handle_shift_op`**: Default value is `no`. Set this parameter to `yes` to enable the tag to honor `handle_shift_op` parameter value even if the `nocheckoverflow` parameter is set to `yes`.

- [*concat_width_nf*](#): Default value is no. Set the value of the parameter to yes to specify if the W164a tag should use new algorithm to calculate the width of concatenations ignoring the self-determined nature of concatenation items.
- [*treat_concat_assign_separately*](#): Default value is no. Set the value of the parameter to yes to use this parameter to report violation for each bucket assignment in unpacked array separately. For a packed array, a violation is reported for the whole array.
- [*process_complete_condop*](#): Default value is no. Set the value of the parameter to yes to enable the tag checking on both operands of the condop assignment.
- [*check_concat_max_width*](#): Default value is no. Set the value of the parameter to yes to calculate both lrm and natural width before reporting violation for the assignment statement. If any of these width matches with RHS, then no violation is reported for the assignment.
- [*check_natural_width_of_multiplication*](#): Default value is no. Set the parameter to yes, to calculate the width of the multiplication expressions as per the natural width even if the `nocheckoverflow` parameter is set to yes. Set the parameter to a comma/ space-separated list of tags to enable (the parameter) for the tags specified in the list.
- [*report_nonblocking_in_error*](#): Default value is no. Set the value of the parameter to yes to report a violation (with Error severity) for nonblocking assignments.
- [*ignore_parameter_assignment*](#): Default value is no. Set this parameter to yes or comma/space-separated list of tag names to enable the tag to not report any violation for parameter and local parameter assignments.
- [*ignore_counter_with_same_width*](#): Default value is no. Set the value of the parameter to yes to not report violations for cases where any constant value added/subtracted from different signal on both LHS and RHS with the same width is treated as a counter.

Example 1:

a = b + 1 where width of both a and b are same.

Example 2:

```
logic [5:0] a;
```

```
logic [5:0]a_nxt;
a_nxt = a + 3; // no violation
```

Example 3:

```
logic [6:0]a;
logic [5:0]a_nxt;
a_nxt = a + 3; // violation as width of a & a_nxt
are different
```

Example 4:

If `ignore_counter_with_same_width` is set to `only_by_one`, then the tag will suppress violations for cases where there is increment/decrement only by 1 (constant's natural width should be 1).

```
wire [2:0] a,b;
assign a = b + 1; // no violation
assign a = b + 1'b1; // no violation
assign a = b + 1'd1; // no violation
```

Example 5:

```
logic [24:0] a, b, c;
assign a = {2'b01, c[22:0]};
assign b = ~a + 25'h1; // no violation
```

Messages and Suggested Fix

The following message is displayed for the `<lexpr>` LHS expression of an assignment of width `<widthl>`, which is less than the `<rexpr>` RHS expression of width `<widthr>`:

[WARNING] LHS: '`<lexpr>`' width `<widthl>` is less than RHS: `<rexpr>` width `<widthr>` in assignment [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the signal.

If the `datapath_or_control` parameter is set `datapath`, `control`, or `all`, the following message is reported:

[WARNING] LHS `<control | datapath>` signal: '`<LHS-expression>`'

width <LHS-width> is less than RHS: '<RHS-expression>' width <RHS-width> in assignment [Hierarchy: '<hier-path>']

Potential Issues

A violation is reported when the LHS width may be less than the RHS width.

Consequences of Not Fixing

This may result in overflow and loss of data.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the line where the width mismatch is detected.

To resolve the violation, explicitly truncate as necessary since it is more readable, instead of relying on the default behavior. Examine the logic to ensure that the truncation does not affect the behavior of the design.

In case of counters specified as integers with range, a fix is not required since the user puts the check for avoiding overflow. It is considered a normal practice to use integer ranges for counters.

Example Code and/or Schematic

Example 1

Consider the following example:

```
wire a,b;  
assign a = b + 1 ; //Violation with strict
```

Example 2

Consider the following example:

```
wire [4:0] AAA, BBB, CCC;  
assign CCC = (AAA + 1) - BBB;
```

In the above example, by default, no violation is reported. However, when the value of the `check_static_value` parameter is set to yes, the tag reports a violation because the RHS expression is a static expression.

Example 3

Consider the following examples for both Verilog and VHDL:

```
assign w1[3:0] = w2[3:0] + 1;    //Verilog
```

Lint Tags

```
w1(3 downto 0) <= w2(3 downto 0) - 1; --VHDL
```

When the value of the `check_counter_assignment` parameter is set to yes, the tag reports a violation.

Default Severity Label

Warning

Reports and Related Files

None

W164b

Identifies assignments in which the LHS width is greater than the RHS width

When to Use

Use this Tag to identify assignments in which the LHS width is greater than the RHS width.

Description

Verilog

The W164b Tag flags assignments in which the LHS width is greater than the (implied) width of the RHS expression.

The Tag determines the assignment type of a binary expression as per the following table:

Condition	Assignment Type of a Binary Expression
LHS < RHS	Expression type of RHS
LHS > RHS	Expression type of LHS
LHS = RHS	The expression type is same as that of the LHS or the RHS type. The assignment type is determined from the following types in the left to right order: reg, wire, real, realtime, time, integer, static, constant integer, Boolean

Width Calculation for Verilog

By default, the width is calculated considering the best fit width of an expression. That is the width in which maximum value of an expression can be accommodated.

When you set the value of the [nocheckoverflow](#) parameter to `yes` then width is calculated according to the LRM and the natural width is considered for constants.

The behavior of the Tag is explained by the cases below.

- For Constant Expressions

- ? For constant integer expressions, the width is calculated based on the value of expression:

```
out[5:0] = 2 + 15 ;    //RHS Width = 5 (Value = 17)
out[10:0] = 100 << 4 ; //RHS Width = 11
out[6:0] = 100 >> 2 ;  //RHS Width = 5
```

- ? If constant is not a part of sub-expression then the specified width is considered:

```
out[11:0] = 12'b0 ;  //RHS Width = 12
```

- ? If the width of a based number is not specified then the natural width is considered:

```
out[11:0] = 'h344;    //RHS Width = 10
```

For above cases, behavior of Tag remains same when the *nocheckoverflow* parameter is set to *yes*.

■ For Arithmetic Operators

- ? For addition operator, the width is calculated as the width in which maximum value of expression is accommodated. Consider the following example:

```
out[3:0] = in1[1:0] + in2[1:0] + in3[1:0]; //RHS Width 4
Max value RHS 3 + 3 + 3 = 9
RHS Width = 4
```

```
out[2:0] = in1[1:0] + (3/3) ; //RHS Width 3
Max Value RHS = 3 + 1 = 4
RHS Width = 3
```

```
out[6:0] = in1[5:0] + 5'b1010; //RHS Width 7
out[4:0] = 15 + 3'b111;       //RHS width 5
```

- ? For the subtraction operator, the width is calculated in a similar way as it is done for addition:

```
out[4:0] = in1[3:0] - in1[1:0]; //RHS Width 5
out[4:0] = in1[3:0] - 4'b1010; //RHS Width 5
```

- ? For the multiplication operator, the width is calculated as follows:

When both operands are variable, then the RHS width is the sum of the width of both operands:

```

out[5:0] = in1[1:0] * in2[1:0] * in3[1:0]; //RHS Width 6
out[4:0] = in1[2:0] * in1[1:0];           //RHS Width 5

```

When one operand is static and other is variable, then the RHS width is calculated considering maximum value of the expression:

```

out[7:0] = in1[3:0] * 4'b1;           //RHS Width 4, Max
value = 15*1 = 15
out[7:0] = in1[3:0] * 4'b10;         //RHS Width 5, Max
value = 15*2 = 30
out[7:0] = in1[3:0] * 4'b1010; //RHS Width 8, Max value = 15*10 =
150
out[6:0] = in1[3:0] * 4;               //RHS Width 6, Max
value = 15*4 = 60

```

- ? For the division operator, the width of the RHS is assumed as the width of left operand:

```

out[2:0] = in1[2:0]/in1[1:0]; //RHS Width 3
out[2:0] = in1[1:0]/in1[2:0]; //RHS Width 2
out[3:0] = in1[3:0]/4'b10;    //RHS Width 4
out[3:0] = 4/in1[3:0];        //RHS Width 3

```

- ? When the [nocheckoverflow](#) parameter is set to yes, the width is assumed to be the same as the width of term having maximum width. Consider the following example:

```

out[3:0] = in1[1:0] + in2[1:0] + in3[1:0]; //RHS
Width 2
out[6:0] = in1[5:0] + 5'b1010; //RHS Width 6
out[4:0] = 15 + 3'b111;         //RHS width 4
out[4:0] = in1[3:0] - in1[1:0]; //RHS Width 4
out[4:0] = in1[3:0] - 4'b1010;  //RHS Width 4
out[4:0] = in1[2:0] * in1[1:0]; //RHS Width 3
out[7:0] = in1[3:0] * 4'b10;    //RHS Width 4
out[2:0] = in1[1:0]/in1[2:0]; //RHS Width 3
out[3:0] = 4/in1[3:0];         //RHS Width 4

```

- For Shift Operator

- ? For the right shift, if left operand width is matching the LHS width then no violation is reported. Consider the following example:

```

out[3:0] = in1[3:0] >> in ;    //RHS Width 4
out[5:0] = in1[4:0] >> 2 ;    //RHS Width 5
out[31:0] = 4 >> in ;        //RHS Width 32
out[4:0] = in1[4:0] >> in2[1:0] ; //RHS Width 5
out[5:0] = in1[4:0] >> in2[1:0] ; //RHS Width 5

```

- ? For left shift, if shifted or left operand width is matching LHS width then no violation is reported. Consider the following example:

```

out[5:0] = 4'b0001 << in ;    //RHS Width 4
out[31:0] = 4 << in ;        //RHS Width 32
out[5:0] = in1[4:0] << 1 ;    //RHS Width 6
out[6:0] = in1[4:0] << 2 ;    //RHS Width 7
out[4:0] = in1[4:0] << 2 ;    //RHS Width 5
out[7:0] = in1[4:0] << in2[2:0] ; //RHS Width 5

```

- ? When parameter *nocheckoverflow* is set to yes then width of left operand is considered for both left shift and right shift operations.

■ For Self-Determined Expression

- ? For self-determined expressions, the width is calculated as per the LRM:

```

wire a,b,c;
assign a = {b+c};    //LHS: 1, RHS: 1
assign out[2:0] = {1'b1,b+c}; //LHS: 3, RHS: 2

```

- ? Behavior of the Tag remains the same when the *nocheckoverflow* parameter is set to yes

■ For Conditional Operator

The width of conditional operator is calculated as follows:

- ? A violation is reported if there is a width mismatch either in left expression or in right expression. Consider the following example:

```

out[2:0] = in1[2] ? in2[0] : in3[2:0] ; //RHS Width 1
out[2:0] = in1[2] ? in2[2:0] : in3[1:0]; //RHS Width 2
out[0] = in1[2] ? in2[0] : in3[2] ; //RHS Width 1

```

- ? Behavior remains same when the [nocheckoverflow](#) parameter is set to yes .

■ For Power Operator

The width of power operator is calculated as follows:

- ? If the RHS expression of the power operator is static, then the width of the expression is calculated as per the following formula:

Expression width = LHS expression width * RHS expression value

Consider the following example:

```
out[23:0] = in1[1:0] ** 12; //RHS Width 24
```

- ? If the RHS expression of power operator is non static then the LHS expression width is reported. Consider the following example:

```
out[1:0] = in1[1:0] ** in2[3:0]; //RHS Width 2
```

- ? When the [nocheckoverflow](#) parameter is set to yes, the LHS expression width is reported. Consider the following example:

```
out[23:0] = in1[1:0] ** 12; //RHS Width 2
out[1:0] = in1[1:0] ** in2[3:0]; //RHS width 2
```

■ For concatenation operator, when the RHS expression is concatenated with zero bits:

- ? No violation is reported when the width of the LHS expression lies between the original width of the RHS expression and the width after adding zero concatenated bits. Here, the original width is the width without considering the zero concatenation.
- ? When the [nocheckoverflow](#) parameter is set to yes, a violation is reported when LHS width is greater than the RHS width after adding zero concatenated bits.

NOTE: The Tag does not report violation if the RHS is numeric constant zero or any expression multiplied by a numeric constant zero or a based number zero. For example, violation will not be reported if the RHS is either of 0, 4 * 0, 4 * 1'b0, w1 * 4'b0, or (w1+w2) * 0.

VHDL

The W164b Tag flags assignments in which the LHS width is greater than the (implied) width of the RHS expression.

For VHDL designs, the W164b Tag does not check function or procedure

bodies as the size of the arguments may depend on the actual passed in the function or procedure call.

In case of all integer types with range specified, the W164b Tag checking is in different manner. Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

signal A1 : integer range -100 to 100 ;
signal A2 : integer range -100 to 100 ;
signal A3 : integer range -201 to 201 ;
signal A4 : integer range -200 to 200 ;
A3 <= A1 + A2 ;
A4 <= A1 + A2;
```

In the first assignment, RHS expression A1+A2 can have values from -200 to 200 but range of the LHS expression A3 is -201 to 201 which is greater than the range of RHS expression. Hence, the W164b Tag flags a message.

However, in the second assignment, range of LHS expression A4 is -200 to 200, which is equal to range of RHS expression A1+A2. Hence, the W164b Tag does not flag this assignment.

NOTE: *If any operand of RHS has non-integer range type, bit-width will be compared as per the package, not the range.*

Assignment Types

The W164b Tag handles different assignment types as follows:

■ `<int-range> <= <int-range> OP <int-range>`

Range will be evaluated for RHS and then compared with the range of LHS.

■ `<non-int-range> <= <non-int-range> OP
<non-int-range>`

The bit-width for RHS and LHS will be calculated as per the package.

■ `<non-int-range> <= <int-range> OP <int-range>`

The range will be evaluated for RHS, which will then be converted in bit-width and the bit-width will then be compared with bit-width of LHS.

- `<int-range> <= <non-int-range> OP <non-int-range>`
- `<non-int-range> <= <int-range> OP <non-int-range>`
- `<int-range> <= <int-range> OP <non-int-range>`

The bit-width of RHS will be calculated from package and then compared with the bit-width of LHS.

An integer type without the range specified will be treated as `<non-int-range>` and bit-width will be calculated instead of range.

Width Calculation for VHDL

Set the [nocheckoverflow](#) parameter to `yes` or `W164b` to calculate the width according to the LRM (numeric_std lib) as per the following cases:

- For the addition and subtraction operators, the width is calculated based on the following Tags:
 - ? If both the operands are variables, then the max width is taken.
 - ? If one operand is a variable and the other is a static, then the width of the variable is taken.
- For the multiplication operator, the width is calculated based on the following Tags:
 - ? If both the operands are variables, then the RHS width is the sum of both the variables.
 - ? If one operand is a variable and the other is a static, then the RHS width is $2 \times (\text{Variable width})$.
- For the division operator, the width is calculated based on the following Tags:
 - ? If both the operands are variables, then the RHS width is the width of the left operand.
 - ? If one operand is a variable and the other is a static, then the RHS width is the width of the variable.
- For concatenation operator, no violation is reported when the RHS expression is concatenated with zero bits and the width of the LHS expression lies between original width of the RHS expression and width after adding zero concatenated bits. Here, original width is the width without considering the zero concatenation.

NOTE: For constant arrays, the declared width is considered when the [nocheckoverflow](#) parameter is set to yes. Whereas by default, the maximum width among the initialized elements is considered. For more, refer to the VHDL [Example 9](#).

NOTE: The W164b Tag supports generate-if and generate-for blocks.

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

- [check_static_value](#): Default value is no. Set the value of the parameter to yes or <Tag_list> to report violation for all cases with width mismatch, involving static expressions and non-static expressions having a static part. Other possible values are [only_const](#) and [only_expr](#).
- [disable_rtl_deadcode](#): The default value is no. Set the value of the parameter to yes to disable violations for disabled code in loops and conditional (if condition, ternary operator) statements.
- [enable_rtl_deadcode](#): The default value is no. Set this parameter to yes to report different violations for dead codes in the RTL resulting from if/for/while and ternary operators that are ignored when the [disable_rtl_deadcode](#) parameter is enabled.
- [sign_extend_func_names](#): The default value is "EXTEND". Set the value of the parameter comma-separated list of function names to enable the W164b Tag to recognize VHDL sign extension functions and calculate width of extend functions as per the const extension argument specified in the argument list.
- [strict](#): Default value is no.
 For Verilog, by default, the Tag reports violation if the RHS expression contains `wire` or `reg`. Set this parameter to yes to report all assignments. Also, the Tag reports integer port signals in the RHS expression, in this case.
 For VHDL, set the value of the parameter to yes to report violation for addition and subtraction expression, if any one operand is a constant and its width is lesser.

- [*use_lrm_width*](#): Default value is `no`. Set this parameter to `yes` to consider the LRM width of integer constants, which is 32 bits.
- [*nocheckoverflow*](#): Default value is `no`. Set this parameter to `yes` or Tag name to check the bit-width as per LRM. Other possible value is the Tag name.
- [*checkOperatorOverload*](#): Default value is `yes`. Set this parameter to `no` to evaluate width of the expression without considering overloaded operators. This parameter is applicable for **VHDL** only.
- [*check_unsign_overflow*](#): Default value is `no`. This indicates the Tag suppresses the overflow when sign extension is used for unsigned signals in addition or subtraction operation. If you set this parameter to `yes` or `W164b`, the Tag does not suppress the overflow and does not report a violation when sign extension is used for unsigned signals in addition or subtraction operation.
- [*check_natural_width_of_div*](#): Default value is `no`. Set the value of the parameter to `yes` to calculate the natural width of the division operator when it's divisor is static, regardless of the value of the *nocheckoverflow* parameter.
- [*check_natural_width_of_modulo*](#): Default value is `no`. Set the value of the parameter to `yes` to calculate the natural width of the modulo operator, regardless of the value of the *nocheckoverflow* parameter.
- [*check_concat_max_width*](#): Default value is `no`. In this case, no violation is reported when the width of the LHS expression is present between the width of the RHS expression without considering zero concatenated bits and the width of the RHS after adding zero concatenated bits. If you set this parameter to `yes`, the RHS width is considered as the width after adding zero concatenated bits. That is, the violation is reported if the LHS width does not match the RHS width after adding zero concatenated bits. This parameter is applicable for **Verilog** only.
- [*handle_shift_op*](#): Default value is `no`. In this case, no violation is reported if the shifted or non-shifted width of a shift expression matches the LHS width of an assignment. But the Tag does not calculate the shifted width, if the RHS of the shift expression is non-static. Set this parameter to `shift_left`, `shift_right`, `shift_both`, `no_shift`, or comma separated list of Tag names, to compare shifted or non shifted widths for left and right shift expressions.

- [*handle_zero_padding*](#): Default value is no. Set the value of the parameter to yes or tag name to perform leading zero expansion and truncation of RHS and LHS of an assignment. This is performed only if the RHS and LHS of the assignment are static. In addition, if the parameter is set to yes, the tag performs zero expansion or truncation if the RHS of the assignment is a concatenation and the first element is padded with zero.
- [*check_counter_assignment*](#): Default value is no. In this case, no violation is reported for counter cases. Set this parameter to yes to report violations for counter cases. In the [*W164b*](#) Tag, this parameter is applicable for **VHDL** only.
- [*check_natural_width_of_multiplication*](#): Default value is no. Set the parameter to yes, to calculate the width of the multiplication expressions as per the natural width even if the `nocheckoverflow` parameter is set to yes. Set the parameter to a comma/ space-separated list of tags to enable (the parameter) for the tags specified in the list.
- [*process_complete_condop*](#): Default value is no. Set the value of the parameter to yes to enable the Tag checking on both operands of the condop assignment.
- [*treat_concat_assign_separately*](#): Default value is no. Set the value of the parameter to yes to use this parameter to report violation for each bucket assignment in unpacked array separately. For a packed array, a violation is reported for the whole array.

Constraint(s)

None

Messages and Suggested Fix

Verilog

The following message appears at the location where the LHS expression `<lexpr>` of width `<widthl>` of an assignment is greater than the RHS expression `<rexpr>` of width `<widthr>`:

[WARNING] LHS: '`<lexpr>`' width `<widthl>` is greater than RHS: `<rexpr>` width `<widthr>` in assignment [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the signal.

Potential Issues

A violation is reported when the LHS width is greater than the RHS width.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, click [How to Debug and Fix](#).

VHDL**Message 1**

The following message appears at the location where the width `<widthl>` of LHS expression `<exprl>` of an assignment is greater than the width `<widthr>` of the RHS expression `<exprr>`:

[WARNING] LHS: '`<exprl>`' (width `<widthl>`) is greater than RHS: '`<exprr>`' (width `<widthr>`) in assignment [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the signal.

Potential Issues

A violation is reported when the LHS width is greater than the RHS width in an assignment.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, click [How to Debug and Fix](#).

Message 2

The following message appears at the location where the range of LHS expression `<exprl>` is wider than the range of RHS expression `<exprr>`:

[WARNING] Range of LHS '`<exprl>`' (`<lrange>`) is wider than the range of RHS '`<exprr>`' (`<rrange>`)

Where, *<rrange>* and *<lrangle>* refer to the range of RHS and LHS expressions, respectively.

Potential Issues

A violation is reported when the range of the LHS expression is wider than the range of the RHS expression.

Consequences of Not Fixing

Bit-width mismatch between the LHS and RHS of a signal assignment is allowed but may lead to inadvertent errors.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the line where the width mismatch is detected.

To resolve the violation, explicitly extend as necessary since it is more readable, instead of relying on the default behavior. Examine the logic to ensure that the truncation does not affect the behavior of the design.

In case of counters specified as integers with range, fix may not be required as user may have put the check to avoid overflow this is the normal practice to use integer range for counters.

Example Code and/or Schematic

Verilog

Example 1

Consider the following example code in which the *strict* parameter reports a violation:

```
wire a,b;
assign a = b + 4 ; //Violation reported with strict
```

Example 2

Consider the following example, in which no violation is reported:

```
A [12:0] = { 3'b000, b[9:0] } ;
```

In this case, the LHS width is 13. This LHS width lies between original RHS width 10, which is calculated without considering leading zeros and RHS width 13, which is calculated after considering leading zeros. Therefore, no violation is reported.

Example 3

Consider another example, in which no violation is reported:

```
A [11:0] = { 3'b000, b[9:0] } ;
```

In this case, the LHS width is 12. This LHS width lies between original RHS width 10, which is calculated without considering leading zeros and RHS width 13, which is calculated after considering leading zeros. Therefore, no violation is reported.

Example 4

Consider another example, in which a violation is reported:

```
A [14:0] = { 3'b000, b[9:0] } ;
```

In this case, the LHS width is 15. This LHS width does not lie between original RHS width 10, which is calculated without considering leading zeros and RHS width 13, which is calculated after considering leading zeros. Therefore, a violation is reported.

Example 5

Consider the following example, in which violations are reported for a concatenation operator:

```
module test3(d,clk,rst,q1, q2, q3, q4, q5);
input d,clk,rst;
output reg [4:0]q1;
output reg [9:0]q2;
output reg [7:0]q3;
output reg [12:0]q4;
output reg [4:0]q5;
wire [4:0]datain;

always @(posedge clk)
begin
  if(rst)
  begin
    q1 = {5'b00000, datain};
    q2 = {5'b00000, datain};
    q3 = {5'b00000, datain};
    q4 = {5'b00000, datain};
    q5 = {5'b00100, datain};

    q2 = {5'b00000, 3'b000, datain};
```

```

    q3 = {5'b00000, 3'b000, datain};

    q2 = {5'b00010, 3'b000, datain};
    q2 = {5'b00000, 3'b010, datain};
    end
end
endmodule

```

The *W164b* Tag reports the following violations for the above example:

LHS: 'q4' width 13 is greater than RHS: '{5'b00000, datain}' width 5 in assignment [Hierarchy: ':test3']

Example 6

Consider the following example:

```
assign out2 [8:0] = {3'b001,in1,8'b00001111};
```

In the above example, the Tag preserves all leading zeros in case of out2 assignment, which is 3'b001. Therefore, the width of the assignment is calculated as 12 bits.

Example 7

The following example demonstrates width calculation for the *W164b* Tag when the *handle_shift_op* parameter is set to *shift_both*:

```

wire [7:0] w1;
wire [1:0] w2, w3, w4;
assign w1 = w2 << w3; //RHS Width is 5
assign w1 = 3'b110 << w2; //RHS Width is 6
assign w1 = w2 << (w3 + w4); //RHS Width is 5
assign w1 = w2[0] << (w3 + w4 + 5'd0); //RHS Width is 7
assign w1 = w2 << (w3 - w4); //RHS Width is 5
assign w1 = w2 << {w3[0],w4[0]}; //RHS Width is 5
assign w1 = w2 << {w3[0],1'b0}; //RHS Width is 4
assign w1 = 3'h001 << w3; //RHS Width is 4
assign w1 = {1'b0, w2[0]} << w3[0]; //RHS Width is 2

```

Example 8

The following example demonstrates width calculation for the *W164b* Tag when the *handle_shift_op* parameter is set to *shift_both*:

```

module m1();
wire [40:0]a, b, c;

assign a[3:0] = b[3:0] << 1;           //RHS width = 5
assign a[3:0] = b[3:0] << c[1];       //RHS width = 5
assign a[3:0] = 11 << c[0];           //RHS width = 5
assign a[31:0] = 11 << c[0];          //RHS width = 5

assign a[3:0] = b[3:0] >> 1;           //RHS width = 3
assign a[2:0] = b[3:0] >> c[1];       //RHS width = 4
assign a[2:0] = 11 >> c[1];           //RHS width = 4
endmodule

```

Example 9

Consider the following example in which the [check_unsign_overflow](#) parameter is set to yes:

```

wire [5:0] a,b;
wire [8:0] d;
assign d = {{2{a[5]}},a} + {{2{b[5]}},b};

```

The *W164b* Tag does not report a violation for the unsigned expression.

Example 10

Consider the following example in which the *W164b* Tag reports violations for all three assignments when the [check_static_value](#) parameter is set to yes:

```

wire [4:0] AAA, BBB;
wire [7:0] CCC;
assign CCC = 6'b100111;
assign CCC = 120;
assign CCC = (AAA + 1) - BBB;

```

VHDL

Example 1

Consider the following example code in which the `strict` parameter reports a violation:

```

library ieee;
use ieee.std_logic_1164.all

```



```
a (5 downto 0) <= b ( 2 downto 0) + 4;
a (5 downto 0) <= 4 + b (2 downto 0);
```

Example 2

Consider the following example:

signal SIG1, SIG3	:Unsigned (3 downto 0)
signal SIG2	:Unsigned (2 downto 0)
signal SIG4	:Unsigned (4 downto 0)
SIG3 <= SIG1+SIG2 //no violation (left width 4 right shift 4)	

In the above example, since both the operands are variables, then the max width is taken. Also, the W164b Tag does not report a violation in this case as the LHS width is equal to the RHS width, which is 4.

Example 3

Consider the following example:

SIG3 <= SIG1 + 17 //no violation (width is 4 on both sides)	
SIG3 <= SIG4 - 17 //violation (left 4, right 5)	

In the above example, the Tag reports violation for the second expression as LHS is 4 and the RHS is 5.

Example 4

Consider the following example:

signal SIG1	:Unsigned (3 downto 0)
signal SIG2	:Unsigned (2 downto 0)
signal SIG3	:Unsigned (1 downto 0)
signal SIG4	:Unsigned (4 downto 0)

signal SIG5	:Unsigned (9 downto 0)
SIG4 < =SIG2*SIG3 //no violation (left width 5 right width 5)	

In the above example, the Tag does not report violation. Also, as both the operands are variables, the RHS width is the sum of both the variables.

Example 5

Consider the following example:

signal SIG1	:Unsigned (3 downto 0)
signal SIG2	:Unsigned (2 downto 0)
signal SIG3	:Unsigned (1 downto 0)
SIG1 < =SIG2*SIG3 //violation (left width 4 right width 5)	

In the above example, since both the operands are variables, the RHS width is the sum of both the variables. Therefore, the LHS width is 4 and RHS width is 5. As a result, the Tag reports a violation in this case.

Example 6

Consider the following example:

SIG1 <= SIG3 * 17 //violation (left 5, right 4)	
SIG5 <= SIG2 * SIG3//violation (left 10, right 5)	

In the above example, one operand is a variable and the other variable is static. Therefore, the RHS width is 2*(Variable width).

Example 7

Consider the following example:

signal SIG1	:Unsigned (3 downto 0)
signal SIG2	:Unsigned (3 downto 0)
signal SIG3	:Unsigned (1 downto 0)

```

signal SIG4          :Unsigned (4 downto 0)
SIG1 <=SIG2/SIG3 //No violation (left width 4
                right width 4)

```

In the above example, both the operands are variables. Therefore, the RHS width is the left operand width, that is, the width of both LHS and RHS is 4. As a result, the Tag does not report a violation in this case.

Example 8

Consider the following example:

```

SIG1 <= SIG2 / 17 //no violation (width is 4 on
                both sides)
SIG1 <= SIG4 / 17 //violation (left 4, right 5)
SIG1 <=17 / SIG2 //no violation (left 4, right 4)

```

In the above example, one operand is a variable and the other variable is static. Therefore, the RHS width is variable width. The Tag reports a violation for the second expression as LHS width is 4 and the RHS width is 5. Also, the Tag does not report a violation for first and third expression as the LHS and RHS width for both the expressions is 4.

Example 9

Consider the following example:

```

architecture rtl of rgx_pbe_gammacmp_lut is
    type aa_type is array(3 downto 0) of std_logic_vector(11
    downto 0);
    constant const_gamma_table : aa_type := (
        X"1ff",
        X"ff",
        X"2ff",
        X"00f"
    );

    signal t : std_logic_vector( 11 downto 0);
    signal a : std_logic_vector( 13 downto 0);
begin

```

```
t <= const_gamma_table(to_integer(unsigned(a)));  
end rtl;
```

In the above example, by default, a violation is reported because the LHS width is 12 and RHS width is 10, which is maximum width among the initialized elements of the array (that is the width of `x"2ff"`). When the [nocheckoverflow](#) parameter is set to `yes`, for RHS, the declared width of an element of array is considered (12), hence no violation is reported.

Default Severity Label

Warning

Tag Group

Lint_Elab_Tags

Reports and Related Files

None

W167

Module has no input or output ports

Language

Verilog, VHDL

Description

The W167 flags modules that do not have any port interface (no input or output ports).

Message Details

Verilog

The following message appears at the location where a module `<module-name>` is defined without any inputs or outputs:

```
Module <module-name> has no ports
```

VHDL

The following message appears at the location where a design unit is defined without any inputs or outputs:

```
Module has no input/output ports
```

Severity

Warning

Suggested Fix

If this is a top-level testbench, there is nothing to fix. Otherwise you can only read to and write from this design unit through global variables, which is unsynthesizable. It may sometimes be OK for simulation monitors (though would still be considered by many to be poor coding style).

W17

Prefer full range of a bus/array in sensitivity list. Avoid bits or slices

Language

Verilog, VHDL

Description

The W17 Tag flags those arrays in sensitivity lists that are not completely specified.

While the logic inside a combinational construct (`always` construct or `process` construct) may be sensitive only to certain index or bit-slices of an array, using only index or bit-slices in the construct's sensitivity list can lead to subtle errors if the construct design is subsequently changed.

Message Details

Verilog

The following message appears at the location of an `always` construct's sensitivity list in hierarchy *<hier-path>* that has a bus *<bus-name>* that is not completely specified:

Bus '*<bus-name>*' specified in sensitivity list with incomplete range [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the complete hierarchical name of the containing scope excluding functions/tasks.

VHDL

The following message appears at the location of a `process` construct's sensitivity list in hierarchy *<hier-path>* that has an array *<array-name>* of type *<sig-type>* but the array is not completely specified:

Complete vector *<sig-type>* '*<array-name>*' is not used in the sensitivity list. [Hierarchy: '*<hier-path>*']

Where, *<sig-type>* can be `signal`, `port`, or `variable` and *<hier-path>* is the complete hierarchical name of the containing scope excluding

Lint Tags

subprograms.

Severity

Warning

Suggested Fix

This is partly a question of taste. If you specify the whole bus and some bits are not read in the block, you may get a warning that the sensitivity list is over-specified. In such cases, it is not possible to satisfy both requirements simultaneously, so you must decide which requirement you want to ignore. From an implementation point of view, it is safest to ignore this Tag.

W171

Case label is non-constant.

Language

Verilog

Description

The W171 Tag flags expressions used as case clause labels.

Typical state machine descriptions require constant case clause labels as in the following example:

```
case (x)
  1: ...
  2: ...
  25: ...
  ...
endcase
```

Some designers may represent complex if conditions as a case construct as in the following example:

```
case (sel)
  expr1: ...
  expr2: ...
  ...
endcase
```

The W171 Tag flags such case constructs.

The W171 Tag flags non-static case clause labels provided they are not assigned a constant value elsewhere in the design.

By default, the `check_const_selector` parameter is set to `no` and the Tag gives violation when case label is non-constant. If you set the `check_const_selector` parameter to `yes`, this Tag checks whether case selector is constant or not. The Tag does not report any violation if case selector is a constant and case label belongs to one-hot operation, although case

label is a variable.**Message Details**

The following message appears at the location where a non-constant case clause label is encountered:

Case-label is not constant

Tag Severity

Warning

Suggested Fix

Generally nothing to fix, unless you find a case which does not match expectations.

Examples

Consider the following example where one clause of the case construct has a non-constant label (sel):

```
module test;
  reg [1:0] addr_int, data_out, data_in;
  reg clk, sel;

  always @(posedge clk)
    case (1)
      0      : data_out <= data_in;
      (sel)  : data_out <= data_in;
      2      : if (~sel) data_out <= data_in;
      3      : data_out <= 2'bz;
      default : data_out <= 2'bx;
    endcase
endmodule
```

For this example, VC SpyGlass Lint generates the W171 Tag message.

W175

A parameter/generic has been defined but is not used

Language

Verilog, VHDL

Description

The *W175* Tag reports parameters and generics that are never used in the design.

While such constructs are allowed, the Tag helps you clean up your design.

NOTE: *For Verilog designs, the Tag also reports a violation for unused local parameters.*

Message Details

Verilog

The following message appears at the location where a parameter *<param-name>* is declared that is never used in the design:

Parameter '*<param-name>*' declared but not used

VHDL

The following message appears at the location where a generic *<gen-name>* is declared that is never used in the design:

The generic '*<gen-name>*' is not used

Tag Severity

Warning

Suggested Fix

Confirm that parameter/generic is really redundant. Remove it if possible to reduce clutter in the design and to reduce warning messages from VC SpyGlass Lint.

W18

Do not infer latches

Language

Verilog, VHDL

Description

The W18 Tag flags latches in the design.

Except where explicitly intended, latch inference is usually caused by an oversight — for example, failure to define a default value for a variable defined in a conditional statement — and may lead to incorrect behavior. Further, inferred latches in a register-based design cause problem for VC SpyGlass Lint DFT tools.

By default, the W18 Tag does not report latches inferred from both .sglib and .lib libraries. Set the [reportLibLatch](#) parameter to `yes` to report library latches.

Message Details

The following message appears at the location where a latch is inferred for signal `<sig-name>`:

```
Latch inferred for signal '<sig-name>'
```

Severity

Info

Suggested Fix

Check the inference to make sure it is what you intended. If not, prevent latch inferences by providing an explicit `else` clause at the end of the `if` statement, or `default` clause at the end of the `case` statement, to prevent inferring the latch.

Examples

In the following example, a latch is inferred for signal `q`:

```
process (reset, d)
```

```
begin
  if (reset = '1') then
    q <= d;
  end if;
end process;
```

For this example, VC SpyGlass Lint generates the following message:

Latch inferred for signal 'q'

W182c

Identifies the time declarations which are not synthesizable

When to Use

Use this Tag to identify the time declarations which are not synthesizable by some synthesis tools.

Description

The *W182c* Tag reports the "time" variable declarations which are not synthesizable by some synthesis tools.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for the `time` declarations which are not synthesizable by a synthesis tool:

[ERROR] 'time' declaration are not synthesizable

Potential Issues

A violation is reported when a time variable is declared.

Consequences of Not Fixing

The `time` declarations have no physical equivalent. Therefore, such declarations are not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location

where the offending `time` data type is used. This is un-synthesizable by several commercially available synthesis tools.

To resolve the violation, remove the `time` declarations from the logic intended for synthesis. Use these in testbenches only.

Example Code and/or Schematic

Consider the following example:

```
module test(in1, in2);  
input in1, in2;  
time t;  
endmodule
```

In the above example, the W182c Tag reports a violation as a `time` data type is used.

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

None

W182g

Identifies the tri0 net declarations which are not synthesizable

When to Use

Use this Tag to identify the `tri0` net declarations which are not synthesizable by some synthesis tools.

Description

The *W182g* Tag reports violations for the `tri0` net declarations used in the design.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is displayed for a `tri0` declaration which is not synthesizable by a synthesis tool:

[Error] 'tri0' net types may not be synthesizable

Potential Issues

A violation is reported when a `tri0` net is used in the design.

Consequences of Not Fixing

The `tri0` and `tri1` net declarations represent connections with resistive pull-down or pull-up. These nets are useful in developing simulation models, but they do not have an unambiguous physical counterpart because mapping depends upon the target technology. For example, some

technologies may not support tristate operations.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the offending "tri0" is declared. This is un-synthesizable by several commercially available synthesis tools.

To resolve the violation, replace the `tri` declarations with the standard signal declarations and instantiate pull-up or pull-down cells from the target technology library, as required.

Example Code and/or Schematic

Consider the following example:

```
module test (y,s0,d1,d0);  
input s0, d1, d0;  
output y;
```

```
tri0 y;
```

```
assign y = s0 ? d0 : d1;  
endmodule
```

In the above example, the W182g Tag reports a violation as a `tri0` net is used in the design.

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

None

W182h

Reports tri1 net declarations that are not synthesizable

When to Use

Use this Tag to detect tri1 net declarations that are not synthesizable

Description

The *W182h* Tag reports the tri1 net declarations even if the net is not used in the design.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a tri1 declaration is encountered.

[ERROR] 'tri1' net types may not be synthesizable

Potential Issues

A violation is reported when a tri1 net is used in the design.

Consequences of Not Fixing

The tri0 and tri1 declarations represent connections with resistive pull-down or pull-up. These nets may not be synthesizable by some synthesis tools and are useful only in developing simulation models. They do not have an unambiguous physical counterpart because mapping depends upon the target technology. For example, some technologies may not

support tristate operations.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the offending "tri1" has been declared. This is un-synthesizable by several commercially available synthesis tools

Replace the `tri` declarations with standard signal declarations and instantiate pull-up or pull-down cells from the target technology library, as required.

Example Code and/or Schematic

Consider the following example:

```
module test(y,s0,d1,d0);  
  input s0, d1, d0;  
  output y;  
  tri1 y;  
  
  assign y = s0 ? d0 : d1;  
endmodule
```

In the above example, the W182h Tag reports a violation as the `tri1` net is used.

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

None

W182k

Reports trireg declarations that are not synthesizable

When to Use

Detect trireg declarations that are not synthesizable.

Description

The *W182k* Tag reports `trireg` register declarations.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a `trireg` declaration is encountered.

[ERROR] 'trireg' net types are not synthesizable

Potential Issues

A violation is reported when a `trireg` net is used in the design.

Consequences of Not Fixing

The `trireg` declarations represent charge storage that does not have a clear physical interpretation in synthesis. Such descriptions are useful only in developing simulation models.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the offending `trireg` is declared. This is un-synthesizable by

several commercially available synthesis tools.

Do not try to infer charge storage elements. Use library memory models if appropriate.

Example Code and/or Schematic

Consider the following example:

```
module test(y,s0,d1,d0);  
input s0, d1, d0;  
output y;
```

```
    trireg y;
```

```
    assign y = s0 ? d0 : d1;  
endmodule
```

In the above example, the W182k Tag reports a violation as a `trireg` net, `y`, is used in the design.

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

None

W182n

Reports MOS switches, such as `cmos`, `pmos`, and `nmos`, that are not synthesizable

When to Use

Detects MOS switches that are not synthesizable.

Description

The *W182n* Tag reports MOS switches, such as CMOS, PMOS, NMOS, used in the design. These switches should be used for modeling only and are not synthesizable by most synthesis tools.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a switch declaration `<switch-name>` is encountered.

[ERROR] Switch '`<switch-name>`' is not synthesizable

Potential Issues

A violation is reported when a MOS switch is used in the design.

Consequences of Not Fixing

Except for custom or analog design, transistor-level design is generally discouraged because behavior and timing are difficult to predict under all possible circumstances.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the offending switches (CMOS/PMOS/NMOS) are declared. This is un-synthesizable by several commercially available synthesis tools.

Do not try to infer switch elements. Use predefined and characterized library cells that have well-defined characteristics and are silicon-proven.

Example Code and/or Schematic

Consider the following example:

```
module test(out, in1, in2);  
input in1,in2;  
output out;  
wire out1,out2;  
wire n;  
cmos (out,in1,in2,n);  
pmos (out1,in1,in2);  
nmos (out2,in1,in2);  
endmodule
```

In the above example, the W182n Tag reports a violation as MOS switches are used in the design.

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

None

W187

The 'default' or 'others' clause should be the last clause in a case statement

Language

Verilog, VHDL

Description

The W187 Tag flags those case constructs where the default clause (Verilog) or others clause (VHDL) is not the last clause in the construct.

While the default clause (or others clause) can be placed anywhere in the case construct, it is recommended to place it as the last clause for better understanding.

Message Details

Verilog

The following message appears at the first line of a case construct where the default clause is not the last clause in the construct:

```
default clause should be last clause in a case stmt
```

VHDL

The following message appears at the first line of a case construct where the others clause is not the last clause in the construct:

```
OTHERS clause should be last clause in a case stmt
```

Tag Severity

Warning

Suggested Fix

Move the default clause (Verilog) or others clause (VHDL) to the end of the case statement.

Verilog Examples

Consider the following example where the default clause of the case construct is not the last clause:

```
module test;
  reg [1:0] addr_int, data_out, data_in;
  reg clk, clk1;
  reg [1:0] sel;
  reg [3:0] a_1, b_1;

  always @(posedge clk or posedge clk1)
    case (sel)
      default : data_out <= 2'bx;
      0       : data_out <= data_in;
      1       : data_out <= data_in;
      2       : if (a_1 == b_1) data_out <= data_in;
      3       : if (a_1 != b_1) data_out <= 2'b11;
    endcase
endmodule
```

For this example, VC SpyGlass Lint generates the W187 Tag message.

W188

Do not write to input ports

Language

Verilog

Description

The W188 Tag flags assignments to input ports.

Writing to an input port creates a wire-or inside the module. Also it creates a possibility for the port to behave as an inout port, which may drive the external logic to a conflict, where a user of the module assumed the port to always behave as an input port.

Message Details

The following message appears at the location where an input port `<port-name>` is assigned a value:

```
input port '<port-name>' should not be assigned any value
```

The following message appears at the location where an input port `<port-name>` is instantiated:

```
input port '<port-name>' should not be connected to an instance  
output port
```

Tag Severity

Warning

Suggested Fix

Buffer or gate the internal feedback to ensure it cannot drive back out through the input port.

W189

Nested Synopsys `translate_off` comments

Language

Verilog

Description

The W189 Tag flags nested `translate_off` comments.

At the beginning of a Verilog file, translation is enabled for synthesizable code. If required, later in the file a block can be treated as non translatable by placing a `translate_off` comment at the beginning of the block and a `translate_on` comment at the end of the block. Hence a nested `translate_off` comment may indicate a possible missing of corresponding `translate_on` comment.

Message Details

The following message appears when a nested `translate_off` comment is encountered:

For a `translate_off`, a `translate_on` should appear, before next `translate_off`

This Tag also highlights the previous nested `translate_off` comment.

Severity

Warning

Suggested Fix

Make sure that for every `translate_off`, a `translate_on` appears, before the next `translate_off`.

Examples

Consider the following example:

```
module test1(outp);  
    output reg outp;
```

Lint Tags

```

        reg [3:0] inp;
        reg [1:0] sel;

//synopsys translate_off
        initial inp=3'b000;
//synopsys translate_off
        initial begin
            sel=2'b00;
        end
//synopsys translate_on
        always @(sel,inp)
            case(sel)
                2'b00 :outp = inp[0];
                2'b01 :outp = inp[1];
                2'b10 :outp = inp[2];
                2'b11 :outp = inp[3];
                default:
                    outp = 1;
            endcase

    endmodule

```

In the above example, the *W189* Tag reports a violation because the second *synopsys translate_off* appears before *synopsys translate_on*. The following message is reported by this example:

For a *translate_off*, a *translate_on* should appear, before next *translate_off*

W19

Reports the truncation of extra bits

When to Use

Detects truncation of extra bits.

Description

Reports a violation when you have specified a constant value wider than the specified width of the constant.

For example, constant 4'b10110 is truncated to 4'b0110 and a message is flagged.

Tag Checking for Binary Based Numbers

The W19 Tag reports violation for the following cases:

- If significant bits are truncated.
- For loss of non-significant bits, if you set the value of the [strict](#) parameter to *yes*.

Tag Checking for Decimal Based Numbers

The W19 Tag reports a violation if significant bits are truncated.

Tag Checking for Hexadecimal and Octal Based Numbers

The W19 Tag reports violation for the following cases:

- If significant bits are truncated.
- If based word contains more bits than specified size and there is truncation of non-significant bits. However, the W19 Tag does not report a violation if based word can fit well within the specified size, as this is a valid design practice.

Tag Exceptions

The W19 Tag does not check for unused macro definitions and unused parameters.

Language

Verilog

Default Weight

5

Parameter(s)

- *strict*: The default value of the parameter is `no` and the Tag W19 does not report truncation of zeroes for binary based numbers. If you set the the value of this parameter to `yes` or the Tag name, the Tag also reports truncation of zeroes for binary based numbers.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a constant `<const-name>` is being truncated as it is wider than the usage context:

[WARNING] Constant `<const-name>` will be truncated

Potential Issues

A violation is reported when you have specified a constant value wider than the specified width of the constant.

Consequences of Not Fixing

When constant value is wider than the width of the constant, it results in truncation of extra bits. This in turn leads to data loss and unexpected code behavior.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where truncation of extra bit on constant number is violated. Search for the based number that is described in the violation message in the highlighted line.

To resolve the violation, determine the width specification and the constant value. Fix the width specification, if incorrect.

Example Code and/or Schematic

Example 1

Consider the following example:

```
module operator(clk1,out1);

    input clk1;
    output out1;
    reg out1;

    always @(posedge clk1) begin
        out1 = 1'b101; //(Constant 1'b101 will be truncated)
        if(out1 == 2'b0101)//Constant 2'b0101 will be truncated
    )
        begin
            end
    end

endmodule
```

In the above example, the *W19* Tag generates the truncation message for the constants, 1'b101 and 2'b0101, as the constants are wider than the usage context.

Example 2

Consider the following example:

2'b111

In the above example, the Tag reports a violation.

Example 3

Consider the following example:

2'b011

In the above example, the Tag reports a violation for this expression, if you have set the `strict` parameter to `yes`.

Example 4

Consider the following example:

2'd15, 1'd3

In the above example, the W19 Tag reports a violation.

Example 5

Consider the following example:

2'd03, 4'd0014

In the above example, the Tag does not report a violation for this expression.

Example 6

Consider the following example:

3'hF, 2'o7

In the above example, the Tag reports a violation.

Example 7

Consider the following example:

3'h01, 3'o01

In the above example, the Tag reports a violation.

Example 8

Consider the following example:

3'h1, 2'o1

In the above example, the Tag does not report a violation for this expression.

Default Severity Label

Warning

Tag Group

Assign

Reports and Related Files

None

W190

Task or procedure declared but not used

Language

Verilog, VHDL

Description

The W190 Tag flags tasks or procedures that are declared but not called in the design.

While describing unused tasks is allowed, the W190 Tag helps you clean up your design of unused tasks, thus reducing clutter in the design description or possibly preventing a potential name confusion error.

Message Details

Verilog

The following message appears at the location where a task *<task-name>* is declared that is not called anywhere:

Task '*<task-name>*' declared but not used

VHDL

The following message appears at the location of a procedure declaration *<proc-name>* that is not used in the design:

The procedure '*<proc-name>*' is not used

Tag Severity

Warning

Suggested Fix

If the task or procedure is confirmed to be redundant, remove it if possible. This will reduce clutter in the design and also reduce warning messages from VC SpyGlass Lint.

W191

Function declared but not used

Language

Verilog, VHDL

Description

The W191 Tag flags functions that are declared but not used in the design.

While unused functions are allowed, the W191 Tag helps you clean up your design of unused functions, thus reducing clutter in the design description or possibly preventing a potential name confusion error.

Message Details

The following message appears at the location of a function declaration *<func-name>* that is not used in the design:

Verilog

Function '*<func-name>*' declared but not used

VHDL

The function '*<func-name>*' is not used

Tag Severity

Warning

Suggested Fix

If the function is confirmed to be redundant, remove it if possible. This will reduce clutter in the design and also reduce warning messages from VC SpyGlass Lint.

W192

Empty block

Language

Verilog

Description

The W192 Tag flags empty `begin-end` blocks.

A `begin-end` block containing no logic may create clutter and reduce readability. It may also represent an unintended edit.

Message Details

The following message appears at the start of an empty `begin-end` block:
`Block does not contain any statement`

Tag Severity

Warning

Suggested Fix

Check to make sure you really intend the block to be empty.

W193

Empty statement

Language

Verilog

Description

The W193 Tag flags empty statements (isolated semicolons).

An empty statement may create clutter and reduce readability. It may also represent an unintended edit or an unintended comment.

You can set the value of the [report_if_blocks_only](#) parameter to `yes` to report violations only when a semicolon is used with the `if`, `else`, and `else-if` statements.

Message Details

Message 1

The following message appears at the location where an isolated semicolon is encountered:

Empty statement

Message 2

The following message appears for an extra semicolon used, if the [report_semicolon](#) parameter is set to `yes`:

Extra semicolon used

Tag Severity

Warning

Suggested Fix

Check to make sure you really intend the statement to be empty.

W208

Nested Synopsys `translate_on` comments

Language

Verilog

Description

The W208 Tag flags the nested `translate_on` comments.

At the beginning of Verilog file , translation is enabled for synthesizable code. If required, later in the file a block can be treated as non translatable by placing a `translate_off` comment at the beginning of the block and a `translate_on` comment at the end of the block. Hence a nested `translate_on` comment may indicate a possible missing of corresponding `translate_off` comment.

Message Details

The following message appears when a nested `translate_on` comment is encountered:

No associated `translate_off` found

Severity

Warning

Suggested Fix

Make sure that no `translate_on` comment is followed by a next `translate_on`, unless a `translate_off` appears before this next `translate_on`.

W210

Number of connections made to an instance does not match number of ports on master

Language

Verilog, VHDL

Description

The W210 Tag flags a violation for module or interface instances with unconnected ports.

The W210 Tag flags message only when number of terminals are less than the number of ports. The W210 Tag ignores ports that are intentionally kept open (by passing extra comma [Verilog] or by connecting them to open [VHDL]).

If any of the unconnected ports are inputs or inouts, this is an error. It may or may not be an error if the unconnected ports are outputs.

NOTE: *The W201 Tag does not report violations for constructs in the dead-code segments of a generate block.*

Parameters

- *set_message_severity*: The default value is `no` and the W210 Tag reports all unconnected ports with severity as **Warning**. Set this parameter to `yes` to report unconnected input or inout ports with messages severity as **Error**.
- *ignore_inout*: Default value is `no`. Set the value of this parameter to `yes` to not report any violation for inout port.

Message Details

Verilog

The following message appears at the location of instance `<inst-name>` of master module `<master-du-name>` where the port, `<port-name>`, is not connected:

[WARNING] Instance `<inst-name>`(master: `<master-du-name>`) has too few ports - `<port-type>` port '`<port-name>`' not connected

where, *<port-type>* can be input or inout.

NOTE: *If the `set_message_severity` parameter is set to yes, the severity of this message will be Error.*

VHDL

The following message appears at the location of a component instantiation *<inst-name>*, of master module *<master-du-name>*, which has no port corresponding to the formal port *<port-name>*:

[WARNING] Port Map has no actual corresponding to formal *<port-type>* '*<port-name>*' (or some of its bits) in instance '*<inst-name>*' (master: *<master-du-name>*)

where, *<port-type>* can be input or inout.

NOTE: *If the `set_message_severity` parameter is set to yes, the severity of this message will be Error.*

Examples

Consider the following example code:

```
module top(input in, mis, output q);
    test u2(mis);
endmodule
module test(input in1, input in2);
endmodule
```

For the above example, the default violation message is:

[WARNING] Instance u2(Master: test) has too few ports - Input port 'in2' not connected

When the [set_message_severity](#) parameter is set to yes, the same message is generated with the message severity as Error.

[ERROR] Instance u2(Master: test) has too few ports - Input port 'in2' not connected

Tag Severity

Warning/Error

Suggested Fix

Make sure that all inputs and inouts are connected (at least tied off

Lint Tags

appropriately if they are don't care and that all outputs you expect to be used are connected.

W213

Reports PLI tasks or functions that are not synthesizable

When to Use

Use this Tag to identify PLI tasks and functions used in the design.

Description

The *W213* Tag reports PLI tasks and functions used in the design.

Language

Verilog

Default Weight

5

Parameter(s)

ignore_pli_tasks_and_functions: By default, the *ignore_pli_tasks_and_functions* parameter is not set. Set this parameter to a comma-separated list of PLI tasks or functions that should be ignored by the *W213* Tag.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a PLI task or a function *<name>* is encountered.

[WARNING] PLI Task/Function '*<name>*' is not synthesizable

Potential Issues

A violation is reported when a PLI task or function is used in the design.

Consequences of Not Fixing

The PLI tasks or functions, such as `$display`, have no physical meaning and therefore are not synthesizable.

How to Debug and Fix

Lint Tags

Double-click the violation message. The HDL window highlights the location where the offending PLI Task/Function is used. This is un-synthesizable by several commercially available synthesis tools.

No fix is required if these functions appear in testbenches or simulation models. If you need to include these functions in RTL for debug purposes, surround them with the `translate_off` and `translate_on` pragmas for your target synthesis tool.

Example Code and/or Schematic

Consider the following example:

```
module test (in1, clk);  
  
    input in1, clk;  
    always @ (clk)  
        $display ("Value of in1 %b\n", in1);  
  
endmodule
```

In the above example, the W213 Tag reports a violation as the PLI task, `$display`, is used in the design.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

W215

Reports inappropriate bit-selects of integer or time variables

When to Use

Use this Tag to identify inappropriate bit-selects of integer or time variables.

Description

The *W215* Tag reports bit-selects of integer or time variables.

NOTE: *The W215 Tag supports generate-if, generate-for, and generate-case blocks.*

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a bit-select of a variable `<var-name>` of type `<var-type>` is used:

[WARNING] Inappropriate bit select for `<var-type>` variable: "`<var-name>`"

Where `<var-type>` can be integer or time.

Potential Issues

A violation is reported when a bit-select is used for an integer or time variable.

Consequences of Not Fixing

Bit-selects are not meaningful for integer variables and time variables.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where a bit-select is used for a variable whose data type is integer or time. You can confirm the data type of the variable by scrolling up the HDL to the most recent definition of the signal.

To fix the violation, remove the bit-select. If you want to select a bit from an integer variable, either mask and shift, or assign the integer to a `reg` and then do bit-select on the `reg`.

Example Code and/or Schematic

Consider the following example:

```
module W215_mod2(in1, sel, out1);
    input in1, sel;
    output out1;
    reg out1;

    integer i;
    initial
        i = 10;

    always@(in1 or sel)
        if(sel == 1'b1)
            out1 <= in1;
        else
            out1 <= i[0];

endmodule
```

In the above example, the *W215* Tag reports inappropriate bit-selects for the `int` variable.

Default Severity Label

Warning

Tag Group

Usage

Reports and Related Files

None

W216

Reports inappropriate range select for integer or time variable

When to Use

Use this Tag to identify inappropriate range selects for integer or time variables.

NOTE: *The W216 Tag supports generate-if, generate-for, and generate-case blocks.*

Description

The W216 Tag reports range selects of integer or time variables.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where range select of a variable `<var-name>` of a type `<var-type>` is used.

[WARNING] Inappropriate range select for `<var-type>` variable: "`<var-name>`"

Where `<var-type>` can be integer or time.

Potential Issues

A violation is reported when a range select is used for an integer or time variable.

Consequences of Not Fixing

Range selects are not meaningful for integer variables and time variables.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where integer or time variable is used with selected range.

Remove the range select. If you want to select a range from an integer variable, either mask and shift, or assign the integer to a `reg` and then do range select on the `reg`.

Example Code and/or Schematic

Consider the following example:

```
module W216_mod2(in1, sel, out1);
    input [1:0] in1;
    input sel;
    output [1:0] out1;
    reg [1:0] out1;

    integer i;

    initial
        i = 10;

    always@(in1 or sel)
        if(sel == 1'b1)
            out1 = in1;
        else
            out1 = i[1:0];

endmodule
```

In the above example, the *W216* Tag reports inappropriate range select for the *int_part_sel* variable.

Default Severity Label

Warning

Tag Group

Usage

Lint Tags

Reports and Related Files

None

W218

Reports multi-bit signals used in sensitivity list

When to Use

Use this Tag to identify multi-bit signals used in the sensitivity list.

Description

The *W218* Tag reports event expressions that check for an edge on a multi-bit signal.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where an edge specification on a multi-bit signal is encountered.

[WARNING] Edge specification should not be used for a multibit expression: '<multi-bit-expressions>'

Potential Issues

A violation is reported when an edge specification is used on a multi-bit expression.

Consequences of Not Fixing

Edge specifications for multi-bit expression is semantically incorrect. In such cases, only the changes on least significant bit are important. Also the direction of an edge on a multi-bit signal is not uniquely defined.

How to Debug and Fix

Lint Tags

Double-click the violation message. The HDL Viewer window shows the sensitivity list, where the multi-bit signal is used. The sensitivity list confirms that more than one bit of the signal is participating.

To fix the violation, use the edge expression on the appropriate bit of the multi-bit signal.

Example Code and/or Schematic

Consider the following example:

```
module test1(in1,clk,out1);  
input [2:0] in1, clk;  
output [2:0] out1;  
  
reg [2:0] out1;  
  
always @(posedge clk)  
out1 = in1;  
  
endmodule
```

In the above example, the *W218* Tag reports a violation as the event expression checks for the edge on a multi-bit signal.

Default Severity Label

Warning

Tag Group

Synthesis, Event

Reports and Related Files

None

W224

Multi-bit expression found when one-bit expression expected

Language

Verilog

Description

The W224 Tag flags multi-bit expressions where single-bit expressions were expected.

A condition expression should evaluate to a single-bit value. While the method for evaluating a multi-bit value as a truth value is well defined, this form is generally less readable and is known to lead to errors of interpretation.

In case of complex expressions, the W224 Tag will report violation only if the final value evaluated (for a condition `expr`) is not a single-bit value. For example, this Tag will report violation for the following expression

```
...
input C;
reg [3:0] H;
...
H = ( H - C ) ? 4'd14 : 4'd2;
...
```

Parameters

- [*ignore_parameters*](#): Default value is no. Set the value of the parameter to yes to enable the W224 tag to ignore violation for the parameters.
- [*ignore_integer*](#): Default value is no. Set the value of the parameter to yes to ignore violations on integer data type.
- [*use_natural_width*](#): Use this parameter to calculate the width using natural width.
- [*report_only_from_one_hierarchy*](#): By default, the value of the parameter is no, and the tag reports violations for multiple hierarchies. Set the value of the parameter to yes to suppress the hierarchy field and report only one violation per assignment.

Message Details

The following message appears at the location of use of a multi-bit expression `<expr>` where a single-bit expression was expected:

Multi-bit expression '`<expr>`' found when one-bit expression expected [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path.

Tag Severity

Warning

Suggested Fix

Select the appropriate bit and test that bit only.

Examples

In the following example, the constant `a_def` is of 16 bits (being an integer) and is used in the `if` statement that expects a scalar signal:

```
`define a_def 1
...
always @(posedge clk)
    if (`a_def)
        q <= d;
```

For this example, VC SpyGlass Lint generates the following message:
Multi-bit expression '1' found when one-bit is expected

W226

Case select expression is constant

Language

Verilog, VHDL

Description

Verilog

The W226 Tag flags case constructs where the case construct selector is a constant or a static expression.

Typical state machine descriptions require non-constant case construct selector as in the following example:

```
case (x)
  1: ...
  2: ...
  25: ...
  ...
endcase
```

Some designers may represent complex if conditions as a case construct as in the following example:

```
case (1)
  expr1: ...
  expr2: ...
  ...
endcase
```

The above case construct uses a constant case construct selector (1) and is equivalent to the following if construct:

```
if (expr1)
  else if (expr2)
    else ...
```

The W226 Tag flags such case constructs.

By default, the [check_const_selector](#) parameter is set to no. When you set the parameter to yes, the W226 Tag checks whether all case labels are

one-hot. If so, the Tag does not report a violation even if the case selector is a constant.

VHDL

The W226 Tag flags case constructs or selected signal assignments where the selector is a constant or a static expression.

A constant control expression or sub-expression evaluates to a TRUE/FALSE situation and thus, the construct is always executed or never executed.

By default, the W226 Tag does not check inside function bodies unless the *strict* Tag parameter is set.

By default, the value of the *ignore_generatefor_index* parameter is set to no. Set the value of this parameter to yes to ignore violations for case statements inside generate-for blocks, if the case selector is a loop variable of generate-for loop.

Message Details

Verilog

The following message appears at the start of a case construct where the selector is a constant or static expression *<expr>*:

Case select expression '*<expr>*' is constant

VHDL

The following message appears at the location of a statement *<stmt>* where the selector is a constant or static expression:

The *<stmt>* expression is constant

Where *<stmt>* can be CASE construct or selected signal assignment statement.

Tag Severity

Warning

Suggested Fix

Generally nothing to fix, unless you find a case which does not match expectations.

Examples

Verilog

Consider the following case construct that has a constant select expression:

```
case (1)
  0      : cc_data_out <=cc_data_in;
  (sel)  : cc_data_out <=cc_data_in;
  2      : if (~sel) cc_data_out <= cc_data_in;
  3      : cc_data_out <= 2'bz;
  default : cc_data_out <= 2'bx;
endcase
```

For this example, VC SpyGlass Lint generates the W226 Tag message.

VHDL

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all
case bit_vector'("11") is
  when "00" | "01" => q <= d;
  when others => q <= not d;
end case;
```

In the above example, the W226 Tag reports a violation because the case construct selector is a constant `bit_vector'("11")`.The following message is reported by this example:

The case expression is constant

W238

Mixing combinational and sequential styles

Language

Verilog

Description

The W238 Tag flags always construct where both blocking assignments and nonblocking assignments are used.

In general, you should use blocking assignments in combinational always constructs and non-blocking assignments in sequential always constructs. Mixing the two assignment types in the same always construct can lead to mismatches between pre- and post-synthesis simulation results. The task calls are considered as blocking assignments, so the violations are reported accordingly.

Message Details

The following message appears at the location where a blocking (non-blocking) assignment is encountered in an always construct after a non-blocking (blocking) assignment has already been found:

```
Mixing combinational and sequential styles
```

```
For a non-blocking or blocking assignment, the previous  
respective blocking or non-blocking assignment is also  
highlighted.
```

Tag Severity

Warning

Suggested Fix

Change all non-blocking assignments (`<=`) in combinational blocks to blocking (`=`) assignments. Change all blocking assignments in sequential blocks to non-blocking assignments.

Examples

In the following example, the task calls are considered as blocking

assignments, so the violations are reported:

```
module top(in, in2, in3, en, out, clk);
output out;
    input in, en, clk, in2, in3;
    reg out;
    wire a, b;

    always_ff @ (posedge clk)
    begin
        out <= a;    //non blocking
        convert(a, out);    //blocking
        out <= a;    //non blocking
        convert(a, out);    //blocking
    end

    task convert;
    input  temp_in1;
    output temp_out1;
    begin
        temp_out1 <= temp_in1;
    end
endtask

endmodule
```


W239

Reports hierarchical references that are not synthesizable

When to Use

Use this Tag to identify hierarchical references used in the design.

Description

The W239 Tag reports hierarchical references. Hierarchical references, such as `top.abx.clk`, are a useful way to probe design behavior during debug.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a hierarchical reference is encountered.

[WARNING] Hierarchical references may not be synthesizable

Potential Issues

A violation is reported when a hierarchical reference is used in the design.

Consequences of Not Fixing

Synthesis tools, in general, do not create connections corresponding to these references.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the

hierarchical references used in the design.

If you are using the reference for simulation, enclose it in code that can be disabled. This enables you to verify that functional behavior of the synthesis logic does not depend on this reference.

If you actually need to make connections in the synthesis logic, do so through the conventional approach, that is, add ports to lower level modules and make connections through those ports.

Example Code and/or Schematic

Consider the following example:

```
module top(output [3:0] w2);  
  assign w2 = temp.w1;  
  
endmodule  
module temp();  
  wire [3:0] w1;  
endmodule
```

In the above example, the W239 Tag reports a violation as a hierarchical reference, `temp.w1`, is used in the design.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

W240

An input has been declared but is not read

Language

Verilog, VHDL

Description

The *W240* Tag reports input ports that are never read in the module.

While such ports are allowed, the Tag helps you clean up your design.

NOTE: *For Verilog, inside the nested `for` loops, signals of user-defined type like structures, interfaces, etc. are considered fully set if used on the left-hand side of an expression and fully read if used on the right-hand side of an expression.*

By default, the Tag reports a violation if any bit of the input port is unread. If you set the [checkfullbus](#) parameter to `yes`, the Tag reports a violation only when the entire input port is unread.

By default, this Tag does not process large arrays. Set the [handle_large_bus](#) parameter to `yes` to process large arrays (greater than 50,000) and report violation if not used correctly.

By default, value [checkfullrecord](#) is set to `no`. Set this parameter to `yes` to not report violation for a VHDL record if at least one element of the record is read.

To see the bits of the input that are not read, refer to the [SignalUsageReport](#) report in the *VC SpyGlass Lint_reports/lint* directory.

By default, the value of [checkfullstruct](#) is set to `no`. Set this parameter to `yes` to not report a violation for structs if at least one element of the struct is read.

Message Details

Verilog

Message 1

The following message appears at the location where an input port `<port-name>` is declared but never read in the module:

Input '`<port-name>`' declared but not read. [Hierarchy: '`<hier-`

path>']

Where, *<hier-path>* is the complete hierarchical name of the containing scope excluding subprograms.

Message 2

The following message is displayed when the signal size is greater than 50,000 and the [handle_large_bus](#) parameter is disabled:

```
signal '<signal-name>' size too big thus not processed, use  
'configure_lint_tag_parameter -tag W240 -parameter  
HANDLE_LARGE_BUS -value yes' for enabling handling of these  
signals
```

If the HANDLE_LARGE_BUS parameter is not enabled, the violation for signals of size greater than 50,000 is missed.

VHDL

The following message appears at the location where an input port *<port-name>* is declared that is never read in the process. For multi-bit ports the Tag also displays the offending bits in the message.

Not all the elements of in port '*<port-name>*' '*<offending-bits>*' are read [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the complete hierarchical path of the containing process.

Tag Severity

Warning

Suggested Fix

In most cases, you should remove the input. This will force a change in design units instantiating this module. In some special cases, for example scan-enable, an unconnected input appears as a place holder for later logic insertion. These cases should be waived.

Examples

VHDL

Consider the following example:

Lint Tags

```

library ieee;
use ieee.std_logic_1164.all
entity test is
    port(a: in bit_vector(0 to 5) ;
         b: in bit;
         c: in bit;
         z: out bit
        );
end test;

architecture behav of test is
begin
    process(a, b)
    begin
        z <= a(3) or b;
    end process;
end behav;

```

For this example, VC SpyGlass Lint generates the following messages:

Not all the elements of in port 'a' (Bits: 0:2 4:5) are read
[Hierarchy: 'test(behav):']

Not all the elements of in port 'c' are read [Hierarchy:
'test(behav):']

The first message indicates that the bits of vector a at the 0, 1, 2, 4, and 5 position numbers of the port were not read and the second message indicates that none of the bits of vector c were read.

Verilog

Consider the following example:

```

module ffd (z,a,b,c);
    input b,c;
    output reg z;
    input [5:0] a;
    always @(a,b)
        z<=a[3] | b;
endmodule

```

In the above example, the *W240* Tag reports a violation because the input

`c` is never read in the module. The following message is reported by this example:

Input 'c' declared but not read.[Hierarchy: ':ffd']

W241

Output is never set

Language

Verilog, VHDL

Description

The W241 Tag flags output ports that are not set (completely set for VHDL design) in the module.

While such ports (for example a Qbar output from a flip-flop) are allowed, un-set outputs can confuse other users.

NOTE: *For Verilog, inside the nested `for` loops, signals of user-defined type like structures, interfaces, etc. are considered fully set if used on the left-hand side of an expression and fully read if used on the right-hand side of an expression.*

By default, this Tag does not process large arrays. Set the [handle_large_bus](#) parameter to `yes` to process large arrays (greater than 50,000) and report violation if not used correctly.

To see the details of the output that is not set, refer to the [SignalUsageReport](#) report in the `VC SpyGlass Lint_reports/lint` directory.

By default, all Tag related data is reported in the [SignalUsageReport](#) report. Set the value of the [disable_signal_usage_report](#) parameter to Tag-name to disable data generation in the [SignalUsageReport](#) report for the specified Tag.

Message Details

Verilog

Message 1

The following message appears at the location where an output port `<port-name>` is declared that is not set in the module:

Output '`<port-name>`' is never set. [Hierarchy: '`<hier-path>`']

The `<hier-path>` is the complete hierarchical name of the containing scope excluding subprograms.

Message 2

The following message is displayed when the signal size is greater than 50,000 and the [handle_large_bus](#) parameter is disabled:

```
signal '<signal-name>' size too big thus not processed, use  
'configure_lint_tag_parameter -tag w241 -parameter  
HANDLE_LARGE_BUS -value yes' for enabling handling of these  
signals
```

If the `HANDLE_LARGE_BUS` parameter is not enabled, the violation for signals of size greater than 50,000 is missed.

VHDL

The following message appears at the location where an output port `<port-name>` is declared that is not completely set in the process:

```
not all the elements of out port '<port-name>' '<offending-  
bits>' are set [Hierarchy: '<hier-path>']
```

Where, `<offending-bits>` refers to the offending bits for multi-bit ports.

The `<hier-path>` is the complete hierarchical name of the containing scope excluding subprograms.

Tag Severity

Warning

Suggested Fix

In most cases, you should remove the output. This will force a change in design units that are instantiating this module. In some special cases, an undriven output appears as a place holder for later logic insertion. These cases should be waived.

Examples

Verilog

Consider the following example:

```
module test1(out1,out2,in1,in2 );  
    input  [13:0] in1;  
    input  [13:0] in2;
```


Lint Tags

```

output [15:0] out1;
output out2;

assign out1[13:0]= in1 | in2;

```

```
endmodule
```

In the above example, the *W241* Tag reports two violations because the output *out2* and *out1[15:14]* are never set in the module. The following messages are reported by this example:

Output 'out1[15:14]' is never set.[Hierarchy: ':test1']

Output 'out2' is never set.[Hierarchy: ':test1']

VHDL

Consider the following example:

```

library ieee;
use ieee.std_logic_1164.all
entity test1 is
    port( in1: in bit_vector(13 downto 0);
          in2: in bit_vector(13 downto 0);
          out1: out bit_vector(15 downto 0);
          out2: out bit);
end test1;

architecture behav of test1 is
begin
    process(in1, in2)
    begin
        out1(13 downto 0) <= in1 or in2;
    end process;
end behav;

```

In the above example, the *W241* Tag reports two violations because the output *out2* and *out1* (bits: 15:14) are never set in the architecture behave of entity test1 . The following messages are reported by this example:

Not all the elements of out port 'out1' (Bits: 15:14) are set.

[Hierarchy: ':test1(behav):']

Not all the elements of out port 'out2' are set. [Hierarchy:
':test1(behav):']

W243

Recursive task enable

Language

Verilog

Description

The W243 Tag flags the task statement block, which has a call to task with the same name.

The W243 Tag also reports a message, if a task call is recursive through more than one task. For example, if 'task_a' calls task 'task_b' and 'task_b' calls 'task_a' then this will be reported as recursive call.

Message Details

The following message appears when a task block having calls to task(s) *<task-name-list>* with the same name or having recursive task calls is encountered:

Recursive task calls in the design:Order of task calls: *<task-name-list>*

Where *<task-name-list>* is the list of tasks, separated by dot, in the order of invocation.

Severity

Warning

Suggested Fix

In case of recursive task enable, there is always a chance that two tasks call each other infinitely. Avoid such situation by using loop constructs or some alternate method of writing the code.

Examples

Consider the following example where a task t1 calls task t1:

```
module taskloop (in1, in2, out1);  
    input  in1, in2;
```

```
output out1;
input in1, in2;
output out1;
reg out1;

always @(in1 or in2)
begin
    t1 (out1, in1, in2);
end

task t1;
output o2;
input i1;
input i2;

begin
    t1 ( o2, i1 , i2 );
end
endtask
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Recursive task calls in the design:Order of task calls: t1.t1

W245

Probably intended "or", not "|" or "||" in sensitivity list

Language

Verilog

Description

The W245 Tag flags bit-wise `or` operator (`|`) or logical `or` operator (`||`) used in the sensitivity list of an `always` construct.

As the sensitivity list of an `always` construct should be sensitive to the events in the control signals, it is recommended to use the `or` operator.

The expression `@(a || b)` is generally not equivalent to `@(a or b)`. The first expression evaluates to the value `a || b` and triggers the event when that value changes. This is generally not the design intent. The second expression triggers when either signal `a` or signal `b` changes which is generally the design intent.

NOTE: The W245 Tag supports *generate-if*, *generate-for*, and *generate-case* blocks.

Message Details

The following messages appear at the location of the sensitivity list of an `always` construct where the bit-wise `or` operator (`|`) is used:

Probably intended "or" instead of "|" in sensitivity list

The following message appears at the location of the sensitivity list of an `always` construct where the logical `or` operator (`||`) is used:

Probably intended "or" instead of "||" in sensitivity list

Tag Severity

Warning

Suggested Fix

If you meant the first case, precompute `a || b`, then use that value in the sensitivity list. Only use `or` in the sensitivity list.

Examples

Consider the following example:

```
always @( clk )  
begin  
...  
end
```

```
always @( clk | clk1)  
begin  
...  
end
```

In the above snippet, the *W245* Tag reports a violation because the bit-wise or operator (`|`) or logical or operator (`||`) is used in the sensitivity list of the `always` construct that is highlighted in red.

Probably intended "or" instead of "||" in sensitivity list

W250

Reports disable statements that are not synthesizable

When to Use

Use this tag to identify `disable` statements used in the design.

Description

The *W250* tag reports the `disable` statements.

Language

Verilog

Default Weight

5

Parameter(s)

avoid_synth_disable: Default value is `no`. Set this parameter to `yes`, to enable the tag to not report violations for synthesizable `disable` constructs for the following cases:

- Tasks
- Blocks
 - ☐ Sequential blocks (whose start-end is defined by `begin-end`)

For Parallel blocks (whose start-end is defined by `fork-join`), the tag does not waive violations as they are considered as non-synthesizable.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a `disable` statement is encountered.

[WARNING] Disable statement may not be synthesizable

Potential Issues

A violation is reported when a `disable` statement is used in the design.

Consequences of Not Fixing

The `disable` statements are useful in behavioral modeling but have no physical counterpart because they represent an arbitrary jump out of loop. Therefore, such descriptions are not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the offending `disable` statement is used. This is un-synthesizable by several commercially available synthesis tools.

No fix required if this is in a simulation mode or testbench. If this is intended to be synthesizable, you should rewrite the code to avoid the `disable` statement.

Example Code and/or Schematic

Consider the following example:

```
module case_250 (clk1,in,cntr,out);

    input clk1,in;
    input [1:0] cntr;
    output out;

    initial begin
        f_190;
    end

    initial begin
        #10 disable f_190;
    end

    task f_190;
    reg clk, f_191;
    reg[1:0] cntr_f;
        if ( clk == 1'b0) f_191 = cntr_f[1];
        else f_191 = cntr_f[0];
    endtask

endmodule
```

Lint Tags

In the above example, the Tag reports a violation as a disable statement is used in the design.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

W253

Data event has an edge

Language

Verilog

Description

The W253 Tag flags data event variables that have been specified with an edge in a timing check system task call.

For example, the `$setup` task data variable should not be an edge.

Message Details

The following message appears at the location where the data event variable `<var-name>` is used with an edge specification in a timing check system task call:

Timing check data event: '`<var-name>`' should not use edge specification

Tag Severity

Warning

Suggested Fix

Replace the edge specification with a legal data variable.

Example

The following example of timing check system task call violates the W253 Tag as the data variable (`out1`) has been specified with an edge specification:

```
$setup(posedge out1, posedge clk, 5, notifier);
```

W254

Reference event does not have an edge

Language

Verilog

Description

The W254 Tag flags reference event variables that have been specified without an edge in a timing check system task call.

For example, the `$setup` task reference variable must be an edge.

Message Details

The following message appears at the location where the reference event variable `<var-name>` is used without an edge specification in a timing check system task call:

Timing check reference event: '`<var-name>`' should use edge specification

Tag Severity

Warning

Suggested Fix

Replace the reference specification with a legal edge variable.

Examples

The following example of timing check system task call violates the W254 Tag as the reference event (`clk`) does not use an edge specification:

```
$setup(out2, clk, 2, notifier);
```

W256

A notifier must be a one-bit register

Language

Verilog

Description

The W256 Tag flags notifiers that are not single-bit registers.

A notifier is the last argument in a timing check system task (\$setup, \$hold, etc.) and must be a single-bit register.

NOTE: *By default, the W256 Tag is switched off. The functionality of this Tag will now be covered by STX_VE_1366 Tag. For more details on the STX_VE_1366 Tag, refer to VC SpyGlass Lint® Built-In Tags Reference Guide.*

Message Details

The following message appears at the location where a notifier `<noti-name>` that is not a single-bit register, is used:

```
notifier '<noti-name>' must be a single-bit register
```

Tag Severity

Warning

Suggested Fix

Replace the notifier specification with a legal single-bit register.

Examples

The following example of timing check system task call violates the W256 Tag as the notifier is not a single-bit register:

```
...  
wire [1:0]notifier = 2'b10;  
...  
$setup(out2, clk, 2, notifier);
```

W257

Synthesis tools ignore delays

Language

Verilog, VHDL

Description

The W257 Tag flags delay/after clauses in the design.

For Verilog designs, the W257 Tag reports a maximum of 500 Tag messages per design and a maximum of 20 Tag messages per module. Set the [allviol](#) Tag parameter to `yes` or `<Tag-name>` to report all violation messages.

As delays are ignored by synthesis tools, the pre- and post-synthesis simulations of designs with delays may not match.

Message Details

Verilog

The following message appears at the location where a `delay` statement is encountered:

Delays will be ignored in synthesis

VHDL

The following message appears at the location where an `after` clause is encountered:

AFTER clause will be ignored in synthesis

Severity

Warning

Suggested Fix

If possible, avoid using delays. Test the detailed timing in post-synthesis timing analysis, and not in simulation. Where essential to avoid race conditions, use only unit delays. Restrict the use of non-unit delays to testbenches only.

W259

Signal has multiple drivers

Language

VHDL

Description

The W259 Tag flags signals that have multiple drivers but no associated resolution function.

A signal is to be said to be driven by multiple drivers, if more than one concurrent statements (process, instance, conditional, and selected signal assignment statements etc.) contain signal transform for the same signal.

By default, only simple types like `std_logic` and `std_logic` vector have defined resolution functions. Signals with multiple drivers but no resolution function are driven to tristate or to value x.

The W259 Tag currently does not handle multi-dimensional arrays, aggregate types, and record types.

NOTE: The W259 Tag supports *generate-if* and *generate-for* blocks.

NOTE: The [W415](#) Tag flags those data type that have built-in resolution function (for example, `std_logic`). However, if the data type has no built-in resolution function, then the W259 Tag is applicable to check for multiple drivers.

Message Details

The following message appears at the location where a signal `<sig-name>` with multiple drivers but no resolution function is used:

Non-resolved signal '`<sig-name>`' has multiple drivers
[Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path.

Tag Severity

Warning

Suggested Fix

Determine first if you really intended a WIRE-OR. If so, provide a

resolution function to resolve the connection.

W263

Reports a case expression width that does not match case select expression width

When to Use

A violation is reported when a case expression width does not match the select expression width.

Description

The *W263* Tag reports the case clause labels whose widths do not match the width of the corresponding case construct selector.

This Tag calculates the width of case expressions on the basis of the following conditions:

- If the [nocheckoverflow](#) Tag parameter is set to `yes` or *W263*, width is calculated as per LRM. However, for constants, natural width is considered.
- If the [nocheckoverflow](#) Tag parameter is set to `no`, width is calculated according to the following methods:
 - ? For plus/minus operator, value-based width is considered.
 - ? For multiplication operator, width is sum of operand widths.
 - ? For division operator, LHS width is considered.
 - ? For concat operator, width is calculated as sum of all the operands.

Tag Exceptions

The *W263* Tag does not report a violation in the following scenarios:

- When case select is a static expression
- When case expression is a constant (integer/macro) or unsized based number, and its width is smaller than the case select width

Language

Verilog

Default Weight

5

Parameter(s)

- ***IGNORE_WITHIN_RANGE_LABELS***: Default value is `no`. Set the value of the parameter to `yes` to ignore the violation, if case label value lies within the range of case select value.
- ***nocheckoverflow***: Default value is `no`. This indicates the *W263* Tag does not check the bit-width as per LRM. Set this parameter to `yes` or Tag name to check the bit-width as per LRM.
- ***strict***: The *W263* Tag is switched off by default. You can enable this Tag by specifying the *W263* command.
- ***use_lrm_width***: Default value is `no` and the *W263* Tag considers the natural width of integer constants and unsized based numbers. Set this parameter to `yes` to consider LRM width of integer constants and unsized based numbers, which is 32.
- ***handle_static_caselabels***: Default value is `no`. Set this parameter to `yes` to ignore violations for static case labels, which are of less width than the width of case selector.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location of a case clause label whose width (`<case-label-width>`) is not same as the case construct selector width (`<selector-width>`).

[WARNING] Case label (`<case-label-name>`) width (`<case-label-width>`) does not match selector (`<selector-name>`) width (`<selector-width>`) [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical name of the containing scope excluding subprograms.

Potential Issues

Violation may arise when the case label width does not match the selector width.

Consequences of Not Fixing

When the case clause labels are of width smaller than the width of the case

construct selector, it might lead to unwanted target in case expressions.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line of case label. Scroll up the HDL Viewer window to view the case selector and the width mismatch. Alternatively, open the HDL in editor and search for case selector.

Modify the case expressions to explicitly match the case selector in all bits. This makes the behavior of the RTL code easier to understand and does not rely on undefined behavior in handling unmatched bits.

Example Code and/or Schematic

Example 1

Consider the following example where the `sel` clause of the case construct is of width 4 bits and the case clause labels have the same or different bit-widths.

```
module test(in, out, sel, data);
    input [3:0] in, data, sel;
    output [3:0]out;
    reg [3:0]out;

    parameter [3:0]S = 23;
    parameter [2:0]P = 10;
    parameter [15:0]Q = 16'h1101;
    parameter R = 2;

    always @(in or sel)
    begin
        case (sel)
            S : out = 4'b0000;
            P : out = 4'b0000;
            Q : out = 4'b0001;
            R : out = 4'b0011;
            in: out = 4'b0011;
            4'b1010 : out = 4'b0011;
            3'b110 : out = 4'b0011;
            default out = 0;
        endcase
    end
endmodule
```

```

        endcase
    end
endmodule

```

The case clause labels (S, in, and 4'b1010) have the same width (4 bits) as the case construct selector sel.

The case clause labels P (3 bits), Q (16 bits), R (32 bits being an integer), and 3'b110 (3 bits) do not have the same bit-width as the case construct selector sel.

Example 2

Consider the following example:

```

module mod(sel,o1,o2,o3,o4,clk);
input clk;
input [1:0] sel;
output [3:0] o1, o2, o3, o4;
reg [3:0] o1, o2, o3, o4;
reg [3:0] mem[0:3];

always@(sel)
begin
    case(sel)
        2'b00 : o1 = mem[0];
        2'b01 : o2 = mem[1] ;
        1'b1  : o4 = mem[3];
        default : o4 = mem[2];
    endcase
end

```

```
endmodule
```

In the above example, the W263 Tag reports a violation as the case label width does not match the selector width.

Example 3

Consider the following example:

```

case (sel)
(a[2:0]+b[2:0]): out=1'b1;

```

In the above example, width of `a[2:0] + b[2:0]` is 4.

Example 4

Consider the following example:

```
case (sel)
(a[2:0]*b[2:0]): out=1'b1;
(a1[2:0]*b1[2]): out=1'b0;
```

In the above example, the `a[2:0]*b[2:0]` is 6. Also, the width of `a1[2:0]*b1[2]` is 3.

Example 5

Consider the following example:

```
case (sel)
(a[2:0]/b[2]): out=1'b0;
```

In the above example, width of `a[2:0]/b[2]` is 3.

Example 6

Consider the following example:

```
case(sel)
{1'b0,a[2:0]}: out=1'b1;
{1'b1,a[2:0]}: out=1'b0;
```

In the above example, width is 3 for the first expression, ignoring all the leading 0s. The width is 4 bits for the second expression.

Example 7

Consider the following example:

```
module top(input clk);
wire      [7:0] pState;
wire      in1,in2;
reg       out;

parameter  ASTATE = 4;
parameter  BSTATE = 512;

    always @ (posedge clk) begin
        case (pState)
```

Lint Tags

```

        ASTATE : out = in1;
        BSTATE : out = in2;
    endcase
end
endmodule

```

In the above example, the W263 Tag does not report a violation when the case label width is less than the selector width, since the case labels ASTATE and BSTATE are parameters.

Example 8

Consider the following example:

```

module test(in, out, sel, data);
input [2:0] in, data, sel;
output [3:0]out;
reg [3:0]out;
always @(in or sel)
begin
case (sel)
4'd3 : out = 4'b0011;
3'b110 : out = 4'b0011;
default out = 0;
endcase
end
endmodule

```

In the above example, value of `sel` will be from 0 to 7. Therefore, case label value lies within this range and no violation is reported under . Hence no violation should be flagged here under the [IGNORE_WITHIN_RANGE_LABELS](#) parameter.

Default Severity Label

Warning

Tag Group

Lint_Elab_Tags, Case

Reports and Related Files

None

W280

A delay has been specified in a nonblocking assignment

Language

Verilog

Description

The W280 Tag flags intra-assignment delays specified with nonblocking assignments.

Such description is unlikely to correspond to the physical implementation. However, such description may be required to use a unit delay to avoid race conditions in simulation.

Parameters

[*report_inter_nba*](#): Default value is `no`. In this case, the W280 Tag does not report inter-nonblocking assignment delay. Set the value of the parameter to `yes` to report such cases.

Message Details

The following message appears at the location where a nonblocking assignment is used to describe an intra-assignment delay:

Intra-assignment delay used in a nonblocking assignment

Tag Severity

Warning

Examples

Consider the following example, where an intra-assignment delay in a nonblocking assignment mimics a gate delay:

```
a <= #10 b;
```

W287a

Some inputs to instance are not driven or unconnected

Language

Verilog, VHDL

Description

The W287a Tag flags module or gate instances where nets connected to input ports are not driven and the instance input which are not connected.

The W287a Tag (Verilog) also checks pins/ports driven by inout pins/ports.

For Verilog designs, the W287a Tag does not flag instance input ports where the connected net is coming out of a black box instance unless the [strict](#) Tag parameter is set.

It is an error in CMOS to let an input float, unless that input has a pull-up or pull-down.

In case, a bit-select or a part-select of a multi-bit signal is connected to an input of a module or gate instance, the W287a Tag requires that all connected bits must be driven and other bits need not be driven.

NOTE: For VHDL, you can enable the W287a Tag by specifying the `configure_lint_tag -enable -tag W287a` command. However, this Tag will not run, if you set the [fast](#) Tag parameter to `yes` and VC SpyGlass Lint product is run.

By default, the W287a tag reports multiple violations when signal name, instance and hierarchy are the same. Set the value of the [report_single_violation](#) parameter to `yes` to report one violation when signal name, instance and hierarchy are the same.

By default, the tag reports violations for multiple hierarchies. Set the value of the [report_only_from_one_hierarchy](#) parameter to `yes` to suppress the hierarchy field and report only one violation per assignment.

By default, the tag reports violation for port connections that have been explicitly left open in the RTL. Use the [filter_mark_open](#) parameter to not report violations on those port connections that have been explicitly left open in the RTL.

Message Details

Verilog

The following message appears at the location of a module instance where a signal *<sig-name>* of instance *<instance-name>* connected to an input port is never driven:

[WARNING] Input '*<sig-name>*' of instance '*<instance-name>*' is undriven. [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the complete hierarchical name of the containing scope excluding subprograms.

Similarly, the following message appears where a signal *<sig-name>* of instance *<instance-name>* is not connected to an input port:

[WARNING] Input '*<sig-name>*' of instance '*<instance-name>*' is unconnected. [Hierarchy: '*<hier-path>*']

VHDL

The following message appears at the location of a component instantiation where bits *<bit-list>* of signal *<sig-name>* of instance *<instance-name>* connected to an input port are never driven:

[WARNING] Input Signal '*<sig-name>*' of instance '*<instance-name>*' not driven. [Elaborated Module Name: *<module-name>*]

Where, *<module-name>* is the module name.

Tag Severity

Warning

Suggested Fix

If you don't care about the input value, tie it high or low, but do not let it float.

Examples

Consider the following example where wire w2 connected to second input port of instance a1 of AND gate cell is not driven:

```
module test(in1, in2, clk, out1);
```

```
input in1, in2, clk;
output out1;

wire w1, w2;

assign w1 = in1 & in2;

and a1(out1, w1, w2);
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Undriven instance input 'w2'. [Hierarchy: ':test']

W287b

Output port to an instance is not connected

Language

Verilog, VHDL

Description

The W287b flags module or gate instances where the output ports are not connected.

While such descriptions are allowed, they are generally design mistakes.

By default, the tag reports violation for port connections that have been explicitly left open in the RTL. Use the [filter_mark_open](#) parameter to not report violations on those port connections that have been explicitly left open in the RTL.

Message Details

Verilog

The following message appears at the location of a module instance where the output port *<port-name>* is not connected:

Instance output port '*<port-name>*' is not connected

VHDL

The following message appears at the location of a component instantiation where the signal *<sig-name>* connected to an output port of the component is never used in the design:

Instance output '*<sig-name>*' not used. [Elaborated Module Name: *<module-name>*]

Where, *<module-name>* is the name of the module.

Tag Severity

Warning

Suggested Fix

Check carefully to make sure that you intended to ignore this output.

W287c

Inout port of an instance is not connected or connected net is hanging

Language

Verilog, VHDL

Description

The W287c flags module or gate instances where inout ports are not connected or connected net is not used anywhere in the module.

While such descriptions are allowed, they are generally design mistakes. When in the input mode, an unconnected inout port is equivalent to an undriven input which is a design error in CMOS unless it is a pull-up or pull-down.

Message Details

Verilog

The following message appears at the location of a module instance where the inout port, *<port-name>*, is not connected:

Unconnected instance inout '*<port-name>*' [Hierarchy: '*<hier-path>*']

The following message appears at the location of the module instance where the net connected to the port *<port-name>* is hanging:

Net connected to instance inout '*<port-name>*' is hanging
[Hierarchy: '*<hier-path>*']

VHDL

The following message appears at the location of component instantiation where port, *<port-name>*, is not connected:

Unconnected instance inout '*<port-name>*' [Elaborated Module Name: *<module-name>*]

The following message appears at the location of component instantiation where a net connected to the port, *<port-name>*, is hanging:

Net connected to instance inout '*<port-name>*' is hanging
[Elaborated Module Name: *<module-name>*]

where, *<module-name>* is the name of the module.

Tag Severity

Warning

Suggested Fix

Tie the input high or low if it will not be used.

W289

Reports real operands that are used in logical comparisons

When to Use

Use this Tag to identify cases where a real operand is used in a logical comparison.

Description

The W289 Tag reports real operands used in logical comparisons.

NOTE: *The W289 Tag supports generate-if, generate-for, and generate-case blocks.*

Language

Verilog

Parameters

None

Constraints

None

Messages and Suggested Fix

The following message appears at the location where a real operand `<oprnd-name>` is used with a logical comparison operator `<op-name>`.

[WARNING] A real operand: '`<oprnd-name>`' should not be used with logical comparison operator '`<op-name>`'

Potential Issues

Violation may arise when a real operand is used with a logical comparison operator.

Consequences of Not Fixing

It is unlikely that conditions, such as `RealValue == 1`, would ever be true, given rounding errors in floating point values.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the logical comparison statement where a real variable is used as one of the

operand.

You should allow some margin for rounding. For example,
`RealValue > 0.99 && RealValue < 1.01`

Example Code and/or Schematic

Consider the following example:

```
module mod(i1, i2, clk, res, o1);
  input  i1, i2;
  input  clk, res;
  output o1;
  reg o1;

  real a, b;

  always @(posedge clk or negedge res)
    if (a!=b)
      o1 <= a + b;
    else
      o1 = a-b;
endmodule
```

In the above example, the *W289* Tag reports a violation as the real operand, *a*, is used with logical comparison operator, '!='.

Default Severity Label

Warning

Tag Group

Expression

Reports and Related Files

None

W292

Reports the comparison of real operands

When to Use

Use this Tag to identify logical comparison operations on real type operands.

Description

The W292 Tag reports logical comparison operations on real type operands.

Language

VHDL

Parameters

None

Constraints

None

Messages and Suggested Fix

The following message appears at the location where a logical comparison of real operands is encountered.

[WARNING] Logical Comparison of Real operands is not recommended

Potential Issues

Violation may arise when real operands are used in logical comparisons.

Consequences of Not Fixing

It is unlikely that conditions, such as `RealValue == 1`, would ever be true, given rounding errors in floating point values.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where real signals are used in comparison.

You should allow some margin for rounding. For example, `RealValue >`

0.99 and RealValue < 1.01

Example Code and/or Schematic

Consider the following example:

```
entity test1 is
  port(
    in1, in2 : in bit;
    out1, out2 : out bit
  );
end test1;

architecture behav of test1 is
  signal b1, b2, b3: real;
  signal sig1: bit;
begin
  process(in1)
  begin
    if(b1 = 1.0) then
      out1 <= in1;
    else
      out1 <= not in1;
    end if;
  end process;
  process(in2)
  begin
    if(b2 = b3) then
      out2 <= in2;
    else
      out2 <= not in2;
    end if;
  end process;
end behav;
```

In the above example, the W292 Tag reports a violation as the logical comparison of real operands is not recommended.

Lint Tags

Default Severity Label

Warning

Tag Group

Expression

Reports and Related Files

None

W293

Reports functions that return real values

When to Use

Use this Tag to identify functions that return `real` values.

Description

The *W293* Tag reports functions that return real values.

Language

Verilog, VHDL

Parameters

None

Constraints

None

Messages and Suggested Fix

Verilog

The following message appears at the start of a function description that returns a real value.

[WARNING] Function returns a real value which is not synthesizable

Potential Issues

Violation may arise when a function returns a real value.

Consequences of Not Fixing

Objects with real values have no physical equivalent and therefore are not synthesizable.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line of function definition, which returns a real value.

VHDL

The following message appears at the start of a subprogram
<subp-name> description that returns a real value:

[WARNING] The <subp-name> returns a real value which may not be synthesizable

Where <subp-name> can be function or procedure.

Potential Issues

Violation may arise when a subprogram returns a real value.

Consequences of Not Fixing

Objects with real values have no physical equivalent and therefore are not synthesizable.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line of function definition, which returns a real value.

No fix required if this occurs in a testbench. If this is targeted for synthesizable RTL, you will need to explicitly implement floating point logic.

Example Code and/or Schematic

Consider the following example:

```
entity ent is
  port (
    in1 : in real;
    out1: out real
  );
end ent;

architecture arc of ent is
  function myFunc (number: real) return real is
  begin
    return number * 3.142857;
  end myFunc;
begin
  process
  begin
```

```
    out1 <= myFunc(in1);  
end process;  
end arc;
```

In the above example, the *W293* Tag reports a violation as the function, `myFunc`, returns a real value, which is unsynthesizable.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

W294

Reports real variables that are unsynthesizable

When to Use

Use this Tag to identify real variables, which are unsynthesizable.

Description

The W294 Tag reports `real` variables used in the design.

NOTE: *This tag does not support `parameter` and `localparam`. It is recommended to use [RealValueUsed-ML](#) tag.*

Language

Verilog

Parameters

None

Constraints

None

Messages and Suggested Fix

The following message appears at the location where a variable `<var-name>` of type `real` is encountered.

[WARNING] Real variable '`<var-name>`' is not synthesizable

Potential Issues

Violation may arise when a design has a real variable.

Consequences of Not Fixing

Objects with real values have no physical equivalent and therefore may not be synthesizable by some synthesis tools. Real variables should only be used in testbenches and simulation models.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the offending real variable is declared. This is un-synthesizable by several commercially available synthesis tools.

If targeting synthesizable logic, use integer or reg variables to fix the violation.

Example Code and/or Schematic

Consider the following example:

```
module test(in1, in2, z);  
  input in1;  
  input in2;  
  output z;  
  real r = 0.025;    //VIOLATION  
  assign z = r + in1 + in2;  
endmodule
```

In the above example, the W294 Tag reports a violation as a real variable, *r*, is used in the design.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

None

W295

Reports event variables that are not synthesizable

When to Use

Use this Tag to identify event variables used in the design.

Description

The W295 Tag reports event variables.

Language

Verilog

Parameters

None

Constraints

None

Messages and Suggested Fix

The following message appears at the location where an event variable is declared.

[ERROR] Event variable may not be synthesizable

Potential Issues

Violation may arise when an event variable is encountered in a design.

Consequences of Not Fixing

Event variables have no physical equivalent and therefore are not be synthesizable by some synthesis tools. However, event constructs may appear in testbenches or system-level models.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line of event signal declaration.

If you intend to make the code synthesizable, trigger on signals.

Example Code and/or Schematic

Example of Event Variable Declared and Used

In the following example, an event variable `test` is declared and used.

```
module test (clk1, in, cntr, out);  
    input clk1, in;  
    input [1:0] cntr;  
    output out;  
  
    event test;  
    reg a_1, a_2;  
  
    always @(clk1 or test)  
        a_2 = in;  
  
    always @(test)  
        a_1 <= in;  
  
    assign out = a_1 + a_2;  
endmodule
```

Example of Event Variable Declared But Not Used

In the following example, an event variable `in2` is declared but is not used.

```
module test(in1, out1);  
    input in1;  
    event in2;  
    output out1;  
  
    reg out1;  
  
    always @(in1)  
    begin  
        #10 out1 = in1;  
    end  
endmodule
```

Example of Event Variable Declared and Triggered

In the following example, an event variable `in2` is declared and triggered.

```
module test(in1, out1);  
    input in1;  
    event in2;  
    output out1;  
  
    reg out1;  
  
    always @(in1)  
    begin  
        #10 out1 = in1;  
        ->in2;  
    end  
endmodule
```

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

None

W306

Converting integer to real

Language

Verilog

Description

The W306 Tag flags `integer` type to `real` type conversions.

Type conversion may indicate an unintended error. Such type of conversions may lead to loss in accuracy.

Message Details

The following message appears at the location where an `integer` type is being converted to a `real` type:

```
Converting integer to real
```

Tag Severity

Warning

Suggested Fix

Check to see if the conversion was intended. If yes, waive messages on those cases.

W307

Converting unsigned (reg type) to real

Language

Verilog

Description

The W307 Tag flags unsigned type (`reg` type) to `real` type conversions. Any type conversion may indicate an unintended error. Such type of conversions may lead to loss in accuracy. Since a `reg` is effectively an unsigned integer, you may unexpectedly get a negative integer where you expected a positive value.

Message Details

The following message appears at the location where a `reg` type is being converted to a `real` type:
Converting reg to real

Tag Severity

Warning

Suggested Fix

Check to see if the conversion was intended. If yes, waive messages on those cases.

W308

Converting real to integer

Language

Verilog

Description

The W308 Tag flags `real` type to `integer` type conversions.

Any type conversion may indicate an unintended error. In this case there may be a loss of significant data (the fractional part of the float).

Message Details

The following message appears at the location where a `real` type is being converted to an `integer` type:

Converting real to integer

Tag Severity

Warning

Suggested Fix

Check to see if the conversion was intended. If yes, waive messages on those cases.

W309

Converting unsigned (reg type) to integer

Language

Verilog

Description

The W309 Tag flags unsigned type (`reg` type) to `integer` type conversions.

Type conversion may indicate an unintended error. Since a `reg` is effectively an unsigned integer, you may unexpectedly get a negative integer where you expected a positive value. There could also be a loss of significant data if the `reg` type is wider than the `integer` type (32 bits).

Message Details

The following message appears at the location where a `reg` type is being converted to an `integer` type:

Converting reg to integer

Tag Severity

Warning

Suggested Fix

Check to see if the conversion was intended. If yes, waive messages on those cases.

W310

Converting integer to unsigned (reg type)

Language

Verilog

Description

The W310 Tag flags `integer` type to unsigned type (`reg` type) conversions.

Type conversion may indicate an unintended error. Since a `reg` is effectively an unsigned integer, what may have been a signed value will become unsigned. This would in turn affect the outcome of arithmetic operations (For example, a negative number becomes bigger than a positive number).

Message Details

The following message appears at the location where an `integer` type is being converted to an unsigned type (`reg` type):

Converting integer to reg

Tag Severity

Warning

Suggested Fix

Check to see if the conversion was intended. If yes, waive messages on those cases.

Example

```
module test(out, in1, in2);  
  input in1, in2;  
  output out;  
  reg out;
```


Lint Tags

```
reg data;
integer int_val1;
reg reg1;
reg [15:0] reg2;
reg [31:0] reg3;
reg [39:0] reg4;

always @(in1 or in2)
begin
  reg1 <= int_val1; // W310 violation
  reg2 <= int_val1; // W310 violation
  reg3 <= int_val1; // W310 violation
  reg4 <= int_val1; // W310 violation
end
endmodule
```

W311

Converting real to unsigned (reg type)

Language

Verilog

Description

The W311 Tag flags `real` type to unsigned type (`reg` type) conversions. Type conversion may indicate an unintended error. In this case, there may be a loss of significant data (the fractional part of the float).

Message Details

The following message appears at the location where a `real` type is being converted to an unsigned type (`reg` type):

```
Converting real to reg
```

Tag Severity

Warning

Suggested Fix

Check to see if the conversion was intended. If yes, waive messages on those cases.

W312

Converting real to single bit

Language

Verilog

Description

The W312 Tag flags violation when a `real` type node is assigned to single-bit node.

For example, consider the following scenario:

```
reg data;  
real realval;  
data = realval;
```

Any type conversion may indicate an unintended error, which may cause loss of significant data.

Message Details

The following message appears at the location where a `real` type is being converted in to a single-bit type:

Converting real to single bit

Tag Severity

Warning

Suggested Fix

Check to see whether the conversion was intended. Waive violations on such cases.

W313

Converting integer to single bit

Language

Verilog

Description

The W313 Tag flags violation on assignment statements where an `integer` type node is assigned to a single-bit node.

For example, consider the following scenario:

```
reg data;  
integer intval;  
data = intval; //violation
```

Any type conversion may indicate an unintended error. In this case there may be a loss of significant data.

Message Details

The following message appears at the location where an `integer` type is being converted in to a single-bit type:

Converting integer to single bit

Tag Severity

Warning

Suggested Fix

Check to see whether the conversion was intended. Waive violations on such cases.

W314

Converting multi-bit reg type to single bit

Language

Verilog

Description

The W314 Tag flags violation when multi-bit `reg` type node is assigned to a single-bit. For example, consider the following scenario:

```
output out1;  
reg [3:0] count;  
out1 = count
```

Any type conversion may indicate an unintended error. In this case, there may be loss of significant data.

By default, the W314 Tag reports only one violation for every violating line, even if a module is instantiated more than once with different parameter values. Set the value of the [allviol](#) parameter to `yes` or `<Tag-name>` to report different violations for every violating line resulting from a module, which is instantiated more than once with different parameter values.

NOTE: The W314 Tag supports *generate-if*, *generate-for*, and *generate-case* blocks.

Message Details

The following message appears at the location where a multi-bit `reg` type is being converted in to a single-bit type:

Converting multi-bit reg type to single bit [Hierarchy: '<hier-path>']

Where, `<hier-path>` is the complete hierarchical path.

Tag Severity

Warning

Suggested Fix

Check to see whether the conversion was intended. Waive violations on such cases.

W316

Reports extension of extra bits in integer conversion

When to Use

Use this Tag to identify extension of extra bits in integer conversion.

Description

The *W316* Tag reports integer conversions where the left expression is wider than the right expression. When assigning a value to an integer variable, if the width of the value is less than the width of an integer (32 bits), the value is extended. Constant expressions are also checked.

This Tag reports the following types of assignments assuming that *n* is an integer variable, *basedNum* is a constant, and *m* and *k* are variables. All of these have bit-widths less than 32 bits:

```
n = basedNum;  
n = basedNum + basedNum;  
n = basedNum + m;  
n = m;  
n = m + k;
```

When an integer variable (left expression) is assigned a value (right expression), such value should be an integer or integer equivalent (that is, 32 bits).

NOTE: The *W316* Tag supports *generate-if*, *generate-for*, and *generate-case* blocks.

Language

Verilog

Parameters

- [*verilint_compat*](#): Default value is *no*. This indicates the *W316* Tag is switched off. Set this parameter to *yes* to run this Tag.

Constraints

None

Messages and Suggested Fix

The following message appears at the location where a width `<widthl>` of left expression is more than a width `<widthr>` of right expression in an integer conversion expression.

[WARNING] LHS expression width: `<widthl>` is greater than RHS expression width: `<widthr>` [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path.

Potential Issues

Violation may arise when the LHS expression width is greater than the RHS expression width.

Consequences of Not Fixing

<what are the consequences of not fixing the violation>

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the integer conversions have the left expression wider than the right expression. View the violation message to check the width of the RHS of the assignment statement.

In simple integer arithmetic in a testbench, this message can be waived or ignored. In synthesizable logic, it is best to assign a value explicitly sized to 32 bits to avoid confusion.

Example Code and/or Schematic

Consider the following example:

```
module test(in1,in2,out);
input [7:0] in1;
input [7:0] in2;
output [7:0] out;
reg [7:0] out;

integer n,m,l;

always @(in1 or in2)
begin
out = in1 & in2;
```

```
n = m[31:16] + 1[15:0];  
end
```

```
endmodule
```

In the above example, the *W316* Tag reports a violation as the LHS expression width is greater than the RHS expression width.

Default Severity Label

Warning

Tag Group

Verilint_Compat, Lint_Elab_Tags

Reports and Related Files

None

W317

Reports assignment to a supply net

When to Use

Use this Tag to identify assignment to a supply net.

Description

The *W317* Tag reports assignments to supply nets.

Language

Verilog

Parameters

None

Constraints

None

Messages and Suggested Fix

The following message appears at the location where a supply net is being assigned.

[WARNING] Assigning to a supply net is an error

Potential Issues

Violation may arise when an assignment to a supply net is encountered in the design.

Consequences of Not Fixing

Assigning to a supply net, such as `supply0` and `supply1`, nets is an error as supply nets cannot be driven.

How to Debug and Fix

Select the message that you want to debug.

In the HDL window, it takes you to the line where assignment to a supply net is made.

To fix the violation, remove the assignment.

Example Code and/or Schematic

Consider the following example:

```
module W317_top1(i1, i2, o1);  
    input i1, i2;  
    output o1;  
    reg o1;  
  
    supply1 vdd;  
  
    assign vdd = i2;  
  
endmodule
```

In the above example, the Tag reports a violation for a supply net, vdd.

Default Severity Label

Warning

Tag Group

Assign

Reports and Related Files

None

W323

Multiply driven inout net

Language

Verilog

Description

The W323 Tag flags non-tristate inout nets that are driven by multiple drivers.

The W323 Tag considers a net to be tristate if all the drivers in its immediate fan-in are tristate elements.

The W323 Tag ignores nets driven from black boxes.

By default, the W323 Tag ignores bus-holders unless the [strict](#) Tag parameter is set.

By default, the tag extracts all the drivers in the fan-in cone. Set the value of the [MAX_DRIVER_COUNT](#) parameter to a positive integer value to limit the extraction of drivers in fan-in cone to a specified value.

Message Details

The following message appears at the location where a non-tristate inout net `<net-name>` is found to be driven in more than one module instance:

Inout '`<net-name>`' is written in more than one place

Tag Severity

Warning

Suggested Fix

Most probably, there is an error in your logic. Ensure that only one driver drives the net, unless you actually intended a tristate net.

W326

Event variable appearing in a posedge/negedge expression

Language

Verilog

Description

The W326 Tag flags `event` variables used with edges.

Events do not have edges and therefore, should not appear in edge-based expressions.

Message Details

The following message appears at the location where an `event` variable is used in an edge-based expression (`posedge` or `negedge` expressions):

Event variable should not appear in posedge/negedge expressions

Tag Severity

Fatal

Suggested Fix

If you intend to make the code synthesizable, trigger on edges of signals.

Examples

Currently Unavailable

W332

Not all cases are covered - default clause will be used

Language

Verilog

Description

The W332 Tag flags case constructs that do not have all possible clauses described and have a default clause.

The W332 Tag ignores case constructs that have Synopsys `full_case` pragma specified.

While such case constructs are allowed, the W332 Tag informs that the default clause will almost certainly be exercised.

Message Details

The following message appears at the first line of a case construct that does not have all possible clauses described and has a default clause:

```
Not all cases are covered in case statement: default case may  
be used
```

Tag Severity

Warning

Suggested Fix

No fix required, but check default case carefully to make sure you cover all default possibilities.

W333

Unused UDP

Language

Verilog

Description

The W333 Tag flags UDPs (user-defined primitives) that are never instantiated in the design.

While such descriptions are allowed, the W333 Tag helps you clean up your design.

Message Details

The following message appears at the first line of a UDP declaration `<udp-name>` that is never instantiated in the design:

UDP '`<udp-name>`' has not been used in the design

Tag Severity

Warning

Suggested Fix

Check all such cases to make sure you expect the UDP not to be instantiated in the design.

W336

Blocking assignment should not be used in a sequential block (may lead to shoot through)

When to Use

Use this Tag to identify blocking assignment in a sequential block.

Description

The *W336* Tag reports a blocking assignment used in sequential always constructs. Hence, it is best to review these violations during RTL creation phase.

Apart from Verilog style blocking assignments, Tag also reports violation for SystemVerilog style of blocking assignments.

Tag Exceptions

The *W336* Tag fails to run if you set the value of the *fast* parameter to *yes* and the VC SpyGlass Lint product is run, simultaneously.

Language

Verilog

Parameters

- *treat_latch_as_combinational*: Default value is *no*. This indicates the *W336* Tag treats combinational block inferring latch as a sequential block. Set this parameter to *yes* to treat combinational block inferring latch as a combinational block.
- *check_temporary_flop*: Default value is *no* and the *W336* Tag does not report violation for temporary flip-flops. Set the value of this parameter to *yes* to report violation for temporary flip-flops.
- *ignore_local_variables*: Default value is *no* and the *W336* Tag reports violations for variables (left hand side of the assignment) defined inside the sequential block. Set the value of this parameter to *yes* to ignore violations for such cases.
- *ignoreCellName*: Default value is not set. Set this parameter to a comma separated list of PERL regular expressions containing the module names that should be ignored by the *W336* Tag.

Constraints

None

Messages and Suggested Fix

[WARNING] Blocking assignment used inside a <type> sequential block

Where <type> is latch or FlipFlop.

Potential Issues

Violation may arise when a blocking assignment is used inside a sequential block.

Consequences of Not Fixing

When a blocking assignment is used in a sequential block, inherent sequence of operation is implied in simulation. However, the synthesized hardware may behave in a concurrent fashion. Therefore, there is no assurance that the gate-level simulations match RTL level simulations. Also, the intent of the designer may not be fully captured in RTL.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the blocking assignment is used to infer flip-flop or latch in a sequential always block.

View the violation message to check the type of the sequential cell (flip-flop or latch) inferred from the block.

Use only nonblocking assignments in sequential blocks.

Example Code and/or Schematic

Example 1

Consider the following example:

```
module test3(clk, reset, d, q);  
  input clk, reset, d;  
  output q;  
  reg q;  
  
  always @(posedge clk or negedge reset)
```



```
begin
  if (!reset)
    q = 1'b0;
  else
    q = d;
end
```

```
endmodule
```

In the above example, the W336 Tag reports a violation as a blocking assignment is used inside a flip-flop inferred sequential block.

Example 2

Consider the following example of SystemVerilog style blocking assignments:

```
always @ (posedge clk)
  sig++;
```

```
always @ (posedge clk)
  sig2 += 1;
```

In the above example, the Tag reports a violation as the flip-flop is inferred in the design.

Example 3

Consider the following example:

```
module mod(in1, in2, clk, sel, out1);
  input [2:0] in1, in2;
  input clk, sel;
  output [2:0] out1;
  reg [2:0] out1;

  always @(posedge clk)
    if(sel)
      out1 = in1;
    else
      out1 = in2;

endmodule
```

In the above example, the W336 Tag reports a violation as blocking assignments are used in the design.

Example 4

Consider the following example:

```
module test(in1,en,out1);
input  in1;
input  en;
output out1;
reg    out1;

always begin
@ (en)
begin
if(en)
    out1 = in1;
end
end
endmodule
```

In the above example, the W336 Tag does not report a violation when the value of the [treat_latch_as_combinational](#) parameter is set to yes.

Default Severity Label

Warning

Tag Group

Assign

Reports and Related Files

None

W337

Reports illegal case construct labels

When to Use

Use this Tag to report violation for illegal case construct labels.

Description

The W337 Tag reports illegal case construct labels, which includes the following types:

- Labels of type `real`

The case constructs are evaluated by comparing the case construct selector value against each case clause label in turn. As real values cannot be accurately compared, it is recommended not to use real values as case labels.

- Labels containing X, Z, or ?

The case labels containing X have meaning only for the `casex` constructs. The case labels containing Z and ? have meaning for the `casex` and `casez` constructs.

- `time` and `event` variables as labels.

By default, the W337 Tag does not report violations for string variables, which are used as case items. Set the value of the [strict](#) parameter to yes to report violations for such cases.

Language

Verilog

Parameters

[strict](#): The default value is no. Set the value of the `strict` parameter to yes to report violations for string variables, which are used as case items.

Constraints

None

Messages and Suggested Fix

The following message appears at the location of a Tag-violating case construct label.

[WARNING] illegal value as case item

Potential Issues

Violation may arise when an illegal value is used as a case construct label.

Consequences of Not Fixing

Real values cannot be accurately compared.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location of the illegal case item.

Use only integer or reg values as case selectors and in case expressions. If you need to use X, Z, or ? in a label, use the appropriate case type.

Example Code and/or Schematic

```
module top;
    reg source;
    reg dest;
    reg clk;
    reg sel;
    reg out;

    always @(posedge clk)
        case (sel)
            out          : dest <=source;
            1             : dest <=source;
            8'b11110000   : dest <=source;
            8'b1111xxxx   : dest <=source;
            8'b1111zzzz   : dest <=source;
            8'b1111????   : dest <=source;
            0.1           : dest <=source;
            default       : dest <=source;
        endcase

endmodule
```

In the above example, the W337 Tag reports a violation as illegal case construct labels, such as, x, z, and ? are used in the design. The Tag will

Lint Tags

also flag a violation as a `real` value (`0.1`) is used as a case label.

Default Severity Label

Warning

Tag Group

Case

Reports and Related Files

None

W339a

Case equal operator (===) and case not equal (!==) operators may not be synthesizable

Language

Verilog

Description

The W339a Tag flags case equality operators — case equal (===) and case not equal (!==) operators.

The case equality operators compare non-physical values (X) as well as real values, and are frequently used in simulation to check for unknown states. However, they have no physical equivalent. Some synthesis tools may be able to handle these operators but reduce them to their non-case equivalents (for example === gets changed to ==). This may lead to mismatches between pre and post synthesis behavior.

Message Details

The following message appears at the location where a case equality operator is encountered:

operator '<operator-name>' should be avoided in synthesis logic

Tag Severity

Warning

Suggested Fix

First, make sure that your synthesis logic does not depend on the simulation checks. Then, bracket the simulation checks inside `translate_off / translate_on` pragmas so that they are ignored in synthesis.

Examples

Example of Case Equal Operator

Consider the following example that uses case equal (===) operator:

```

module MEM(clk, address, enable, data);
    input [7:0] data;
    input enable, clk;
    output [7:0] address;

    reg [7:0] address;

    always @(posedge clk)
    begin
        if (enable === 1)
            address <= data;
        else
            address <= 8'h00;
        end
    endmodule

```

VC SpyGlass Lint generates the W339a Tag message for every use of case equality operators.

Example of Case Not Equal Operator

In the following example, the case not equal operator (`!==`) is used:

```

module test(in1, in2, clk, out1);
    input [1:0] in1, in2;
    input clk;
    output [1:0] out1;

    reg [1:0] out1;

    wire [1:0] count;

    assign count = (in1 !== 1'b1)? 2'b10 : 2'b01;

    always @(posedge clk)
    begin
        out1 <= (in2 !== count) ? in1 : in2;
    end
endmodule

```

VC SpyGlass Lint generates the W339a Tag message for every use of case

equality operators.

W34

Macro defined but never used

Language

Verilog

Description

The W34 Tag flags macros that are defined but are not used.

You may define a macro but may not use it in the design. Such a case is not an error but generally a mistake.

NOTE: *By default, the W34 Tag is switched off and will be deprecated in a future VC SpyGlass Lint release. The problem reported by this Tag can be handled by most of the commercial compilers.*

Message Details

The following message appears at the location where an unused macro `<macro-name>` is defined in the file, included through the include directive:

```
Macro '<macro-name>' is never used
```

Tag Severity

Info

Suggested Fix

Check all such cases. If you see a macro reported that you think should be used, suspect a possible typing error.

W342

Reports constant assignments that are X-extended

When to Use

Use this Tag to identify constant assignments that are X-extended.

Description

The *W342* Tag reports constant assignments where the size specification of the assigned value is wider than the value specification and the high-order bit/byte is X (unknown). As a result, the constant would be X-extended.

Tag Exceptions

Tag checking is not done for unused macro definitions and unused parameters.

Language

Verilog

Parameters

- *strict*: The default value of this parameter is no. Set the value of this parameter to yes to switch on the Tag and report constant assignments that are X-extended. You can also specify a comma-separated list of Tags as an input to this parameter.

Constraints

None

Messages and Suggested Fix

The following message appears at the location of constant assignment where a constant *<const-name>* is extended by Xs to match the size specification.

[WARNING] Constant *<const-name>* will be X-extended, value has fewer bits than size specification

Potential Issues

Violation may arise when the constant value has fewer bits than size specification.

Consequences of Not Fixing

If the constant value has fewer bits than the size specification, then the constant is X-extended.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the line where the constant that is X-extended.

To fix the violation, it is best to use the correct width and value specification, rather than depending on default extension.

Example Code and/or Schematic

Consider the following examples:

Specified Value	Is Equivalent to...	Whether Message
8'bx0	8'bxxxxxxx0	Yes as the value is X-extended
9'hx0	9'bxxxxx0000	Yes as the value is X-extended
8'hx0	8'bxxxx0000	No as the value is not X-extended

Default Severity Label

Warning

Tag Group

Expression

Reports and Related Files

None

W343

Reports constant assignments that are Z-extended

When to Use

Use this Tag to identify constant assignments that are Z-extended.

Description

The *W343* Tag reports constant assignments where the size specification of the assigned value is wider than the value specification and the high-order bit/byte is Z. As a result, the constant will be Z-extended.

Tag Exceptions

Tag checking is not done for unused macro definitions and unused parameters.

Language

Verilog

Parameters

- *strict*: The default value of this parameter is no. Set the value of this parameter to yes to switch on the Tag and report constant assignments that are Z-extended. You can also specify a comma-separated list of Tags as an input to this parameter.

Constraints

None

Messages and Suggested Fix

The following message appears at the location of constant assignment where a constant *<const-name>* is extended by Zs to match the size specification.

[WARNING] Constant *<const-name>* will be Z-extended, value has fewer bits than size specification

Potential Issues

Violation may arise when the size specification of the assigned value is wider than the value specification and the high-order bit/byte is Z.

Consequences of Not Fixing

Not fixing the violation can make the constant Z-extended.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location of the offending constant that is Z-extended.

To fix the violation, it is best to use the correct width and value specification, rather than depending on default extension.

Example Code and/or Schematic

Consider the following examples:

Specified Value	Is Equivalent to...	Whether Message
8'bz0	8'bzzzzzzz0	Yes as the value is Z-extended
9'hz0	9'bzzzzz0000	Yes as the value is Z-extended
8'hz0	8'bzzzz0000	No as the value is not Z-extended

Default Severity Label

Warning

Tag Group

Expression

Reports and Related Files

None

W345

Presence of an event control in a task or procedure body may not be synthesizable

Language

Verilog, VHDL

Description

The W345 Tag detects presence of an event control in a task or a procedure body. The Tag reports violation at the first event control statement in a task. It also reports the total number of event control statements present in the task.

The presence of an event control within a task or procedure is unsynthesizable by some synthesis tools.

Message Details

Verilog

The following message appears at the first event control statement in a task:

```
Event control statement should not be used in task body, total  
event usage '<num>' in task '<task_name>'
```

Where,

- *<num>* is the total number of event control statements present in a task.
- *<task_name>* is the name of the violating task.

VHDL

The following message appears at the location where a clock edge is used in a sub-program:

```
clock in sub-program body may not be synthesizable
```

Tag Severity

Warning

Suggested Fix

No fix required if this occurs in a testbench. If this is targeted for synthesis, rewrite the task or procedure to avoid the event control.

W346

Task may be unsynthesizable because it contains multiple event controls

Language

Verilog

Description

The W346 Tag flags task descriptions with multiple event control statements.

Task descriptions with multiple event control statements are not synthesizable.

Message Details

The following message appears at the first line of a task description that has multiple event control statements:

Task containing multiple event controls may be unsynthesizable

Tag Severity

Warning

W348

Unspecified width for integer expression in a concatenation

Language

Verilog

Description

The W348 Tag flags concatenation expressions where the width of an integer expression is unspecified.

Concatenations define objects of a specific width. Therefore, each item within the concatenation list must have a defined width. An integer value by default has a width of 32 bits that may probably be larger than you intended and therefore triggers a violation.

The W348 Tag does not report violation for unsized based number used in concatenation. Since, such type of error is a syntax error, it is reported by VC SpyGlass Lint STX_VE_384 Tag.

Message Details

The following message appears at the location of a concatenation expression where one of the integer expression `<int-expr>` in the concatenation list does not have a specified width:

```
Undefined width for integer expression '<int-expr>' in  
concatenation
```

Tag Severity

Warning

Suggested Fix

Provide an explicit width for constant values, making it clear what width is intended.

For example, `{sig1, sig2, 3, sig3}` triggers a violation by the W348 Tag. To avoid this violation, replace it with `{sig1, sig2, 2'b11, sig3}`.

If you are using an expression like `3+x` in concatenation, precompute the

expression and then use as many bits as required in the concatenation. Consider the following example that uses the expression, `3+x`, in concatenation:

```
...{sig1, sig2, 3+x, sig3}...
```

The above code can be replaced with the following:

```
tmp = 3+x;  
...{sig1, sig2, tmp[1:0], sig3}...
```

Examples

The following concatenation example is an error because item 17 (being an integer) is considered not to have a defined width:

```
out1 = {in1, in2, 17, in3}
```

W350

A control character appears inside a string

Language

Verilog

Description

The W350 Tag flags control characters found in strings.

Some EDA tools may not be able to handle control characters in strings.

Message Details

The following message appears at the line where a control character is encountered in a string `<string>` after substring `<substring>`:

control character found after substring `<substring>` in the string `<string>`

Tag Severity

Warning

Suggested Fix

If your editor has a method to display control characters, use that to locate the problem. Otherwise, delete and re-enter the string.

W351

A control character appears inside a comment

Language

Verilog, VHDL

Description

The W351 Tag flags control characters found in comment lines.

Some EDA tools may not be able to handle control characters in comment line.

Message Details

The following message appears at the line where a control character is encountered in a comment line:

```
Control character in comment
```

Tag Severity

Warning

Suggested Fix

If your editor has a method to display control characters, use that to locate the problem. Otherwise, delete and re-enter the string.

W352

Reports “for” constructs with condition expression

When to Use

Use this Tag to identify `for` constructs with condition expression.

Description

The *W352* Tag reports `for` constructs with condition expression that evaluates to a constant.

NOTE: *The W352 Tag supports generate-for block.*

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a `for` construct with constant control expression is encountered.

[WARNING] The 'for' condition is constant - the loop will either never execute or never terminate

Potential Issues

Violation may arise when a `for` construct is used in a condition expression that evaluates to a constant.

Consequences of Not Fixing

When the `for` construct has a constant condition expression, the construct either exits immediately without doing anything or never exits.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the

line where a `for` loop with a constant control expression is specified.
To fix the violation, rewrite the condition.

Example Code and/or Schematic

Example 1

Consider the following example;

```
module mod(out, in, clk, reset);

    output [3:0] out;
    input [3:0] in;
    input clk;
    input reset;
    reg [3:0] out;
    integer i;

    always @ (posedge clk or negedge reset)
        for (i = 0; 4; i = i + 1)
            if ( !reset )
                out[i] = 1'b0;
            else
                out[i] = in[i];

endmodule
```

In the above example, the W352 Tag reports a violation as the `for` condition is constant. The `for` loop will either never execute or never terminate.

The conditional expression of the following `for` loop is a constant:

```
for (i=0 ; 2; i=2)
...
```

Example 2

Consider the following example:

```
`define ITER 5
module W352_mod1(in1, in2, out1, out2, out3, out4);
```

Lint Tags

```

input [5 : 0] in1, in2;
output [5 : 0] out1, out2, out3, out4;
reg [5 : 0] out1, out2, out3, out4;
parameter COND = 5;
integer i;

always@(in1 or in2)
    for(i = 0 ; `ITER; i = i + 1)
        out1[i] = in1[i] | in2[i];

always@(in1 or in2)
    for(i = 0 ; COND; i = i + 1)
        out2[i] = in1[i] | in2[i];

always@(in1 or in2)
    for(i = 0 ; 5; i = i + 1)
        out3[i] = in1[i] | in2[i];

always@(in1 or in2)
    for(i = 0 ; i <= 5; i = i + 1)
        out4[i] = in1[i] | in2[i];

endmodule

```

In the above example, the W352 Tag reports violation for the first three `for` loops as the conditional expression for all of them is constant. These `for` loop will either never execute or never terminate.

However, the W352 Tag does not flag a violation for the last `for` loop as the condition expression does not evaluate to a constant.

Default Severity Label

Warning

Tag Group

Loop

Reports and Related Files

None

W362

Reports an arithmetic comparison operator with unequal length

When to Use

Use this Tag to identify arithmetic comparison operator with unequal length.

Description

The W362 Tag reports arithmetic comparison operations with operands of unequal widths. Following is the list of operators covered under this Tag:

Relational Operators	
Greater than (>)	Greater than or equal to (>=)
Less than (<)	Less than or equal to (<=)
Equality Operators	
logical Equality(==)	logical inequality(!=)
case equality(===)	case inequality(!==)

NOTE: If the count of operators is more than 500 in an expression, then the expression is ignored for Tag checking.

NOTE: The W362 Tag does not report violation for counter cases, for example, `data == data+1`.

If the [nocheckoverflow](#) parameter is set to `yes` or `W362`, the width of the expression is calculated as per the LRM. However, for constants, normal width is considered.

Operator	LRM Width	Normal Width
<code>+, -</code>	Max (lhswidth, rhswidth)	Max (lhswidth, rhswidth) + 1
<code>*</code>	Max (, rhswidth)	lhswidth + rhswidth
<code>/</code>	Max (lhswidth, rhswidth)	lhswidth
<code>%</code>	Max (lhswidth, rhswidth)	RHS (non-static) - RHS width RHS (static) - width of (RHS Value -1)

Handling of Unary Negation:

In case of unary negation, the width is calculated as follows:

- **Variable and based numbers:** If an operand is a variable or a based number, the width is incremented by one. For example, width of `-a[3:0]` is 5, and width of `-3'b101` is 4, etc. This is applicable only when the [*nocheckoverflow*](#) parameter is set to `no`.
- **Constant and unsized based number:** In case of constants, the natural width is calculated first and is then incremented by one. For example, width of `-9` is 5. An unsized based number is treated similarly. For example, width of `-'b101` is 4. However, if you set the [*use_lrm_width*](#) parameter to `yes`, the width is considered as 32 bit.

Tag Exceptions

The *W362* Tag does not report a violation if any one operand is of integer data type. See the [Example 5](#) for details.

Language

Verilog

Parameters

- [*check_sign_extend*](#): Default value is `no`. Set the value of the parameter to `yes` to check for width mismatch due to sign extension in signed comparisons.
- [*handle_zero_padding*](#): Default value is `no`. Set the value of the parameter to `yes` to perform leading zero expansion and truncation of RHS of an assignment. This is performed only if the RHS of the assignment is static.
- [*disable_rtl_deadcode*](#): The default value is `no`. Set the value of the parameter to `yes` to disable violations for disabled code in loops and conditional (if condition, ternary operator) statements.
- [*enable_rtl_deadcode*](#): The default value is `no`. Set this parameter to `yes` to report different violations for dead codes in the RTL resulting from `if/for/while` and ternary operators that are ignored when the [*disable_rtl_deadcode*](#) parameter is enabled.

- [use_lrm_width](#): Default value is `no`. This indicates the *W362* Tag considers the natural width of integer constants. Set this parameter to `yes` to consider the LRM width, which is 32 bits.
- [nocheckoverflow](#): Default value is `no`. This indicates the *W362* Tag does not check the bit-width as per LRM. Set this parameter to `yes` or Tag name to check the bit-width as per LRM.
- [check_static_value](#) and [strict](#): By default, the *W362* Tag does not report violation when the right or left expression is a constant, including parameter, sized or unsized based number, and unsized integer. Setting the [check_static_value](#) parameter to `yes` changes this behavior, and setting the [strict](#) parameter in addition further alters the behavior. The following table summarizes these variations in behavior:

Type of left or right expression	check_staic_value set to <code>no</code>	check_staic_value set to <code>yes</code>	
		strict set to <code>no</code>	strict set to <code>yes</code>
Parameter	Does not report	Reports if the width of constant is larger	Reports any width mismatch
Sized based number (8'h15)	Does not report	Reports if the width of constant is larger	Reports any width mismatch
Unsized based number ('h15)	Does not report	Reports if the width of constant is larger	Reports if the width of constant is larger
Unsized integer (18)	Does not report	Does not report	Reports if the width of constant is larger

See the [Example 6](#) for details.

Also, when set to `yes`, the [check_static_value](#) parameter checks for width mismatch involving static expressions and non-static expressions that contain a static part. Refer to the [check_static_value](#) section for more details.

NOTE: When the [strict](#) parameter is set, the *W362* Tag does not report violation for width mismatch in the for loop condition.

Constraints

None

Messages and Suggested Fix

The following message appears at the location where a width `<widthl>` of a left expression `<exprl>` does not match a width `<widthr>` of a right expression `<exprr>` in an operation of an arithmetic operator `<op-name>`.

[WARNING] For operator (`<op-name>`), left expression: "`<exprl>`" width `<widthl>` should match right expression: "`<exprr>`" width `<widthr>` [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path.

Potential Issues

A violation is reported when an arithmetic operation has operands of unequal length.

Consequences of Not Fixing

For some range of values of the wider operand in arithmetic comparison operations, the comparison operation evaluates to a constant, independent of the value of the narrower operand. This may result in an unexpected behavior.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the width mismatch is found for comparison operator.

This is not a major issue. However, you can avoid possible problems by explicitly comparing sub-expressions of equal width. This also enhances the code readability.

Example Code and/or Schematic

Example 1

Consider the following example:

```
module top(q,clk, d,reset);  
input  clk,d,reset;  
output q;
```

Lint Tags

```

reg q;
reg [3:0] a;
reg [7:0] b;
reg [1:0] data;

always @(posedge clk)
begin
  if (data == reset)    //violation
    a <=17;
  else
    b = 8'hbb;
end
endmodule

```

For the above example, the W362 Tag reports the following violation message:

For operator (==), left expression: "data" width 2 should match right expression: "reset" width 1 [Hierarchy: ':top']

Example 2

Consider the following example in which the `if` statement condition expression is an arithmetic comparison operation involving expressions of unequal bit-width.

```

module test (clk);
  input clk;

  reg [3:0] a;
  reg [7:0] b;
  reg [2:0] data;

  always @(posedge clk)
  begin
    if (data <= (a[1] + b[2]))
      a <= 17;
    else
      b <= 8'hbb;
  end
endmodule

```

Example 3

In the following example, the [check_sign_extend](#) parameter is set to yes.

```
reg signed [9:0] data;
always@(*)
if (data == 'sh300) ;
```

In this example, VC SpyGlass Lint reports the following violation message because the width of the left expression does not match the width of the right expression:

For operator (==), left expression: "data" width 10 should match right expression: "'sh300" width 32 [Hierarchy: ':top ']

Example 4

Consider the following example:

```
module top(input clk);
reg [6:0]a;
reg [2:0] data;
always @(posedge clk)
begin
    for(data =1;data <= 8'hAA; data++)
        a <= 17;
end
endmodule
```

For the above example, when the `strict` parameter is set to yes, the W362 Tag does not report violation for width mismatch in the `for` loop condition. Whereas, by default, the Tag reports violation for such cases.

Example 5

Consider the following example:

```
module top(input clk);
reg [3:0] a, b, data;
integer i1;
int i2;

always @(posedge clk)
begin
if (data == i1)
```

Lint Tags

```

        a <= 17;
    else if(data == i2)
        b = 8'hbb;
    end

endmodule

```

In the above example, the *W362* Tag does not report violations because operands *i1* and *i2* are of integer/int data type.

Example 6

Consider the following example:

```

module top(input clk);
    reg [3:0] a, b, data;
    integer i1;
    int i2;
    parameter P1 = 119;
    parameter P2 = 2;

    always @(posedge clk)
    begin
        if (data == P1)           //violation with check_static_value
            a <= 17;
        else if(data == P2) //violation with check_static_value and strict
            b = 8'hbb;
        if (data == 10'd119)      //violation with check_static_value
            a <= 17;
        else if(data == 2'b10) //violation with check_static_value and strict
            b = 8'hbb;
        if (data == 'd119)        //violation with check_static_value
            a <= 17;
        else if(data == 'b10)
            b = 8'hbb;
        if (data == 119)          //violation with check_static_value and strict
            a <= 17;
        else if(data == 2)
            b = 8'hbb;
    end
end

```

```
endmodule
```

In the above example, violations are reported for different cases (mentioned above in red) when the [check_static_value](#) and/or [strict](#) parameters are set.

Default Severity Label

Warning

Tag Group

Expression, Lint_Elab_Tags

Reports and Related Files

None

W372

A PLI function (\$something) not recognized

Language

Verilog

Description

The W372 Tag flags user-defined PLI functions that are not registered in the Verilog Lint Tagdeck file.

Besides the standard set of PLI functions that ships with each Verilog simulator, you can define your own PLI functions. Then, you need to register each user-defined PLI function to the Verilog Lint Tagdeck by adding a line like the following sample in the PERL Tagdeck file:

```
$dummyPLIfunc = "" if !defined $dummyPLIfunc;
```

Also, the W372 Tag specifically reports for '\$cast' PLI tasks or functions if the [report_cast](#) parameter is set to yes

Message Details

The following message appears at the location where an unregistered user-defined PLI function *<func-name>* is encountered:

PLI function (<func-name>) not recognized

The following message is displayed when the [report_cast](#) parameter is set to yes:

Non-synthesizable system function "<\$cast>" is ignored, a 0 return value is assumed. The signal <signal-name> is unassigned

Tag Severity

Warning

Suggested Fix

Ensure that all the PLI functions to be used are registered in VC SpyGlass Lint (this is locally configurable). If they are, check to make sure you have not mis-typed the name.

W373

A PLI task (`$something`) is used but not recognized

Language

Verilog

Description

The W373 Tag flags user-defined PLI tasks that are not registered in the Verilog Lint Tagdeck file.

Besides the standard set of PLI tasks that ships with each Verilog simulator, you can define your own PLI tasks. Then, you need to register each user-defined PLI task to the Verilog Lint Tagdeck by adding a line like the following sample in the PERL Tagdeck file:

```
$dummyPLItask = "" if !defined $dummyPLItask;
```

Message Details

The following message appears at the location where an unregistered user-defined PLI task `<task-name>` is encountered:

PLI Task (<task-name>) not recognized

Tag Severity

Warning

Suggested Fix

Ensure that all the PLI tasks to be used are registered in VC SpyGlass Lint (this is locally configurable). If they are, check to make sure you have not mistyped the name.

W391

Reports modules driven by both edges of a clock

When to Use

Use this Tag to identify modules driven by both edges of a clock.

Description

The *W391* Tag reports modules where both edges of a clock are used to describe sequential elements.

Tag Exceptions

The *W391* Tag fails to run if you set the *fast* Tag parameter to *yes* and VC SpyGlass Lint product is run.

Also, no Tag checking is done when a clock is driven by a constant value, which is propagated through an assignment or by using the SGDC file.

Language

Verilog, VHDL

Parameters

- *allow_combo_logic_base*: Default value is *no*. Set this parameter to *yes* to skip all VC SpyGlass Lint-generated combinational elements in the fan-in cone while finding the reset/preset source net.
- *check_complete_design*: Default value is *no*. This indicates the *W391* Tag checks within the design unit. Set this parameter to *yes* to check within the complete design.
- *DISABLE_CLOCK_SORTING*: Default value is 0. This parameter when set to 1 stops sorting of clocks from CDC API.

Constraints

- *set_case_analysis*: The *W391* Tag supports the constant value propagation. Therefore, this Tag does not traverse on a blocked path.
- *assume_path*: A path through a black box is treated as a blocked path if there is no *assume_path* constraint specified in the SGDC file for that black box.

Messages and Suggested Fix

For a design unit `<module-name>` where different edges of a clock signal `<clk-name>` have been used to define sequential elements `<hier-flop1-name>` and `<hier-flop2-name>`, respectively, the following message appears at the location where the output net of a second flip-flop `<hier-flop2-name>` is first set.

[WARNING] Design unit '`<module-name>`' uses both edges of clock '`<clk-name>`' (1st at flop '`<hier-flop1-name>`', 2nd at flop '`<hier-flop2-name>`')

Potential Issues

Violation may arise when a module uses both edges of a clock.

Consequences of Not Fixing

As a result of using both the edges, the behavior of that module gets dependent on the duty cycle of the clock.

How to Debug and Fix

Double-click the violation message to view the Incremental Schematic corresponding to the message. The Incremental Schematic displays the following information:

- Negative Edge Clock Path from Clock Source to Flip-Flop Clock Pin.
- Positive Edge Clock Path from Clock Source to Flip-Flop Clock Pin.
- Flip-Flop with Negative Edge Clock.
- Flip-Flop with Positive Edge Clock.

To fix the problem, avoid using both edges of a clock in general. If absolutely necessary, localize such cases as much as possible, that is, use different edges for different modules.

Example Code and/or Schematic

Verilog Example

Consider the following example where a flip-flop `out1` is described using the `posedge` of clock `clk` when a flip-flop `out2` has already been described using the `negedge` of the same clock `clk` in the same module `test`.

Lint Tags

```

module test (out1, out2, in1, in2, clk);
    input  in1, in2, clk;
    output out1, out2;

    reg out1, out2;

    always @(posedge clk)
        out1 = in1;

    always @(negedge clk)
        out2 = in2;
endmodule

```

VHDL Example

Consider the following example where both edges of the `clk` clock signal are used to describe sequential elements:

```

library ieee;
use ieee.std_logic_1164.all
architecture rtl of test is
begin
    P1 : process(clk, reset1)
    begin
        if ( reset1 = '1' ) then
            q1 <= '0';
        elsif( clk'event and clk = '1' ) then
            q1 <= d;
        end if;
    end process P1;

    P2 : process(clk)
    begin
        if( clk'event and clk = '0' ) then
            if ( reset2 = '1' ) then
                q2 <= '0';
            else
                q2 <= d;
            end if;
        end if;
    end if;
end if;

```

```
        end process P2;  
    end rtl;
```

Default Severity Label

Warning

Tag Group

Lint_Clock

Reports and Related Files

None

W392

Reports reset or set signals used with both positive and negative polarities within the same design unit

When to Use

Use this Tag to identify reset or set signals used with both positive and negative polarities.

Description

The *W392* Tag reports reset or set signals that have been used in both negative and positive polarities in the same module.

Tag Exceptions

The *W392* Tag fails to run if you set the *fast* Tag parameter to *yes* and VC SpyGlass Lint product is run. Also, this Tag does not report a violation for set/reset pin driven by constant value that can be propagated.

Language

Verilog, VHDL

Parameters

- *allow_combo_logic_base*: Default value is *no*. Set this parameter to *yes* to skip all VC SpyGlass Lint-generated combinational elements in the fan-in cone while finding the reset/preset source net.
- *checksyncretst*: Default value is *yes*. This indicates the *W392* Tag considers synchronous resets for Tag checking. The Tag tries to find the synchronous resets from RTL *always* blocks. The Tag also honors the *reset -sync* constraint and looks for the out-of-phase flip-flops. Set this parameter to *no* to ignore synchronous resets.
- *check_complete_design*: Default value is *no*. This indicates the *W392* Tag checks within the design unit. Set this parameter to *yes* to check within the complete design.
- *fast*: Default value is *no*. Set this parameter to *yes* to suppress synthesis. Hence, the *W392* Tag will be switched off.
- *waiver_compat*: Default value is *no*. If you set the value of this parameter to *yes* or <Tag-name>, it ensures that the Tag does not generate the

line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.

- **[check_latch](#)**: Default value is no and the Tag reports the edge triggered cases only. Set the `check_latch` parameter to yes to enable the Tag to report non-edge triggered cases also.
- **[IGNORE_SYNC_RESET_FOR_ASYNC_RESET_FLOP](#)**: The default value is yes. Set the value of the parameter to no to ignore sync set or reset for flops which have async set or reset.
- **[REPORT_FILE_LINE_ON_SOURCE](#)**: The default value is 0. Set the value of the parameter to 1 to report violations on back reference on RTL data net of fanout flop.

Constraint(s)

- **[set_case_analysis](#)**: The W392 Tag supports the constant value propagation. Therefore, this Tag does not traverse on a blocked path. For black boxes, this Tag traverses further only if the [assume_path](#) constraint is specified in the SGDC file.

Messages and Suggested Fix

Message 1

The following message appears at the location where a different polarity of a reset signal `<rst-name>` is used when another polarity has already been used in the same module `<module-name>` in a file `<file-name>`, line `<num>`.

[WARNING] Reset '`<rst-name>`' also used with different polarity in module '`<module-name>`'(file: '`<file-name>`' line: '`<num>`')

Potential Issues

Violation may arise when two different IP blocks, each operating perfectly fine with a given polarity of reset/set, are connected together at a SoC level.

Also, in some cases, the designer might intend to have mutually exclusive operations (usually different modes of design).

Consequences of Not Fixing

When both polarities of reset/set signal are used, one logic block always

remain in a reset/set state. If a reset/set signal is high, the logic that operates on positive reset/set polarity would be reset/set. However, when the reset/set signal is de-asserted, some other portion of logic would be reset/set. The usage leads to mutually exclusive blocks of logic.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location of first usage of the reset.

Also, the corresponding Incremental schematic displays the following information:

- Negative polarity Reset Path from Reset Source to Flip-Flop Reset Pin.
- Positive polarity Reset Path from Reset Source to Flip-Flop Reset Pin.
- Flip-Flop with Negative polarity Reset.
- Flip-Flop with Positive polarity Reset.

The violation message also shows the file and line number of reset used with different polarity. User can trace the RTL code with this information to see the other usage of the reset.

NOTE: *If the Incremental schematic is not supported for the violation, it means the violation is related to synchronous resets.*

In majority of cases, such behavior is unintentional. To fix this problem, review the design and change the reset logic so that externally reset is active only in one state.

Message 2

The following message appears if the [check_latch](#) parameter is set as the Tag will check for latches:

[WARNING] Reset <rst-name> used with different polarity in the design at latches <latch1-name> and <latch2-name>

Potential Issues

Violation may arise when two different IP blocks, each operating perfectly fine with a given polarity of reset/set, are connected together at a SoC level.

Also, in some cases, the designer might intend to have mutually exclusive operations (usually different modes of design).

Consequences of Not Fixing

When both polarities of reset/set signal are used, one logic block always

remain in a reset/set state. If a reset/set signal is high, the logic that operates on positive reset/set polarity would be reset/set. However, when the reset/set signal is de-asserted, some other portion of logic would be reset/set. The usage leads to mutually exclusive blocks of logic.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location of first usage of the reset.

Also, the corresponding Incremental schematic displays the following information:

- Negative polarity Reset Path from Reset Source to Latch Reset Pin.
- Positive polarity Reset Path from Reset Source to Latch Reset Pin.
- Latch with Negative polarity Reset.
- Latch with Positive polarity Reset.

The violation message also shows the file and line number of reset used with different polarity. User can trace the RTL code with this information to see the other usage of the reset.

Example Code and/or Schematic

Verilog

Example 1:

Consider the following example:

```
module test3( Q1, Q2, DataIn, C_SCLK1, C_SRST1 );
input [2:0] DataIn;
input C_SCLK1;
input C_SRST1;
output [2:0] Q1, Q2;
reg clk;
reg [2:0] Q1,Q2,Q3;
always @( posedge C_SCLK1 or posedge C_SRST1 )
begin
  if (C_SRST1) //VIOLATION
    Q1 = 2'b11;
  else
    Q1 = DataIn[1:0];
end
```

Lint Tags

```

end
always @( posedge C_SCLK1 or negedge C_SRST1 )begin
if (!C_SRST1)
    Q2 = 2'b11;
else
    Q2 = DataIn[1:0];
end
endmodule

```

In the above example, W392 Tag reports a violation as both polarities of reset signal, C_SRST, is used in the same module.

Example 2:

Consider the following example, where the [check_latch](#) parameter has been set to yes:

```

module test (y1, y2, data1, data2, enable, preset, clear);
    input data1, data2, enable, preset, clear;
    output y1, y2;
    reg y1, y2;

    always @(enable or clear or preset or data1)
        begin: forset1_PROC
            if (clear)
                y1 = 0;
            else
                if (preset) //high active "preset"
                    y1 = 1;
                else
                    if (enable)
                        y1 = data1;
            end

    always @(enable or clear or preset or data2)
        begin: forset2_PROC
            if (clear)
                y2 = 0;
            else
                if (!preset) //low active "preset"

```

```

        y2 = 1;
    else
        if (enable)
            y2 = data2;
        end
    endmodule

```

For the above example, the Tag reports the following violation message:
Reset 'test.preset' used with different polarity in the design at latches 'test.y2' and 'test.y1'

VHDL

Consider the following example where both polarities of the `rst` reset signal have been used in same architecture.

```

library IEEE;
use ieee.std_logic_1164.all;

entity ent is
    port (d, clk, rst: in bit; q1,q2: out bit);
end ent;

architecture behav of ent is
begin
    process(clk,rst)
    begin
        if (rst = '1') then
            q1 <= '0';
        elsif clk'event and clk ='1' then
            q1 <= d;
        end if;
    end process;
    process(clk,rst)
    begin
        if rst = '0' then
            q2 <= '0';

```

Lint Tags

```

        elsif clk'event and clk = '1' then
            q2 <= d;
        end if;
    end process;
end behav;

```

The schematic for this example is given below:

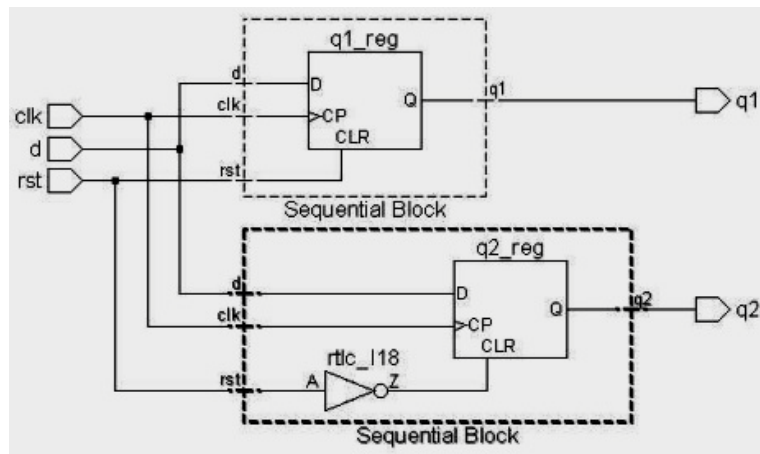


FIGURE 1. Incremental schematic

Default Severity Label

Warning

Tag Group

Lint_Reset

Reports and Related Files

None

W395

Multiple asynchronous resets or sets in a process or always may not be synthesizable

Language

Verilog, VHDL

Description

The *W395* Tag reports if more than one asynchronous reset or set signals exist in the same `process` or `always` block.

Some synthesis tools may be able to handle `process` or `always` blocks with multiple asynchronous resets or sets but such designs are unlikely to be portable.

NOTE: *The Tag supports generate-if and generate-for blocks.*

Message Details

Verilog

The following message appears at the start of an `always` block that uses multiple asynchronous set or reset signals (`<rst-list>`):

Multiple set/reset signals `<rst-list>` used in the `always` block may not be synthesizable [Hierarchy: '`<hier-path>`']

VHDL

The following message appears at the start of a `process` block that uses multiple asynchronous set or reset signals (`<rst-list>`):

Multiple set/reset signals `<rst-list>` used in the `process` block may not be synthesizable

Severity

Warning

Suggested Fix

Precompute a single condition under which the `process` or `always` block should be reset or set, and use that condition as a reset or set inside the

process or always block.

Examples

VHDL

Consider the following example where the process block uses two reset signals (rst1 and rst2):

```
library ieee;
use ieee.std_logic_1164.all

process (clk1, rst1, rst2)
begin
    if (rst1 = '1') then
        t <= '0';
    elsif (rst2 = '1') then
        t <= '0';
    elsif (clk1'event and clk1 = '1') then
        t <= not in1;
    end if;
end process;
```

For this example, VC SpyGlass Lint generates the following message:

Multiple set/reset signals 'rst1', 'rst2' used in the process block may not be synthesizable

Verilog

Consider the following example where the always block uses two reset signals (rst1 and rst2):

```
always @(posedge clk, posedge rst1, posedge rst2)
begin
    if (rst1)
        t <= 1'b0;
    else if (rst2)
        t <= 1'b0;
    else if (clk)
        t <= in1 & in2;
end
```

For this example, VC SpyGlass Lint generates the following message:
Multiple set/reset signals 'rst1', 'rst2' used in the always
block may not be synthesizable

W396

A process statement has clock signal, but no asynchronous reset signal

Language

VHDL

Description

The W396 Tag flags processes that have a clock signal but no asynchronous reset signal.

If a process statement has a clock signal then an asynchronous reset signal must be given so that the corresponding sequential circuit starts from a known state.

NOTE: *You can enable the W396 Tag by either specifying the `configure_lint_tag -enable -tag W396` command. However, this Tag will not run if you set the [fast](#) Tag parameter to `yes` and `lint` product is run.*

Message Details

Following message appears at the start of a process that has a clock signal but no asynchronous reset signal:

```
Process has a clock signal, but not an asynchronous reset
signal
```

Severity

Warning

Suggested Fix

Use this Tag to check processes which are not asynchronously reset. If you are using only synchronous reset, ignore the Tag.

Examples

Consider the following example where the process has a clock signal `clk` but no asynchronous reset signal:

```
entity test is
```

```
    port(in1: in bit;  
         clk: in bit;  
         q: out bit);  
end test;  
  
architecture behav of test is  
begin  
    process (clk)  
    begin  
        if (clk'event and clk = '1') then  
            q <= in1;  
        end if;  
    end process;  
end behav;
```

For this example, VC SpyGlass Lint generates the W396 Tag message.

W397

Destination of an assignment is an IN port

Language

VHDL

Description

The W397 statement flags signal assignments to input ports.

As you cannot drive an input, such assignments are design errors. An input bit should be driven only by an input port.

Message Details

The following message appears at the location where an input port `<port-name>` is assigned a value:

Cannot assign to IN port '<port-name>'

Severity

Fatal

Suggested Fix

Remove the assignment to the input port. If an OR or WIRE-OR function is necessary, create that function explicitly.

W398

Reports a case choice when it is covered more than once in a case statement

When to Use

Use this Tag to identify duplicate case constructs.

Description

The *W398* Tag reports duplicate choices in case construct. It is most likely that this error occurs while using don't-care values in the case labels. It is the only way that this condition can occur in VHDL. Consider the following Verilog example:

```
case (choice)
  100: ...
  10?: ...// violation
  default: ...
endcase;
```

Consider the following VHDL example:

```
library ieee;
use ieee.std_logic_1164.all
case (choice)
  when "100" => ...
  when "10-" => ...-- violation
  when others => ...
end case;
```

Tag Exceptions

The *W398* Tag does not process case constructs when the `parallel_case` pragma is specified.

Language

Verilog, VHDL

Parameters

- *strict*: The strict parameter is applicable only for the Verilog Tag. Default value for this parameter is `no`. In this case, the Tag does not report

violation for the non-static case labels. Set the value of the parameter to *yes* or *<Tag-name>* to report violations for non-static case labels.

- ***waiver_compat***: Default value is no. If you set the value of this parameter to *yes* or *<Tag-name>*, it ensures that the Tag does not generate the line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.

Constraints

None

Messages and Suggested Fix

Verilog

The following message appears at the location where a duplicate choice *<duplicate_case>* of an already existing case label *<existing-case-label>* in case construct is encountered.

[WARNING] Case '*<duplicate_case>*' covered more than once at '*<existing-case-label>*'

Potential Issues

Violation may arise when a case choice is covered more than once in a case statement.

Consequences of Not Fixing

Repeated clauses in case construct are redundant and may result in an error for some applications. All such redundant choices are not covered by simulator and synthesis tool.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the duplicate case label is used.

To find the initial usage of the overlapping case label, scroll up the HDL Viewer window or search backward for the second expression given in the violation message.

To fix this problem, carefully examine offending case items and modify the coding to avoid overlap.

VHDL

The following message appears at the location of a statement *<stmt>* when a clause value *<valuec>* overlaps the value of another clause *<valuee>* found earlier at a line *<num>*.

[WARNING] Choice *<valuec>* overlaps with another entry *<valuee>* (line no *<num>*) in *<stmt>* statement

Where *<stmt>* can be a case construct or a selected signal assignment statement.

Potential Issues

Violation may arise when a clause value overlaps with another clause value in a case construct or in a signal assignment statement.

Consequences of Not Fixing

Repeated clauses in a case construct or a signal assignment statement is redundant and may result in an error for some applications. All such redundant choices are not covered by simulator and synthesis tool.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the duplicate case label is used.

To find the initial usage of the overlapping case label, scroll up the HDL Viewer window or search backward for the second expression given in the violation message.

To fix this problem, carefully examine offending case items and modify the coding to avoid overlap.

Example Code and/or Schematic

Verilog Example

Consider the following example where the choice 2'b00 is repeated:

```
module test(a, b, c, o1);
    input [3:0]c;
    input [3:0]a;
    input [1:0]b;
    output [3:0]o1;
    reg [3:0]o1;
```

```

always@(c or b)
begin
  case(b)
    2'b00 : o1 = a || c;
    2'b01 : o1 = a & c;
    2'b00 : o1 = a | c;      //violation by default
      c : o1 = a ^ b;
    2'b11 : o1 = a ^ c;
      c : o1 = a || b;      //violation when strict is set
    default : o1 = 0;
  endcase
end
endmodule

```

In the above example, the W398 Tag reports a violation for the case label 2'b00 because it is repeated within the same case block. When the strict parameter is set, the W398 Tag reports a violation for the case label c because it is a non-static case label.

VHDL Example

Example of a CASE Construct

In the following example, the clauses 0-0 and -00 (at line 13) overlap:

```

library ieee;
use ieee.std_logic_1164.all

12: case sig3 is
13:   when "-00" => sig1 <= sig2;
14:   when "0-0" => sig1 <= sig2 + 10;
15:   when "000" => sig1 <= sig2 + 20;
16:   when "010" => sig1 <= sig2 + 20;
17:   when "011" => sig1 <= sig2 + 20;
18:   when "100" => sig1 <= sig2 + 20;
19:   when "101" => sig1 <= sig2 + 20;
20:   when "110" => sig1 <= sig2 + 20;
21:   when "111" => sig1 <= sig2 + 20;
22:   when others => null;
23: end case;

```

Example of a Select Statement

In the following example, the clauses --- and ZX1 overlap:

```
with a select
  b <= 0 when "ZX1",
      6 when "ZX0",
      8 when "---",
      unaffected when others;
```

Default Severity Label

Warning

Tag Group

Case

Reports and Related Files

None

W401

Clock signal is not an input to the design unit

Language

Verilog, VHDL

Description

The W401 Tag flags clock signals that are not inputs to the modules where the clock signals are used to describe flip-flops.

The W401 Tag flags clock signals that are gated, buffered, or internally generated in the module where they are used.

Using internally generated clocks may cause problems in timing budgeting and in VC SpyGlass Lint DFT tools.

The W401 Tag does not flag a violation for clock pin driven by a constant value, which can be propagated through assignment or using the SGDC file.

Message Details

The following message appears at the location where a Tag-violating clock *<clk-name>* is used in design unit *<du-name>*:

Clock '*<clk-name>*' is not an input to design unit '*<du-name>*'

Tag Severity

Warning

Suggested Fix

Localize clock generation and gating to a single module if possible, so that timing and test issues can be managed carefully with respect to that one module rather than in many locations in the design.

Examples (Verilog)

Consider the following example where the DFF_clk of flip-flop out1 is internally generated:

```
module test(in1, in2, in3, out1);
```

```

    input in1, in2, in3;
    output out1;

    my_clock mod1(in1, in2, in3, out1);
endmodule

module my_clock(in1, in2, clock, out1);
    input in1, in2, clock;
    output out1;

    reg DFF_clk, out1;

    always@(posedge clock)
    begin
        DFF_clk <= in1;
    end

    always@(posedge DFF_clk)
    begin
        out1 <= in2;
    end
endmodule

```

For this example, VC SpyGlass Lint generates the following message:
 Clock 'test.mod1.DFF_clk' is not an input to design unit
 'my_clock'

Examples (VHDL)

Consider the following example where the clock signal `intClk` of flip-flop output is internally generated (process `FF_CLK`) in architecture `struct_CLK_Source` of entity `ent_CLK_Source`:

```

library ieee;
use ieee.std_logic_1164.all

entity ent_CLK_Source is
    port (data: in Bit;
          clk_in: in Bit;

```

Lint Tags

```

        reset: in Bit;
        output: out Bit);
end ent_CLK_Source;

architecture struct_CLK_Source of ent_CLK_Source is
    signal intClk: Bit;
begin
    FF_CLK: process(clk_in)
    begin
        if(clk_in'event and clk_in = '1') then
            intClk <= data;
        end if;
    end process FF_CLK;

    FF: process(intClk, reset)
    begin
        if (reset = '1') then
            output <= '0';
        elsif (intClk'event and intClk = '1') then
            output <= data;
        end if;
    end process FF;
end struct_CLK_Source;

entity ent_Test is
    port(d: in Bit;
        clk: in Bit;
        rst: in Bit;
        dataOut: out Bit);
end ent_Test;

architecture struct_Test of ent_Test is
    component CLK_Source
        port (data: in Bit;
            clk_in: in Bit;
            reset: in Bit;
            output: out Bit);
    end component;

```

```
begin
  inst1: ent_CLK_Source
    port map (data => d,
              clk_in => clk,
              reset => rst,
              output => dataOut);
end struct_Test;
```

For this example, VC SpyGlass Lint generates the following message at the location of instantiation `inst1` of entity `ent_CLK_Source`:

Clock 'ent_Test.inst1.intClk' is not an input to design unit
'ent_CLK_Source(struct_CLK_Source)'

W402

If you internally generate reset signals, do so in a single module instantiated at the top-level of the design

Language

VHDL

Description

The W402 Tag flags reset signals that are internally generated at level other than the top-level of the design.

Internal reset generation creates significant problems for VC SpyGlass Lint DFT methodologies. If you must gate resets, you should localize gating to one module at the top-level of the design and provide a method to disable gating in test mode.

The W402 Tag reports violation if any one or more condition described below occurs:

- Top-level module has asynchronous resets and they are not ports
- Asynchronous resets are generated in more than one instance
- All asynchronous resets are generated in an instance, but all output ports of that instance are not reset ports.

Message Details

Following message appears at the location where a reset signal `<rst-name>` is generated at a level other than the top-level of the design:

```
Gated reset <rst-name> is not generated in single module at top level
```

Severity

Warning

Suggested Fix

Consider gating the generated reset with an externally controllable test

mode signal.

W402a

Synchronous reset signal is not an input to the module

Language

Verilog

Description

The W402a Tag flags synchronous reset signals that are not inputs to the module.

A signal appears to be used as a synchronous reset and is generated inside the same module. This can cause difficulties in verification where it may be desirable to start all logic in a reset state.

The W402a Tag flags gated, buffered, or internally generated synchronous reset signals.

NOTE: *Synchronous reset detection in the VC SpyGlass Lint product is heuristic. Hence, this Tag may report some false errors.*

Message Details

The following message appears at the location where a reset signal `<rst-name>` is used in the module `<module-name>` that is not an input to the module:

Reset '`<rst-name>`' is not an input to the module '`<module-name>`'

Severity

Warning

Suggested Fix

If the error is false, consider waiving it.

Examples

Consider the following example where synchronous reset signal `rst` is not an input to the module:

```
module test(in, out, clk);
```

```
input [3:0] in;
input clk;
output [3:0] out;
reg [3:0] out;
reg rst;

always@(posedge clk)
begin
    if(rst) out <= 0;
    else out <= in;
end
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Reset 'rst' is not an input to the module 'test'

W402b

Asynchronous set/reset signal is not an input to the module

Language

Verilog

Description

The W402b Tag flags asynchronous set/reset signals that are not inputs to the module.

The W402b Tag flags gated, buffered, or internally generated asynchronous set/reset signals.

Such set/reset signals may cause problems for VC SpyGlass Lint DFT tools.

Parameters

[*waiver_compat*](#): Default value is no. If you set the value of this parameter to yes or <Tag-name>, it ensures that the Tag does not generate the line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.

Message Details

The following message appears at the location where the output of a flip-flop <flop-name> is first assigned when its asynchronous <pin-type> signal <pin-name> is not an input to the module <module-name>:

```
<pin-type> '<pin-name>' to flop '<flop-name>' (line <num>, file
<file-name>) is gated or internally generated (in module
<module-name>)
```

where <pin-type> is set or reset and <pin-name> is the name of set/reset.

Tag Severity

Warning

Suggested Fix

Consider gating the generated set/reset with an externally controllable test

mode signal.

Examples

Currently Unavailable

W414

Reports nonblocking assignment in a combinational block

When to Use

Use this Tag to identify nonblocking assignments in a combinational block.

Description

The *W414* Tag reports violations:

- When a nonblocking assignment is followed by a blocking assignment in a combinational block
- When a nonblocking assignment (signal used on LSH of assignment) is used in a clock path, unless the block is a latch

For combinational blocks, the Tag reports violations only for the following cases:

- A nonblocking assignment is followed by the blocking assignments. For example, consider the following cases:

Case 1

```
out1 <= in1 & in2;
out2 = in3 & in4;
```

Case 2

```
out1 <= in1 & in2;
out2 < = in3 & in4;
```

Case 3

```
out1 = in1 & in2;
out2 <= in1 & in2;
```

- Signal that is assigned using the nonblocking assignments is used in the clock path of a flip-flop. For example, consider the following case:

Case 1

```
always @(in1 or in2)
  clk <= in1;
always @(posedge clk) //used as a clock
  out1 <= in1 & in2;
```

When the parameter *fast* is set to *yes*, the RTL version of the W414 Tag is run.

The RTL version of the W114 Tag reports violation for those non-blocking assignments that are followed by a blocking assignment in a combinational block. For example, consider the following cases:

Case 1

```
out1 <= in1 & in2;  
out2 = in3 & in4;
```

Case 2

```
out1 <= in1 & in2;  
out2 <= in3 & in4;
```

Case 3

```
out1 = in1 & in2;  
out2 <= in1 & in2;
```

Language

Verilog

Parameters

- *treat_latch_as_combinational*: This indicates that the W414 Tag treats combinational block inferring latch as a sequential block. Set this parameter to *yes* to treat the combinational block inferring latch as a combinational block. When the RTL version of the Tag is run, all the latches are combinational except *always_latch*. Set the *treat_latch_as_combinational* parameter to *yes* to treat *always_latch* as a combinational block for the RTL version.
- *fast*: The default value is *no*. Set the value of the parameter to *yes* to run the RTL version of the W414 Tag.

Constraints

None

Messages and Suggested Fix

The following message appears at the location where a nonblocking assignment is used in a combinational *always* construct:

[WARNING] nonblocking assignment should not be used in a

combinational block

Potential Issues

Violation may arise when a nonblocking assignment is used in a combinational block.

Consequences of Not Fixing

Not fixing the violation may result in unexpected code behavior.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the nonblocking assignment is used for the specific signal. This assignment is specified inside a combinational `always` block.

You can confirm whether the `always` block is combinatorial using one of the following ways:

- Visual inspection of the block
- Scrolling up the HDL Viewer window till you reach the `always` block
- If the `always` block is a long block, view the HDL in an editor, and search backward for the nearest `always` keyword

If the `always` block is not combinational and is a latch, set the value of the [treat_latch_as_combinational](#) parameter as `yes`.

To fix this problem, use only blocking assignments in combinational blocks.

Example Code and/or Schematic

Consider the following example:

```
module test3(set, reset, p, q);
input set, reset;
output p, q;
reg p, q;

always @ (posedge p)
    q <= set;
always @(set or reset)
begin
    p <= set & reset;
    q <= set | reset;
```

```
end
```

```
endmodule
```

In the above example, the *W414* Tag reports a violation as a nonblocking assignment is used in a combinational `always` block, when the *fast* parameter is not set.

Default Severity Label

Warning

Tag Group

Assign

Reports and Related Files

None

W415

Reports variable/signals that do not infer a tristate and have multiple simultaneous drivers

When to Use

Use this Tag to identify variable/signals that do not infer a tristate and have multiple simultaneous drivers.

Description

The *W415* Tag reports variables/signals that do not infer tristate nets and have multiple simultaneous drivers.

The *W415* Tag does not report violation for tristate nets whose immediate fan-in drivers are tristate elements. However, the Tag reports a violation when an enable signal for two tristate elements is same as both the elements get enabled at the same point of time.

This Tag reports variables/signals that are driven in more than one sequential *always/process* constructs, instances, and/or concurrent statements. Such cases are errors unless the variable infers a tristate net.

Tag Exceptions

Following are the expectations to the *W415* Tag:

- The *W415* Tag fails to run if you set the *fast* Tag parameter to *yes* and VC SpyGlass Lint product is run.
- By default, this Tag does not report any violation if the multiple driven signals/variables are not loaded or they have hanging driver input.
- In VHDL, the *W415* Tag reports those data types that have built-in resolution function, such as `std_logic`.

Language

Verilog, VHDL

Parameters

- *strict*: By default, the *W415* Tag ignores instances, bus-holders and top-level inout ports as drivers. Set this parameter to consider instances, bus-holders and top-level inout ports as drivers. For all other modules,

inout ports are always checked for Tag violation irrespective of the value of this parameter.

NOTE: *A violation is reported even without the strict parameter, if two or more output terminals are connected together with an inout port. No violation is reported (even if the strict parameter is set), if an inout port is connected with the non-enabled tristate (disabled or the tristate having unknown value) driver.*

- [*assume_driver_load*](#): Default value is no. This indicates the W415 Tag does not flag any violation, if the multiple driven signal/variable is not loaded or have hanging driver input. Set the value of the parameter to yes to report violations for such cases.
- [*checkconstassign*](#): Default value is no. This indicates the W415 Tag ignores nets that are driven multiple times by the same constant value. Set this parameter to yes to report such cases.
- [*check_bbox_driver*](#): Default value is no. Set this parameter to yes to consider the black-box as a valid driver.
- [*ignore_equivalent_drivers*](#): Default value is yes. Set the value of the parameter to no to report violations for a net, which is driven by two identical instances of basic two input gates of type OR/NOR/AND/NAND and both instances are driven by same input signals.
- [*ignore_greybox_drivers*](#): Default value is no. Set the value of the parameter to yes to consider greybox as a valid driver, if this parameter is set to 'yes' then Tag will not consider greybox as a valid driver.
- [*handle_equivalent_drivers*](#): The default value is no. Set the value of the parameter to yes to report violations when a net is simultaneously driven by two identical instances of two input gates of type OR, NOR, AND, or, NAND and both instances are driven by same input signals.
- [*MAX_DRIVER_COUNT*](#): The default value is 200. Set the value of the parameter to a positive integer value to limit the extraction of drivers in fan-in cone to a specified value.

Constraints

None

Messages and Suggested Fix

The following message appears at the location where a signal

<sig-name> is found to be driven by more than one sequential always/process construct, instances, and/or concurrent statements:

[WARNING]Signal '*<sig-name>*' has multiple simultaneous drivers

Potential Issues

Violation may arise when a signal has multiple simultaneous drivers.

Consequences of Not Fixing

A net that is not a tristate net should have a unique driver. If such nets are driven by more than one net, there is a possibility of an electrical conflict. For example, consider a net driven by outputs of two different AND gates. If one output is 0 and other output is 1, the driven net is in a non-digital state. Such cases lead to excessive current flows from one AND gate to another AND gate causing chip electrical failures, causing chip malfunction. The effect of a non-digital net can also propagate to its fan-out points. For such fan-out points, the net behaves as if it was floating resulting in further chip electrical and functional failures.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic shows the multiple driven net and a corresponding load for that net.

Most probably, there is an error in your logic. To fix the violation, ensure that variable/signal does not have multiple drivers, unless you actually intended tristate logic.

Example Code and/or Schematic

Example 1

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
port(
in1,in2 : in std_logic;
output  : out std_logic
);
end test;
architecture rtl of test is
```

```
begin
output <= in1 ;
output <= in2 ;
end rtl;
```

In the above example, the *W415* Tag reports a violation as the signal, *output*, has multiple simultaneous drivers, *in1* and *in2*.

Default Severity Label

Warning

Tag Group

MultipleDriver

Reports and Related Files

None

W415a

Signal may be multiply assigned (beside initialization) in the same scope

Language

Verilog

Description

The *W415a* Tag reports signals that are assigned multiple times within the same block.

The Tag reports a violation when an assignment overrides previous assignment other than the initialization statement within the same block.

If a signal is multiply assigned in parallel conditional blocks and if those conditions can be true simultaneously, the second assignment may override the first. The Tag reports such assignments.

NOTE: *The Tag supports the multi-line functionality.*

By default, the Tag checks for all kinds of multiple assignments (both blocking and nonblocking) within the same scope. Use the [checknonblocking](#) Tag parameter to restrict Tag checking to nonblocking assignment statements. Use the [checkblocking](#) Tag parameter to restrict Tag checking to blocking assignment statements.

The Tag only reports for assignments at the RTL description level irrespective of the logic inferred after synthesis.

By default, the [ignore_reinitialization](#) parameter not set and the *W415a* Tag reports violations for re-initialization when the [check_initialization_assignment](#) parameter is set to `yes`. Set the [ignore_reinitialization](#) parameter to `yes` to ignore violations for re-initialization assignments inside the `for` loops.

By default, the Tag reports a violation when an entire vector is initialized with an initialization value and a subset of the vector is overwritten. If you set the [check_initialization_assignment](#) parameter to `yes`, the Tag does not report a violation for such cases. However, at this parameter value (`yes`), the Tag reports a violation when a vector is initialized with an initialization value and the entire vector is assigned again with another value.

By default, the tag reports violation for signals, which are inside `if/else` or

case statement. Set the value of the [ignore_if_case_statement](#) parameter to yes to ignore violations for signals, which are inside if/else or case statement.

Example 1

In the following example, the [check_initialization_assignment](#) parameter is set to yes:

```
reg [3:0] b;
always @(*)
begin
    if(inp)
    begin
        b = a;
        b[2] = 1'b0;
    end
end
```

In this case, the vector `b` is initialized with an initialization value and a subset of the vector is overwritten. Therefore, the *W415a* Tag does not report a violation in this case.

Example 2

In the following example, the [check_initialization_assignment](#) parameter is set to yes:

```
reg [3:0] b;
always @(*)
begin
    if(inp)
    begin
        b = a;
        b[3:2] = 1'b0;
    end
end
```

In this case, the vector `b` is initialized with an initialization value and a subset of the vector is overwritten. Therefore, the *W415a* Tag does not report a violation in this case.

Example 3

In the following example, the [check_initialization_assignment](#) parameter is set to yes:

```
reg [3:0] b;
always @(*)
begin
    if(inp)
    begin
        b = a;
        b = 1'b0;
    end
end
```

In this case, the vector b is initialized with an initialization value, and the entire vector is assigned again with another value. Therefore, the *W415a* Tag reports a violation in this case.

Example 4

In the following example, the [check_initialization_assignment](#) parameter is set to yes:

```
reg [3:0] b;
always @(*)
begin
    if(inp)
    begin
        b = a;
        b[3:0] = 1'b0;
    end
end
```

In this case, the vector b is initialized with an initialization value, and the entire vector is assigned again with another value. Therefore, the *W415a* Tag reports a violation in this case.

By default, the Tag reports assignment to a variable inside a conditional construct if the variable is also being assigned non-static value in a nonblocking manner before the conditional construct as in the following examples:

Example 1

Consider the following example:

```
always @ (clk)
begin
    if (clk)
    begin
        out1<= in1; //Assignment of non-static value
        if(en)
            out1<= in2; // Violation
        end
    end
end
```

Example 2

Consider the following example:

```
always @ (clk)
begin
    out1<= in1; //Assignment of non-static value
    if (clk)
    begin
        out1<= in1; //Violation
    end
end
```

In the above example, the Tag reports a violation as the variable is assigned a non-static value.

The *W415a* Tag does not report assignment to a variable inside a conditional construct if the variable is also being assigned static value in a nonblocking manner before the conditional construct as in the following examples:

Example 1

```
always @ (clk)
begin
    if (clk)
    begin
        out1<= 1'b0; //Assignment of static value
        if(en)
            out1<= in2; // No Violation
    end
end
```

```

    end
end

```

Example 2

```

always @ (clk)
begin
    out1<= 1'b0; //Assignment of static value
    if (clk)
    begin
        out1<= in1; // No Violation
    end
end

```

If you set the [*ignore_nonBlockCondition*](#) parameter to *yes*, then no violation is reported for case mentioned above. For example,

```

...
a <= b;
if (C1)
begin
    a <= b1; // ignore_nonBlockCondition set to yes
end
...

```

The *W415a* Tag does not report violation if two *if* conditions are mutually exclusive as only one path will be executed. For example:

```

if(sel)
    out = in1;
if(!sel)
    out = in2; //Only one condition will be true in above //case
if(sel[1])
    out = in1;
if(!sel[1])
    out = in2;

if(sel==0)
    out=in1;
if(sel==1)
    out=in2;

```

If the conditions contain binary expression then simple conditions are checked. Otherwise, no checking is done. For example:

```
if(sel==2)
    out=in1;
if(sel>1)
    out=in2; //The conditions can be true simultaneously
```

If the left expression of binary expressions is complex or right expression of binary expression is not constant, no checking is done. For example:

```
if(sel==0)
    out = in1;
if(sel>a)
    out = in2; //No Condition checking as right
               //expression is not constant
if(sel+sel1>a)
    out = in3; //No Condition checking as left expression
               //is not simple expression
```

By default, the *W415a* Tag does not report violation if the RHS value in an assignment is same as the RHS value in the previous assignment. Set the *reportsimilarassgn* Tag parameter to *yes* to flag such cases. For example, the *W415a* Tag will not report violation in the following case with the *reportsimilarassgn* Tag parameter is set to *no*:

```
if(sel)
    q =a+b;
if(sel0)
    q = b+a;
if(sel1)
    q = 1;
if(sel2)
    q = 1;
```

NOTE: Only simple expressions are checked for checking similar assignments. Complex expressions such as *a+b+c* and *a+c+b* are not checked.

By default, the *W415a* Tag reports violation for a signal assigned multiple times on the same line inside a for-loop irrespective of the expression on

RHS of the assignment. Set the [ignore_multi_assign_in_forloop](#) parameter to Yes to skip Tag checking inside a for-loop for multiple assignments on the same line.

In the following examples, the W415a Tag reports violations for signal *a*. When the [ignore_multi_assign_in_forloop](#) parameter is set to Yes, no violations are reported.

Example 1

```
reg a, b, c;
  for(i = 0; i < 2; i = i + 1)
    begin
      a = a + 1;
    end
```

Example 2

```
reg a, b, c;
  for(i = 0; i < 2; i = i + 1)
    begin
      a = b + c;
    end
```

By default the [waiver_compat](#) parameter is set to no. If you set the value of this parameter to yes or <Tag-name>, it ensures that the Tag does not generate the line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.

By default the value of [ignore_bitwiseor_assignment](#) parameter is set to no and the Tag reports a violation for a bitwise or assignment inside the for loop that is assigned multiple times on the same line. Set the value of the parameter to yes to not to report violation for such cases.

For example:

```
reg a, b, c;
  for(i = 0; i < 2; i = i + 1)
    begin
      a |= a + 1;
    end
```

In the above snippet, by default the W145a Tag reports violation for signal

a assigned inside the `for` loop. But when the [ignore_bitwiseor_assignment](#) parameter is set to `yes`, no violation is reported for the `bitwise` or `assignment` inside the `for` loop.

The *W415a* Tag checks a vector when it is initialized with an initialization value and the vector is assigned again with another value within a "for" loop.

For example, consider the following snippet:

```
module test (a, b, clk, y);
input [3:0] a, b;
input clk;
output [3:0] y;
reg [3:0] y;
integer i;

always @(clk) begin
    y <= 3'b000;
    for (i=0; i<=3; i=i+1)
        y[i] <= a[i] & b[i]; // warning
end
endmodule
```

For the above example, the Tag reports the following violation message:

Signal y[0:3] is being assigned multiple times (previous assignment at line 9) in same always block [Hierarchy: ':test']

Message Details

Message 1

The following message appears at the location where a signal *<sig-name>* is assigned another value after it has already been assigned a value at line number *<line-num>* in the same always construct:

Signal *<sig-name>* is being assigned multiple times (previous assignment at line "*<line-num>*") in same always block [Hierarchy: '*<hier-path>*']

Message 2

The following message appears at the location where a signal *<sig-name>* is assigned multiple values in the same for-loop and in the same always construct:

Signal *<sig-name>* is being assigned multiple times (assignment within same for-loop) in same always block [Hierarchy: '*<hier-path>*']

Tag Severity

Warning

Suggested Fix

Aside from overwriting an initial value, multiple overwrites of a value within a block is at least confusing. In some cases, it may indicate an error. Therefore, it is recommended to rewrite the code to remove multiple overwrites.

Examples

Consider the following examples where signal *out* is being assigned multiple times in the same `always` construct:

Example of Simple Multiple Assignments

Consider the following example:

```
always @(in1 or in2)
begin
    out = in1;
    out = in2;
end
```

Here, the previous assignment to signal *out* is overwritten by the second assignment. For this example, VC SpyGlass Lint generates the following message:

Signal 'out' is being assigned multiple times in same always block

Example of Conditional Multiple Assignments

Consider the following example:

```

always @(in1 or in2 or a or b)
begin
  if (a)
    out = in1;
  if (b)
    out = in2;
end

```

Here, the previous assignment to signal `out` is overwritten by the second assignment. Also, there is potential case of overwriting to signal `out`. For this example, VC SpyGlass Lint generates the following message:

Signal 'out' is being assigned multiple times in same always block

Example of Multiple Assignments at Different Level

Consider the following example:

```

...
a = b;
if (C1)
begin
  a = b1; //first assignment
  if (C12)
    a = b12; //second assignment
end
...

```

Since the assignments are at different level in nested conditional blocks, VC SpyGlass Lint does not flag a violation.

Example of Multiple Assignments in Different Branches of case Construct

Consider the following example:

```

always @(in1 or in2)
begin
  case (sel)
    1'b0 : out = in1;
    1'b1 : out = in2;
    default : out = 1'b0;
  end
end

```

Since, there is no overwriting to signal `out`, VC SpyGlass Lint does not flag a violation.

Example of Multiple Assignments in Different Branches of If Statement

Consider the following example:

```
always @(in1 or in2 or c)
begin
    if (c)
        out = in1;
    else
        out = in1;
end
```

Since, there is no overwriting to signal `out`, VC SpyGlass Lint does not flag a violation.

Example of Multiple Assignments in Same For-Loop and Same Always construct

Consider the following example:

```
reg [0:5] out2;
always @(in1)
begin
    for(i = 0; i < 2; i = i + 1)
    begin
        out2[3:4] <= in1[i];
    end
end
```

In the above example, the W415a Tag reports a violation for the signal, `out2 [3 : 4]`, because it is assigned multiple times within the same for loop and same always construct.

W416

Width of return type and return value of a function should be same (Verilog). Reports functions in which the range of the return type and return value of a function are not same (VHDL)

When to Use

Use this Tag to identify the functions in which the range of the return type and return value of a function is not the same.

Description

Verilog

The *W416* tag reports functions in which the width of the return type is not same as the function return value. This tag reports a violation for the following cases:

- There is a width mismatch between the return type and the return value of the function.
- There are order mismatches between the declared ranges in the return type and the return value - (reversed ranges).
- The tag will check for the order mismatches with the following limitations:
 - ☐ There should not be a width mismatch between the return type and the return value of the function.
 - ☐ The number of ranges in the return type and the return value should be same.
 - ☐ LHS and RHS of ranges should be static.

VHDL

The *W416* tag reports functions in which the range of the return type is not same as the function return value. This Tag reports violation for the following cases:

- The left values and right values of the ranges of the return type and return value are different even if the widths are same.
- The range types of the return type and return value are different (to in one range and downto in the other).

Language

Verilog, VHDL

Parameters

Verilog

- [nocheckoverflow](#): Default value is `no`. Set the value of the parameter to `yes` or tag name to calculate the width as per the LRM. See [Width Calculation for Verilog](#) and [Width Calculation for VHDL](#) to know more about calculating width.
- [check_static_value](#): Default value is `no`. Set the value of the parameter to `yes` to report violation for cases with width mismatch, involving static expressions and non-static expressions having a static part.
Other possible values:
 - ☐ `only_const` - Reports a violation for cases involving static expressions.
 - ☐ `only_expr` - Reports a violation for non-static expressions having a static part.
 - ☐ `only_static` - Does not report a violation for expression that does not have any static part.
 - ☐ `parameter` - Reports a violation when the expression is a parameter.
 - ☐ `sized_based_number` - Reports a violation when the expression is a sized based number.
 - ☐ `unsized_based_number` - Reports a violation when the expression is a unsized based number.
 - ☐ `constant` - Reports a violation when the expression is a constant.
- [disable_rtl_deadcode](#): Default value is `no`. Set the value of the parameter to `yes` to disable violations for disabled code in loops and conditional (if condition, ternary operator) statements.
- [enable_rtl_deadcode](#): The default value is `no`. Set this parameter to `yes` to report different violations for dead codes in the RTL resulting from `if/for/while` and ternary operators that are ignored when the [disable_rtl_deadcode](#) parameter is enabled.

- [*handle_zero_padding*](#): Default value is `no`. Set the value of the parameter to `yes` or tag name to perform leading zero expansion and truncation of RHS of an assignment.
- [*use_lrm_width*](#): Default value is `no`. Set this parameter to `yes` to consider the LRM width of integer constants, which is 32 bits.
- [*W416_vhdl_only*](#): Default value is `no`. Set this parameter to `yes` to enable the tag to use only the VHDL version and check VHDL parts of the design.
- [*report_violation_type*](#): Default value is `both` and in this case the tag reports a violation for any width mismatch. Set the value of the parameter to `truncation` to report a violation when the function return value width is more than function return type width. Set the value of the parameter to `extension` to report a violation when the function return value width is less than function return type width.

Constraints

None

Messages and Suggested Fix

Verilog

Message 1: Width mismatch

[WARNING] Return type width '<width>' is <less | greater> than return value width '<width>' in function 'function_name'.
[Hierarchy: '<hierarchy>']

Potential Issues

A violation is reported when the return type width is lesser or greater than the width of the return value of a function.

Consequences of Not Fixing

Not fixing the violation may lead to unexpected tag behavior.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the RTL line where the function value is returned.

Message 2: Range mismatch

[WARNING] Order mismatch between return type range(s)

'<range_list>' and return value ranges(s) '<range_list>' in function '<function_name>'. [Hierarchy: '<hierarchy>']

Potential Issues

A violation is reported when there is an order mismatch between the return type and the return value in a function.

Consequences of Not Fixing

The order mismatch between expected return value of a function and the actual return value of a function can lead to two different interpretations during verification and test development.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the RTL line where the function value is returned.

To fix the violation, it is recommended to follow consistent order declarations throughout the design. To fix this problem, match the order of return value with that of return type of the function.

VHDL

The following message appears at the location of the return statement of a function `<func-name>` where the range of the return type (`<rettype-range>`) is different from the range of the return value (`<retvalue-range>`):

[WARNING] Range Mismatch between return type (`<rettype-range>`) and return value (`<retvalue-range>`) in function '`<func-name>`'

Where `<rettype-range>` and `<retvalue-range>` are range specifications in `<num> to <num>` format or `<num> downto <num>` format.

Potential Issues

Violation may arise when there is a range mismatch between return type and return value in a function.

Consequences of Not Fixing

The range mismatch between expected return value of a function and the actual return value of a function can lead to two different interpretations during verification and test development.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location of the return statement of a function where there is mismatch in range of return type of the function and the range of return value of the function.

To fix the violation, it is recommended to follow consistent range declarations throughout the design. To fix this problem, match the range of return value with that of return type of the function.

Example Code and/or Schematic

Example 1

Consider the following example:

```
...
subtype S1 is MH (0 to 7);
subtype S2 is MH (1 to 8);
...
function foo (a: S1) return S1 is
variable EN: S2;
...
return EN;
...
```

In the above example, left values and right values of the range of the return type S1 (0,7) are different from those of the range of the return value S2 (1,8) while the actual width is same (8 bits). Therefore, the W416 Tag reports a violation.

Example 2

Consider the following example:

```
...
subtype S1 is MH (0 to 7);
subtype S2 is MH (7 downto 0);
...
function foo (a: S1) return S1 is
variable EN: S2;
...
return EN;
...
```

In the above example, the W416 Tag reports a violation as the range types

Lint Tags

of the return type and return value are different (to in one range and downto in the other).

Default Severity Label

Warning

Tag Group

Function-Subprogram

Reports and Related Files

None

W421

Reports “always” or “process” constructs that do not have an event control

When to Use

Use this Tag to identify `always` or `process` constructs that do not have an event control.

Description

Verilog

The *W421* Tag reports `always` constructs that neither have a sensitivity list nor an event control statement.

VHDL

The *W421* Tag reports `process` constructs that neither have a sensitivity list nor a wait statement.

Language

Verilog, VHDL

Parameters

None

Constraints

None

Messages and Suggested Fix

Verilog

The following message appears for an `always` construct that has neither a sensitivity list nor an event control statement.

[WARNING] No event control (@) in always block

Potential Issues

Violation may arise when an `always` block does not have an event control.

Consequences of Not Fixing

With no associated event statement, an `always` block remains active and no other block can start execution. This can result in the simulator hanging on the design.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the `always` block with no event control is declared.

Unless the `always` block is the very top process in the analysis and you are depending on an internal task or procedure to terminate simulation, specify an event control statement for the corresponding violating `always` block.

VHDL

The following message appears for a process construct that has neither a sensitivity list nor a wait statement.

[WARNING] No wait statement or sensitivity list in process

Potential Issues

Violation may arise when a process construct does not have a `wait` statement.

Consequences of Not Fixing

With no associated event/wait statement, a process construct is always active and no other block can start execution. This can result in the simulator hanging on the design.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the `process` block with no event control is declared.

Unless the `process` block is the very top process in the analysis and you are depending on an internal task or procedure to terminate simulation, specify a wait statement for corresponding violating process construct.

Example Code and/or Schematic

Consider the following example:

```
entity ent_test is
  port(a: in bit;
        b: in bit;
```

```
        z: out bit
    );
end ent_test;

architecture behav of ent_test is
begin
    process
    begin
        z <= a or b;
    end process;
end behav;
```

In the above example, the *W421* Tag reports a violation as the process statement is declared without an event control.

Default Severity Label

Warning

Tag Group

Event

Reports and Related Files

None

W422

Unsynthesizable block or process: event control has more than one clock

When to Use

Use this Tag to identify the potentially unsynthesizable block or process.

Description

The W422 Tag reports violation for always constructs or process constructs having event control descriptions with more than one clock.

For Verilog, the W422 Tag flags potentially un-synthesizable block as shown below:

```
always @(posedge clk1 or posedge clk2) ...
```

For VHDL designs, VC SpyGlass Lint recognizes the following types of constructs as clocks:

<code>clk'event and clk = '1'</code>
<code>not clk'stable and clk = '1'</code>
<code>wait until clk = '1'</code>
<code>wait until clk'event and clk = '1'</code>
<code>falling_edge(clk) or rising_edge(clk)</code>

Tag Exceptions

The W422 Tag reports violation for potentially un-synthesizable block. It is possible that the block is synthesizable by some synthesis tool.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

Verilog

The following message appears at the location of an event control description using more than one clock:

[WARNING] Block might be un-synthesizable by some tool: event control has more than one clock

Potential Issues

Violation may arise when an event control uses more than one clock.

Consequences of Not Fixing

The violating block may not be synthesizable by some synthesis tool as an event control has more than one clock.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the start line of the `always` block in which more than one clock event control is used.

If you want to switch on edges of both clock signals, precompute a combined clock signal and switch on the appropriate edge of that combined signal.

VHDL

The following message appears at the first line of a process construct that has two clocks `<clk1-name>` and `<clk2-name>`:

[WARNING] Multiple clock signals `<clk1-name>`, `<clk2-name>` used in the process may not be synthesizable

Potential Issues

Violation may arise when multiple clock signals are used in a process construct.

Consequences of Not Fixing

Many synthesis tools support more than one clock inside a process, but this is not considered good coding style because the trigger conditions for the

process as a whole can be confusing.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the start line of the process block in which more than one clock event control is used.

Unless you know for certain that your synthesis tool can handle multiple clocks in a process, break these cases into multiple processes, each with a single clock.

Example Code and/or Schematic

Verilog

The following Verilog code shows the example of an event control construct, which is not synthesizable:

```
module mod(in1, in2, clk1, clk2, out1);
  input in1, in2;
  input clk1, clk2;
  output out1;
  reg out1;
```

```
    always@(posedge clk1 or posedge clk2)
        out1 = in1 ^ in2;
```

```
endmodule
```

The W422 Tag reports a violation message for the above example.

VHDL

Consider the following example where two clocks clk1 and clk2 are used in the same process:

```
library IEEE;
use ieee.std_logic_1164.all;

entity multclk is
  port(
    d,clk1,clk2 : in std_logic;
    q1          : out std_logic
  );
```

```
end multclk;

architecture arc of multclk is

begin
  process(clk1, clk2)
  begin
    if (clk1'event and clk1 = '1') then
      q1 <= '0';
    end if;
    if (clk2'event and clk2 = '1') then
      q1 <= d;
    end if;
  end process;
end process;
```

For this example, VC SpyGlass Lint generates the following message:

Multiple clock signals clk1, clk2 used in the process may not be synthesizable

Default Severity Label

Warning

Tag Group

Lint_Clock

Reports and Related Files

No related reports or files.

W422L

Multiple clocks in the event control list of latch is not allowed

Language

Verilog

Description

The *W422L* Tag flags multiple clocks used in event control list of latch descriptions. For constructs, `always @ {*}` and `always_latch`, event control list is auto inferred.

Message Details

The following message appears at the location of the event control list of a latch description where more than one user-specified clock is used:

[WARNING] Multiple specified clocks in the sensitivity list is not allowed

Tag Severity

Warning, Info

Verilog Examples

Currently Unavailable

W423

A port with a range is redeclared with a different range

Language

Verilog

Description

The W423 Tag flags ports that are re-declared with a different range in the same module.

While such description is not necessarily an error, it may not be the design intent.

However, the tag does not report violation when width of the signal and redeclaration of signal is equal. Consider the following example:

```
output dout;  
wire [0:0] dout;
```

Here although dout is redeclared with different range but the width is same, that is it's a single bit.

Message Details

The following message appears at the location where a port *<port-name>* is re-declared with a different range:

Port '*<port-name>*' (Range *<previous-range>*) is redeclared with different range *<latest-range>*. [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the complete hierarchical name of the containing scope excluding subprograms.

This Tag also highlights previous declaration with different range.

Tag Severity

Warning

Suggested Fix

In some cases this may be a neat way of compactly managing a calculation, but it can be confusing. Recommend in general you avoid this

style. If necessary, declare and assign to a temporary bus of the appropriate width.

Examples

Example 1

In the following example, the input port `in1` is being re-declared as a wire with a different range:

```
input [15:0] in1;  
wire [20:0] in1;
```

While such description is not necessarily an error, it may not be the design intent. In the above example, the top five bits of wire `in1` will not be driven by the input port `in1`.

Example 2

Consider the following example:

```
output dout;  
wire [0:0] dout;
```

The tag doesn't report violation for the above example.

W424

Ensure that a function or a sub-program does not sets a global signal/variable

When to Use

Use this Tag to identify a function or a sub-program, which is writing to a signal/variable outside its scope.

Description

The W424 Tag reports violation for signals or variables that are not passed as parameters to a sub-program but are modified in the sub-program. The Tag reports a violation only once for each global variable in a function or a sub-program.

Tag Exceptions

In VHDL, using global signal/variable inside a pure function is not allowed. VC SpyGlass Lint generates syntax error, STX_472, in such cases.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

[*ignore_interface_locals*](#): default value is off. Set the value of the parameter to yes to enable the W424 tag to not report violations for functions inside an interface that are written to variables/signals in the interface scope.

Constraint(s)

None

Messages and Suggested Fix

Verilog

The following message appears at the location where a variable `<var-name>` that is not passed as a parameter, is modified in the function:

[WARNING] Function should not set a global variable: <var-name>

Potential Issues

Violation may arise when a function sets a global variable.

Consequences of Not Fixing

A global variable can potentially be modified from any part of the design. Therefore, it has unlimited potential for creating mutual dependencies, which increases complexity. This can lead to unexpected design behavior when the code is changed.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where a global variable is set inside the function body. This means this variable is not a function argument and is defined outside the function body, but it is set inside the function.

To fix the violation, it is advisable to only read from and write to internal variables.

VHDL

Message 1

The following message appears at the location where a signal <sig-name> that is not passed as a parameter, is modified in the sub-program <subp_name>:

[WARNING] Signal '<sig-name>' being modified in sub-program '<subp-name>' is outside its scope

Potential Issues

Violation may arise when a signal modified in a sub-program is outside the scope of the sub-program.

Consequences of Not Fixing

A global signal can potentially be modified from any part of the design. Therefore, it has unlimited potential for creating mutual dependencies, which increases complexity. This can lead to unexpected design behavior when the code is changed.

How to Debug and Fix

Double-click the violation message. The HDL window, highlights the location of the declaration of the signal (global signal), which is modified

inside the violating sub-program.

To fix the violation, it is advisable to only read from and write to internal variables.

Message 2

The following message appears at the location where a variable `<var-name>` that is not passed as a parameter, is modified in the sub-program `<subp_name>`:

[WARNING] Variable '`<var-name>`' being modified in sub-program '`<subp-name>`' is outside its scope

Potential Issues

Violation may arise when a variable modified in a sub-program is outside the scope of the sub-program.

Consequences of Not Fixing

A global variable can potentially be modified from any part of the design. Therefore, it has unlimited potential for creating mutual dependencies, which increases complexity. This can lead to unexpected design behavior when the code is changed.

How to Debug and Fix

Double-click the violation message. The HDL window, highlights the location of the declaration of the variable (global variable), which is modified inside the function specified in the violation message.

To fix the violation, it is advisable to only read from and write to internal variables.

Example Code and/or Schematic

Consider the following example where the function `factorial` sets the values of a global variable `g1`:

```
module func(in1, out1, out2, rst, clk);
    input clk, rst;
    input [3:0] in1;
    output [31:0] out1, out2;

    reg [31:0] out1, out2;
    integer g1;
```


Lint Tags

```
always @(in1 or rst)
  if (rst)
    out1 = 1;
  else
    out1 = factorial(in1);

always @(posedge clk)
  out2 <= out1;

function [31:0] factorial;
  input [3:0] operand;
  reg [4:0] index;
  begin
    factorial = operand ? 1 : 0;
    g1 = 1;
    for (index=2; index<=15; index=index+1)
      begin
        if (index <= operand)
          factorial = index * factorial;
        end
      end
    end
  endfunction
endmodule
```

For this example, VC SpyGlass Lint generates the following violation message:

Function should not set a global variable: g1

Default Severity Label

Warning

Tag Group

Function-task

Reports and Related Files

No related reports or files.

W425

Ensure that a function or a sub-program does not uses a global signal/variable

When to Use

Use this Tag to identify a function or a subprogram, which is reading from a signal/variable outside its scope.

Description

The W425 Tag reports violation for signals or variable that are not passed as parameters to a function or sub-program but are read in the functions or sub-programs. The Tag reports a violation only once for each global variable in a function or a sub-program.

Tag Exceptions

In VHDL, using global signal/variable inside a pure function is not allowed. VC SpyGlass Lint generates syntax error, STX_472, in such cases.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

[*ignore_interface_locals*](#): default value is off. Set the value of the parameter to yes to enable the W424 tag to not report violations for functions inside an interface that are written to variables/signals in the interface scope.

Constraint(s)

None

Messages and Suggested Fix

Verilog

The following message appears at the location where a variable `<var-`

name> that is not passed as a parameter to a function, is read in the function:

[WARNING] Function uses a global variable: <var-name>

Potential Issues

Violation may arise when a function uses a global variable.

Consequences of Not Fixing

A global variable can potentially be modified from any part of the design. Therefore, it has unlimited potential for creating mutual dependencies, which increases complexity. This can lead to unexpected design behavior when the code is changed.

Also, usage of such functions under `always @*` creates simulation-synthesis mismatch because `always @*` is not sensitive to global variables used inside function.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where a global variable is read inside the function body. This means this variable is not a function argument and is defined outside the function body, but read inside the function.

To fix the violation, it is advisable to read only from the internal variables and task arguments.

VHDL

The following message appears at the location where a global variable named <*name*> that is not passed as a parameter to function <*func-name*>, is read in the sub-program:

[WARNING] Global shared variable '<name>' is read in function '<func-name>'

Potential Issues

Violation may arise when a global shared variable is read in a function.

Consequences of Not Fixing

A global variable can potentially be modified from any part of the design. Therefore, it has unlimited potential for creating mutual dependencies, which increases complexity. This can lead to unexpected design behavior when the code is changed.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where a global variable is read inside the function body. This means this variable is not a function argument and is defined outside the function body, but read inside the function.

To fix the violation, it is advisable to only read from and write to internal variables.

Example Code and/or Schematic

In the following example, function func1 directly reads signal in1:

```
module test (clk1, in1, in2, cntr, out1);
    input clk1, in1, in2;
    input [1:0] cntr;
    output out1;

    assign out1 = func1(cntr, in2, clk1);

    function func1;
        input clk, in_f;
        input [1:0] cntr_f;

        if (clk == 1'b0)
            func1 = in1 + cntr_f[1];
        else
            func1 = cntr_f[0];
        endfunction
    endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Function should not use a global variable: in1

Default Severity Label

Warning

Tag Group

Function-task

Lint Tags

Reports and Related Files

No related reports or files.

W426

Ensure that the task does not sets a global variable

When to Use

Use this Tag to identify the task, which is writing to a variable outside its scope.

Description

The W426 Tag reports violation for tasks that set global variables.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a variable `<var-name>` that is not passed as a parameter, is set in the task:

[WARNING] Global variable '<var-name>' should not be 'set' in task

Potential Issues

Violation may arise when a global variable is set in a task.

Consequences of Not Fixing

A global variable can potentially be modified from any part of the design. Therefore, it has unlimited potential for creating mutual dependencies, which increases complexity. This can lead to unexpected design behavior when the code is changed.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where a global variable is set inside the task. This means this variable is not a task argument and is defined outside the task, but it is set inside the task.

To fix the violation, it is advisable to only read from and write to internal variables.

Example Code and/or Schematic

In the following example, the global signal `b` is set in task `proc_a`:

```
module test(in, out, clk);
    input [3:0] in;
    input clk;
    output [3:0] out;

    reg [3:0] out, a, b, c;

    always @(posedge clk)
        begin
            a = in - 1;
            proc_a(a, c);
            out = a;
        end

    task proc_a;
        input [3:0] a;
        output [3:0] c;
        reg [3:0] c;

        begin
            b = a + 1;
            if (a==0)
                disable proc_a;
            c = b + 2;
        end
    endtask
endmodule
```

For this example, VC SpyGlass Lint generates the following message:
global variable 'b' should not be 'set' in task

Default Severity Label

Warning

Tag Group

Function-task

Reports and Related Files

No related reports or files.

W427

Ensure that a task does not uses a global variable

When to Use

Use this Tag to identify a task, which is reading a variable outside its scope.

Description

The W427 Tag reports violation for those variables that are not passed as parameters to a task but are read in the task.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a variable `<var-name>` that is not passed as a parameter to a task, is read in the task:

[WARNING] Global variable '`<var-name>`' should not be 'read' in task

Potential Issues

Violation may arise when a global variable is read in a task.

Consequences of Not Fixing

A global variable can potentially be modified from any part of the design. Therefore, it has unlimited potential for creating mutual dependencies, which increases complexity. This can lead to unexpected design behavior when the code is changed.

Also, usage of such tasks under `always @*` create simulation-synthesis mismatch because `always @*` is not sensitive to global variables used inside tasks.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where a global variable is read inside the task. This means this variable is not an argument to the task and is defined outside the task, but it is read inside the task.

To fix the violation, it is advisable to only read from and write to internal variables.

Example Code and/or Schematic

In the following example, the global signal `b` is read in task `proc_a`:

```
module test(in, out, clk);
    input [3:0] in;
    input clk;
    output [3:0] out;

    reg [3:0] out, a, b, c;

    always @(posedge clk)
    begin
        a = in - 1;
        proc_a(a, c);
        out = a;
    end

    task proc_a;
        input [3:0] a;
        output [3:0] c;
        reg [3:0] c;

        begin
            b = a + 1;
            if (a==0)
                disable proc_a;
```

Lint Tags

```
        c = b + 2;  
    end  
    endtask  
endmodule
```

For this example, VC SpyGlass Lint generates the following message:
Global variable 'b' should not be 'read' in task

Default Severity Label

Warning

Tag Group

Function-task

Reports and Related Files

No related reports or files.

W428

Ensure that a task is not called inside a combinational block

When to Use

Use this Tag to identify the tasks that are called inside a combinational block.

Description

The W428 Tag reports violation for task calls in combinational `always` constructs unless the construct describes a latch.

As tasks may contain event controls, calling a task inside a combinational block may turn that block, unexpectedly, into a sequential block. Ensure that the correct behavior is being modeled.

When the parameter *fast* is set to `yes`, the RTL version of the Tag W428 is run. The `always` block inferring latch is considered as combinational block. The RTL version of this Tag reports inside the latch inferred `always` block, except `always_latch`.

Language

Verilog, VHDL

Default Weight

5

Parameter(s)

fast: The default value is `no`. Set the value of the parameter to `yes` to suppress synthesis of the source RTL description.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a task call is encountered in a combinational `always` construct:

[WARNING] Task called in a combinational block

Potential Issues

Violation may arise when a task is called in a combinational block.

Consequences of Not Fixing

Since tasks may contain event controls, a task called inside a combinational `always` construct may turn that construct, unexpectedly, into a sequential `always` construct.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where a task call has been made inside a combinational `always` construct. User can check the nature of the `always` block using one of the following ways:

- Slightly scrolling up the HDL window, till he reaches the `always` block.
- If the `always` block is a very long block, user might view the HDL in an editor, and, search backwards for the nearest `always` block.

To fix the violation, it is advisable to use functions, rather than tasks, inside combinational blocks.

Example Code and/or Schematic

Consider the following example:

```
module test(in, out, clk, reset);
    input  in;
    input clk, reset;
    output out;
    reg    out;

    always@(posedge clk)
    begin
        proc_d(in);// No Violation, Task called from sequential
    block
    end

    always@(clk)
    begin
        proc_d(in); // Violation, Task called from
    combinational block
    end
endmodule
```

```
end

task proc_d;
input  in;
begin
end
endtask

endmodule
```

In the above example, the W428 Tag reports a violation as a task is called from a combinational `always` block, when the *fast* parameter is not set.

Default Severity Label

Warning

Reports and Related Files

No related reports or files.

W429

Task called in a sequential block

Language

Verilog

Description

The W429 Tag flags task calls in sequential `always` constructs.

Since tasks may contain event controls, a task called inside a sequential `always` construct may unexpectedly create an implicit state machine (which may not be synthesizable if the task uses different edges or clocks from the block in which it is instantiated). Therefore while it is not an error to call a task inside a sequential `always` block, it should be handled with an extra caution.

Message Details

The following message appears at the location where a task call is encountered in a sequential `always` construct:

```
Task called in a sequential block
```

Tag Severity

Warning

Suggested Fix

Nothing to fix if this is done right, but does suggest need for careful review.

Examples

Consider the following example where a sequential `always` construct has a task call `proc_a`:

```
module test(in, out, clk);  
    input [3:0] in;  
    input clk;  
    output [3:0] out;  
  
    always @(*)  
        proc_a(in, out, clk);  
endmodule
```

```
reg [3:0] out;

reg [3:0] r1, r2, r3;

always @(posedge clk)
begin
    r1 = in - 1;
    proc_a(r1, r3);
    out = r3;
end

task proc_a;
// <task-body>
endtask
endmodule
```

For this example, VC SpyGlass Lint generates the W429 Tag message.

W43

Reports unsynthesizable wait statements

When to Use

Use this Tag to identify the wait statements that are not synthesizable.

Description

The W43 Tag reports violation for wait statements used in the design.

Following forms of wait statement may not be synthesizable by some synthesis tools:

```
wait;
wait on <sensitivity list>;
wait for <time-expression>;
```

The wait statement of the form, `wait until <Boolean-expression>`, can be used to infer a clock and hence, it is synthesizable.

The W43 Tag checks only for unsynthesizable wait statements.

Language

VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a wait statement is encountered:

[WARNING] <stmt> statement may not be synthesizable

Where, `<stmt>` refers to a form of wait statement. It can have values such as `Timeout` expression in wait or `Sensitivity` list in wait.

Potential Issues

Violation may arise when an unsynthesizable wait statement is encountered in the design.

Consequences of Not Fixing

Some synthesis tools may not be capable of translating wait statements into efficient logic.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the wait statement is used.

No fix is required in testbenches or simulation models. In code targeted for synthesis, recode to meet synthesizability guidelines.

Example Code and/or Schematic

Consider the following example in which a wait statement is used in a while loop:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity test2 is
port(
  in1: in std_logic_vector(2 downto 0);
  out1: out std_logic_vector(2 downto 0)
);
end test2;

architecture behav of test2 is
signal sig1: std_logic_vector(2 downto 0);
begin
  process
  begin
    while(sig1 = "111") loop
      out1 <= in1;
```

Lint Tags

```
        sig1 <= "10X";  
        wait on in1;  
    end loop;  
    end process;  
end behav;
```

VC SpyGlass Lint flags the following violation for the above case:
Sensitivity list in wait statement may not be synthesizable

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

No related reports or files.

W430

The "initial" statement is not synthesizable

When to Use

Use this Tag to identify the unsynthesizable `initial` statements.

Description

The W430 Tag reports violation for `initial` constructs used in the design.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where an `initial` construct is encountered:

[WARNING] Initial statement is not synthesizable

Potential Issues

Violation may arise when an `initial` construct is used in the design.

Consequences of Not Fixing

The `initial` constructs have no physical equivalent. Therefore, such constructs are unsynthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where offending initial construct are used. This is un-synthesizable by

several commercially available synthesis tools.

No fix required if the statement is used inside a testbench. In general you should not use initial blocks in code targeted for synthesis. Use physically realizable reset logic to initialize code instead. If you must include an initial block in synthesizable code, bracket it in `translate_off` and `translate_on` pragmas to disable the code in synthesis.

Example Code and/or Schematic

Consider the following example:

```
module W430_mod1(in1,clk,out1,out2);  
    input in1,clk;  
    output reg out1,out2;  
  
    initial  
    begin  
        out1 = 1'b0;  
        out2 = 1'b0;  
    end  
  
endmodule
```

In the above example, the W430 Tag reports a violation as an initial statement is used in the design.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

No related reports or files.

W433

More than one top-level design unit

Language

Verilog

Description

The W433 Tag flags multiple top-level modules in the design.

Designs containing more than one top-level module may be a mistake. This can occur due to the following reasons:

- The design is being analyzed together with library elements which are referenced directly and all the library elements are not used in the design. As a result, all the unused elements become top-level units.
- The design is missing some intermediate-level components, due to which the lower-level components in the hierarchy do not get linked. As a result, the lower level components become top-level units.
- The design contains certain modules that are specified as stop-levels but no top-level is specified. As a result, units appearing under those stop modules and not elsewhere in the design become top-level units.

Message Details

The following message appears at the first line of the first top-level module of a design that has more than one top-level module:

```
Design contains more than one top level module
```

Tag Severity

Info

W438

Ensure that a tristate is not used below top-level of design

When to Use

Use this Tag to identify the tristate descriptions that are used below top-level of the design.

Description

The W438 Tag flags tristate descriptions that are not at the top-level of the design. Tristate busing should be restricted to the top-level of the design.

In most of the current SoC designs, tristate signals are not widely used. If tristate signals are used, their usage is usually for muxing of the top-level buses or for some specialized fabric. Such fabrics are usually implemented at the top-level of SoC, and most of the designers prefer to manage the hierarchy accordingly for bus-holder cells, which in some technologies require a weak resistor. Bus holder cells ensure that the tristate bus does not float for a long time.

Language

Verilog

Default Weight

5

Parameter(s)

- **fast**: The default value is no. Set the value of the parameter to yes to suppress synthesis of the source RTL description.
- **CHECK_Z_ASSIGN**: By default, this parameter is set to 0. This indicates that the tag will not report violations on Z assignment cases. If you set this parameter to 1, the tag will report violations on Z assignment cases.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a tristate net `<sig-`

name> is used at a level other than the top-level:

[WARNING] Tristate function '<sig-name>' used below top level of design

Potential Issues

Violation may arise when a tristate function is used below top level of design.

Consequences of Not Fixing

For local sub-modules and lower level hierarchy blocks, use of tristates should be avoided for testability, timing, and reliability reasons.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the tristate value, *z*, is used in the design.

To fix the violation, use muxing at lower levels, rather than tristates.

Example Code and/or Schematic

Consider the following example where tristate descriptions are not at the top-level:

```
module test(a1, b1, x, y, ena, out1, clk);
    input [3:0] x, y;
    input clk;
    input [1:0] ena;
    output [3:0] a1, b1, out1;

    reg [3:0] out1;

    always @(posedge clk)
        begin
            out1 <= x | y;
        end

    statel ONE (a1, b1, x, y, ena);
endmodule

module statel(a, b, x, y, ena);
    input [1:0] ena;
```

Lint Tags

```
input [3:0] x, y;
output [3:0] a, b;

assign a = (ena == 2'b01) ? x : 4'bz;
assign b = (ena == 2'b10) ? 4'bz : y;
endmodule
```

For this example, VC SpyGlass Lint generates the following violation messages:

Tristate function 'a[3:0]' used below top level of design

Tristate function 'b[3:0]' used below top level of design

Default Severity Label

Warning

Tag Group

Lint_Tristate

Tag Severity

Warning

W442a

Ensure that for unsynthesizable reset sequence, first statement in the block must be an if statement

When to Use

Use this Tag to identify the asynchronous `reset` sequence where the first statement is not an `if` statement.

Description

The W442a Tag reports violation for asynchronous reset sequences where the first statement is not an `if` statement.

The first statement after the sensitivity list on the `always` construct must be an `if` statement using the reset signal.

Tag Exceptions

This is not a requirement inside testbenches or other simulation only code.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location of an `always` construct where the first statement is not an `if` statement using a reset signal:

[ERROR] Asynchronous reset/set always block has missing 'if' statement at the top level

Potential Issues

Violation may arise when an asynchronous reset/set always block has missing `if` statement at the top level.

Consequences of Not Fixing

In general, synthesis tools expect that the first statement inside an asynchronously reset block is an `if` statement. Else, the RTL may become unsynthesizable. Therefore, if code is meant for synthesis, this style should not be used.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the start line of the `always` block in which the first statement is not an `if` statement.

To fix the violation, reorganize the block into one or more blocks, such that any asynchronous reset block meets the requirement.

Example Code and/or Schematic

Consider the following example where the first statement in the `always` construct describing an asynchronous reset sequence, is not an `if` statement:

```
module DFF(D, clk, R, Q);
    output Q;
    input D, clk, R;
    reg Q;

    reg d1, d2;
    always@(posedge clk or posedge R)

        begin
            d2 = d1;
            if(R) Q = 0;
            else Q = D;
        end
endmodule
```

In the above example, the W442a Tag reports a violation as the asynchronously reset/set `always` block has a missing `if` statement.

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

No related reports or files.

W442b

Ensure that for unsynthesizable reset sequence, reset condition is not too complex

When to Use

Use this Tag to identify unsynthesizable and complex reset sequence.

Description

The W442b Tag reports violation for complex reset sequences.

The Tag also reports violation for reset signal when it is compared either with any other signal or variable.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a complex reset sequence is encountered:

[ERROR] In asynchronous reset/set always block, comparison is being made to non-constant expression (<expr>) in reset/set condition

Potential Issues

Violation may arise when a reset signal is compared with any other signal or variable or a non-constant expression.

Consequences of Not Fixing

Only certain forms of asynchronous reset descriptions are recognized by

the synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the asynchronous reset signal is compared with a non-constant expression.

Scroll through the HDL to search for the value of the signal used in comparison of the asynchronous reset signal.

To fix the violation, precompute the reset condition and then use that value as the actual reset.

Example Code and/or Schematic

Example 1

Consider the following example:

```
module mod(in1, clk, reset, set, out1);
    input [1:0] in1;
    input clk, reset, set;
    output [1:0] out1;
    reg [1:0] out1;

    always @(posedge clk or posedge reset)
    begin
        if(reset == !set)
            out1 <= 2'b00;
        else
            out1 <= in1;
        end
    endmodule
```

In the above example, the W442b Tag reports a violation as an asynchronous reset signal is compared to a non-constant expression, `!set`, in an always block.'

Example 2

Consider the following example:

```
always @(posedge reset or negedge clk)
begin
```

Lint Tags

```
    if (reset != b)
        q = 1'b0;
    else
        q = d;
end
```

In the above example, the W442b Tag report violations as the asynchronous `reset` signal is compared with the other signal `b`.

To fix the violation, precompute the reset condition and then use the resultant value as the actual reset as shown below:

```
assign actual_reset = (reset != b);
always @(posedge actual_reset or negedge clk)
begin
    if (actual_reset)
        q = 1'b0;
    else
        q = d;
end
```

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

No related reports or files.

W442c

Ensure that the unsynthesizable reset sequence are modified only by ! or ~ in the if condition

When to Use

Use this Tag to identify the unsynthesizable reset sequence that are modified by anything other than ! or ~ in the `if` condition.

Description

The W442c Tag reports violation for reset sequences that are modified by operators other than logical inverse (!) and bit-wise inverse (~) operators.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a reset signal is being modified by an operator other than logical inverse (!) and bit-wise inverse (~) operators:

[ERROR] Asynchronous reset/set always block may have the reset/set condition only as a simple identifier or its negation (! or ~)

Potential Issues

Violation may arise when a reset signal is being modified by an operator other than logical inverse (!) and bit-wise inverse (~) operators.

Consequences of Not Fixing

You can modify the reset signals using only logical inverse (!) and bit-wise inverse (~) operators in the if statement in the reset sequences. Modification by all other types of operators is not synthesizable.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the asynchronous reset signal is modified by anything other than ! or ~. Scroll through the HDL for the usage of asynchronous reset signal.

To fix the violation, precompute the value you want to use in the condition before using it.

Example Code and/or Schematic

Consider the following example:

```
module test(reset,q);
output q;
input reset;
reg q,clk,d;

always @(posedge reset or negedge clk)
begin
    if (&reset)
        q = 0;
    else
        q = d;
    end
endmodule
```

In the above example, the W442c Tag reports a violation as an asynchronous reset/set always block may have the reset/set condition only as a simple identifier or its negation (! or ~).

To fix the above violation, precompute the value you want to use in the condition and then use that value as shown below

```
...
assign actual_reset = &reset;
always @(posedge actual_reset or negedge clk)
begin
```

```
        if (actual_reset)
            q = 1'b0;
        else
            q = d;
        end
    ...
```

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

No related reports and files.

W442f

Ensure that the unsynthesizable reset sequence is compared using only == and != binary operator in the if condition

When to Use

Use this Tag to identify the reset sequences where reset signal is being compared using an operator other than the binary equal (==) and not equals (!=) operator.

Description

The *W442f* Tag reports violation for reset sequences where reset signal is being compared using an operator other than the binary equal (==) and not equals (!=) operator.

Language

Verilog

Default Weight

5

Tag Exceptions

The *W442f* Tag does not report a violation for "===" or "!=" used inside if condition, because these operators are treated as "==" or "!=", respectively.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a reset sequence uses an operator other than the binary equal (==) or not equals (!=) operator in the `if` statement:

[ERROR] only '==' and '!=' binary operators are allowed in

validation of the asynchronous reset/set condition

Potential Issues

Violation may arise when a reset sequence uses an operator other than the binary equal (==) or not equals (!=) operator in the `if` statement.

Consequences of Not Fixing

Reset signal that are compared using an operator other than the binary equal (==) and not equals (!=) operator are not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the binary operator other than == or != is used in the asynchronous reset signal expression. Scroll through the HDL for the type of binary expression used inside the asynchronous reset expression. To fix the violation, either precompute the reset condition and use the resultant value as the actual reset or use a nested `if` condition.

Example Code and/or Schematic

Consider the following example:

```
module test(q,reset,set);
input reset,set;
output q;
reg q,clk;
always @(posedge clk or negedge reset)
begin
    if (reset & set)
        q = 0;
    end
endmodule
```

In the above example, the W442f Tag reports a violation as only '=' and '!=' binary operators are allowed in validation of the asynchronous reset/set condition.

To fix the violation, you can use any one of the following methods:

- Precompute the reset condition and use that value as the actual reset. For example:

Lint Tags

```
assign actual_reset = reset & set;
always @(posedge actual_reset or negedge clk)
begin
    if (actual_reset)
        q = 1'b0;
    else
        q = d;
end
```

- Use a nested if condition. For example:

```
always @(posedge reset or negedge clk)
begin
    if (reset )
        if ( set )
            q = 1'b0;
        else
            q = d;
    else
        q = d;
end
```

Default Severity Label

Error

Tag Group

Synthesis

Reports and Related Files

No related reports or files.

W443

'X' value used

Language

Verilog, VHDL

Description

The W443 Tag flags based numbers that contain the unknown value character (X).

The unknown value character (X) has no physical counterpart and may lead to a mismatch between pre- and post-synthesis simulation.

By default, the W443 Tag does not check the presence of X value in the `default` statement of the case construct in Verilog designs. Use the [strict](#) Tag parameter to check in the `default` statement also.

For Verilog, no Tag checking is done for unused macro definitions and unused parameters.

NOTE: *The W443 Tag ignores case select item value within case constructs.*

Message Details

Verilog

The following message appears at the location where an unknown value character (X) is encountered in a based number `<num>`:

Based number `<num>` contains an X - has no meaning in synthesis

VHDL

The following message appears at the location where a forcing unknown character (X) is encountered in a bit string:

'X' state used - has no meaning in synthesis

Tag Severity

Warning

Suggested Fix

Use of X states is often valuable in functional debug, so the best way to handle these cases is to bracket them in `translate_off`, `translate_on` pragmas, making them visible in simulation but invisible in synthesis. You should check carefully to ensure that implemented behavior is implicitly dependent on X behavior.

Examples

Verilog

Consider the following example:

```
module top(out, en1);
    input en1;
    output reg [1:0] out;

    always @ (en1)begin
        if (en1)
            out<=2'bxx;
        else
            out<=2'b01;
        end
    endmodule
```

In the above example, the *W443* Tag reports a violation because the based numbers that contain the unknown value character (X) are assigned to out in the module. The following message is reported by this example:

Based number 2'bxx contains an X - has no meaning in synthesis

VHDL

Consider the following example:

```
entity top is
port(
    en1: in std_logic;
    out1: out std_logic_vector(1 downto 0)
);
```

```
end top;

architecture behav of top is begin
    process (en1) begin
        if (en1 = '1') then
            out1<="XX";
        else
            out1<="01";
        end if;
    end process;
end behav;
```

In the above example, the *W443* Tag reports a violation because the based numbers that contain the unknown value character (X) are assigned to *out1* in the architecture behave of the entity *top*. The following message is reported by this example:

'X' state used - has no meaning in synthesis

W444

'Z' or '?' value used

Language

Verilog, VHDL

Description

The W444 Tag flags all occurrences of the high impedance character ('Z') in the design.

Using 'Z' in the design creates several issues, such as:

- If 'Z' is used in an assignment statement, tristates are inferred
- If 'Z' is used in a comparison expression, the condition is always considered false in synthesis and may lead to a mismatch between pre- and post-synthesis simulation

The W444 Tag also flags the usage of '?', which is Verilog HDL alternative for the z character.

By default, the W444 Tag does not check for the presence of 'Z' value in the `default` statement of the `case` construct in Verilog designs. Use the [strict](#) Tag parameter to check the `default` statement also.

For Verilog, no Tag checking is done for unused macro definitions and unused parameters.

NOTE: *The W444 Tag ignores case select item value within case constructs.*

Message Details

Verilog

The following message appears at the location where a high impedance character ('Z') or '?' is encountered in a tristate value `<value>`:

Tristate value `<value>` specified

VHDL

The following message appears at the location where a high impedance character ('Z') is encountered in a bit string:

Tristate value specified

Tag Severity

Warning

Suggested Fix

For assignments, confirm that you intended to infer a tristate. Avoid using 'Z' values in comparisons, except for simulations tests, which you should bracket in `translate_off`, `translate_on` pragmas to avoid inferring spurious logic.

Examples

Verilog

Consider the following example:

```
module top(out, in1, en1);
    input in1;
    input [1:0] en1;
    output out;
    reg out;

    always @ (en1 or in1)
        begin
            if (en1 == 2'b1z)
                out <= in1;
            else
                out <= 1'b?;
            end
        endmodule
```

In the above example, the *W444* Tag reports two violations because of the high impedance characters ('Z') and ('?') in the design.

The following messages are reported by this example:

Tristate value '2'b1z' specified

Tristate value '1'b?' specified

VHDL

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

entity test1 is
port(
    clk: in std_logic;
    out1: out std_logic_vector(2 downto 0)
);
end test1;

architecture behav of test1 is
begin
    process(clk)
    begin
        if (clk'event and clk = '1') then
            out1 <= "10Z";
        end if;
    end process;
end behav;
```

In the above example, the *W444* Tag reports a violation because of the high impedance character ('Z') in the design. The following message is reported by this example:

Tristate value specified

W448

Reset/set is used both synchronously and asynchronously

Language

Verilog, VHDL

Description

The *W448* Tag reports signals that are used both as synchronous reset/set and asynchronous reset in a design.

Using a signal both as a synchronous and asynchronous reset/set can be confusing and difficult to debug. An exception to this is the generation of an "asynchronous assert, synchronous deassert" reset, but such cases should be localized.

Synchronous reset/set function is triggered with respect to a clock edge/phase. The reset/set signal can transition well in advance meeting the required set up and hold times. However, the resetting effect of a synchronous reset/set is synchronized to the clock edge.

The asynchronous reset/set takes place immediately after the assertion of the reset/set signal. Here, no clock edge/phase is required for resetting function.

If a single signal is used both synchronously and asynchronously then the cycle partitioning intent should be reviewed. In such cases, the integrity of scan and capture operations may get hampered during testing phase. However, there are some rare cases when the requirement is to use a reset/set signal both synchronously and asynchronously.

Use the following option to generate the [W448_Report](#), which lists the reset nets that are used synchronously and asynchronously:

```
-report=W448_Report
```

NOTE: *In Lint, synchronous reset/set detection is heuristic and subject to false errors.*

Parameters

- [report_local_name](#): By default, the value of the parameter is set to yes. It will report non hierarchical reset signal name in violation message. Set the value of the parameter to no to report hierarchical signal name.

- [*traverse_fanin_for_src_net*](#): Default value is 0. Set this parameter to 1 to stop the traversal in fanin of synchronous reset to detect flattened source reset.

Design Impact

Functionality (bug escape) and testability

Message Details

Following message appears at the location where a reset signal `<rst-name>` is declared that is used both as a synchronous reset and asynchronous reset in the same design:

Reset '`<rst-name>`' used both synchronously and asynchronously

Tag Severity

Warning

Suggested Fix

Check each case to ensure you understand the implications of using both edge-based and state-based reset. Try to prefer usage based on just one style and localize variants as much as possible

Examples (Verilog)

Consider the following example where signal `rst` is used both as synchronous reset as well as asynchronous reset:

```
module test(in, out1, out2, clk, rst);
  input in;
  input clk, rst;
  output out1, out2;
  reg out1, out2;
  always @(posedge clk or posedge rst)
  begin
    if(rst) out1 <= 0;
    else out1 <= in;
  end
  always@(posedge clk)
```

```

begin
    if(rst) out2 <= 0;
    else out2 <= in;
end
endmodule

```

For this example, VC SpyGlass Lint generates the following message:
 Reset 'rst' is used as both synchronous and asynchronous reset
 The schematic for the above example is shown below:

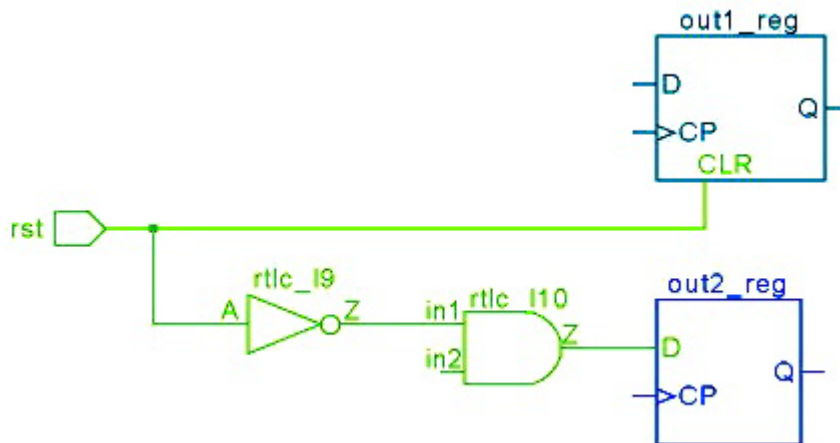


FIGURE 2. Incremental schematic

Examples (VHDL)

Consider the following example where the signal `reset` is used both as synchronous and asynchronous resets in the same architecture:

```

library ieee;
use ieee.std_logic_1164.all

entity test is
    port(d: in Bit;
          clk: in Bit;
          reset: in Bit;

```

Lint Tags

```

        q1: out Bit;
        q2: out Bit);
end test;

architecture structure of test is
begin
    P1: process(clk, reset)
    begin
        if (reset = '1') then
            q1 <= '0';
        elsif (clk'event and clk = '1') then
            q1 <= d;
        end if;
    end process P1;

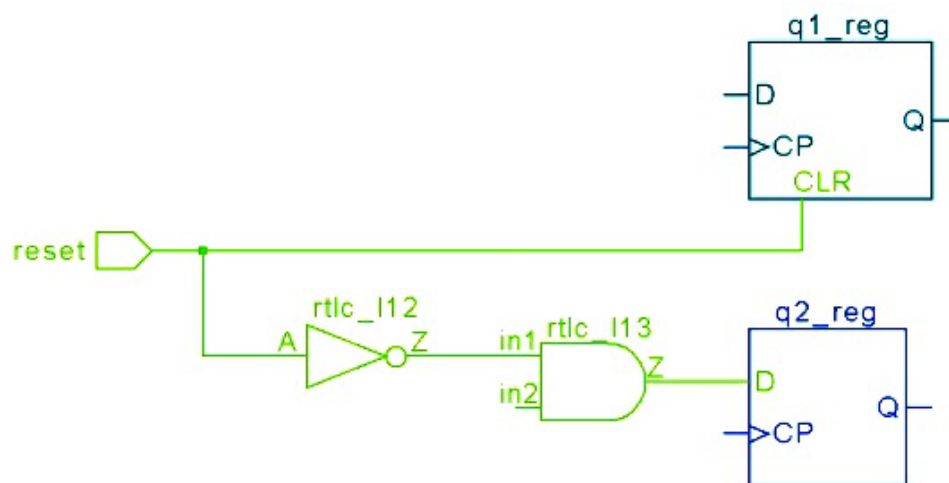
    P2: process(clk)
    begin
        if (clk'event and clk = '0') then
            if (reset = '1') then
                q2 <= '0';
            else
                q2 <= d;
            end if;
        end if;
    end process P2;
end structure;

```

For this example, VC SpyGlass Lint generates the following message:

Reset 'reset' is used as both synchronous and asynchronous reset

The schematic for the above example is shown below:

**FIGURE 3.** Incremental schematic

W453

Large case constructs should not be used

Language

Verilog

Description

The W453 Tag flags case constructs with large selector bit-width and large number of case clauses.

By default, the W453 Tag flags case constructs with the case select condition bit-width greater than 16 bits and the number of case clauses greater than 20. Use the [casesize](#) Tag parameter to change the default values.

The W453 Tag flags case constructs where both conditions are violated (select bit-width *and* number of clauses). Violation of one condition is not a Tag-violation.

When case select expression has larger width, number of case clauses becomes larger to cover all the cases. Also, this is a readability problem for user.

NOTE: *The W453 Tag supports generate-for and generate-case blocks.*

Message Details

The following message appears when a case construct with a large condition bit-width *<bit-width>* and more number of case clauses *<num-of-clauses>*, is encountered:

case construct is too wide with condition bit-width '*<bit-width>*' bits, and '*<num-of-clauses>*' clauses. [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the complete hierarchical path of the containing scope.

Severity

Warning

Suggested Fix

In the case of large case selector width, try to break the case statement into several case statements with small case selector width.

W456a

**A signal is included in the sensitivity list of a combinational always block but none of its bits are read in that block (Verilog)
A signal is included in the sensitivity list of a combinational process block but none of its bits are read in that block (VHDL)**

Language

Verilog, VHDL

Description

Verilog

The W456a Tag flags signals that are in the sensitivity list of a combinational always construct but are never read in the construct.

NOTE: *The W456a Tag is compatible with Design Compiler behavior. Therefore, this Tag will report violation only when none of the bits of signal are read in a combinational always block.*

While such signals do not affect the functionality, they slow down the design simulation as simulators will re-evaluate the block on each change in the variable. Also, the logic related to a signal in the sensitivity list may have been accidentally removed or not added, which results in wrong functionality.

By default, the W456a Tag does not report messages for constants present in sensitivity list. Set the [strict](#) tag parameter to flag constants also.

By default, the Tag checking is performed for all branches of *if-elseif-else* statement irrespective of the evaluated condition. Set the [disable_rtl_deadcode](#) parameter to *yes* to stop Tag checking for disabled *if-elseif-else* branch.

NOTE: *The W456a Tag supports generate-if and generate-for block.*

VHDL

The W456a Tag flags signals that are in the sensitivity list of a process construct but are never read in the construct.

While such signals do not affect the functionality, they slow down the design simulation as simulators will re-evaluate the block on each change in the variable. Also, the logic related to a signal in the sensitivity list may have been accidentally removed or not added, which results in wrong functionality.

NOTE: *The W456a Tag is compatible with Design Compiler behavior. Therefore, this Tag will report violation only when the whole bus is present in the sensitivity list of a process and none of its bits are read in that process.*

For sequential process constructs, signals other than the clock and the reset signals are not required in the sensitivity list even if they are read in that process construct and hence, are flagged.

By default, the W456a reports violations for all the records present in the process sensitivity list and whose elements are not read in the process block. Set the [checkfullrecord](#) parameter to yes to not to report violation for a record if any of its element bits is read.

Message Details

Verilog

The following message appears at the location of the sensitivity list of a combinational always construct where the signal or variable `<name>` is present in the sensitivity list of the always construct but is never read in that always construct:

The signal/variable '`<name>`' is not required in sensitivity list as none of its bits is read in always block [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the containing scope.

VHDL

The following message appears at the location of the sensitivity list of a combinational process construct where an object `<name>` of type `<type>` is present in the sensitivity list of the process construct but is never read in that process construct:

The `<type>` '`<name>`' is not required in sensitivity list as none of its bits is read in process [Hierarchy: '`<hier-path>`']

Where, `<type>` can be in port or signal and `<hier-path>` is the complete hierarchical path of the containing process.

The following message appears at the location of the sensitivity list of a sequential process construct where a signal or input port `<name>` is present but is not required in the sensitivity list of the process construct:

The `<type>` '`<name>`' is not required in sensitivity list of the

```
sequential process [Hierarchy: '<hier-path>']
```

Where, *<type>* can be signal or in port and *<hier-path>* is the complete hierarchical path of the containing scope.

Tag Severity

Warning

Suggested Fix

Remove these signals from the sensitivity list.

Examples

Verilog

Consider the following example:

```
module top( in1, in2, out1 );
    input in1, in2;
    output out1;
    reg out1;

    always @(in1 or in2)
    begin
        out1 = ~ in2;
    end

endmodule
```

In the above example, the *W456a* Tag reports a violation because the *in1* is included in the sensitivity list of the combinational *always* block but none of it's bits are read in that block. The following message is reported by this example:

Either all or some of the bits of signal/variable 'in1' are not required in sensitivity list, since they are not read in always block [Hierarchy: ':top:']

VHDL

Consider the following example:

```
library ieee;
```

```
use ieee.std_logic_1164.all

entity top is
port(in1: in bit;
      in2: in bit;
      out1: out bit
    );
end top;

architecture behav of top is
begin
    process(in1, in2)
    begin
        out1 <= not in2;
    end process;

end behav;
```

In the above example, the *W456a* Tag reports a violation because the *in1* is included in the sensitivity list of the combinational `process` block but none of its bits are read in that block. The following message is reported by this example:

The in port 'in1' may not be required in sensitivity list as not all of its bits are read in process [Hierarchy: ':top(behav)::']

W464

Ensure that the unrecognized synthesis directive is not used in the design

When to Use

Use this Tag to identify the unsupported pragma or synthesis directives used in the design.

Description

Verilog

For Verilog designs, the W464 Tag reports all pragma directives of specified prefix in the design.

By default, VC SpyGlass Lint flags pragma directives of prefix type `synthesis` if the value `synthesis` is also passed using the following VC SpyGlass Lint while invoking VC SpyGlass Lint:

```
-Xspyglass_pragma=<value>
```

Therefore, the following pragma directives are flagged:

```
//synthesis VC SpyGlass
//synthesis translate_off
//synthesis synthesis_on
```

To specify a different prefix type, first replace the default prefix value (`synthesis`) given in the W464 Tag registration of the Lint-Verilog Tagdeck file (`verilint.pl`) with your prefix type:

```
"prefix notmatch synthesis"
```

For example, to flag all pragma directives of prefix type `synopsys`, modify the value as follows:

```
"prefix notmatch synopsys"
```

Then, specify the VC SpyGlass Lint `-Xspyglass_pragma=<value>` command with the same value (`synopsys`) while invoking VC SpyGlass Lint.

VHDL

For VHDL designs, all Synopsys synthesis directives except the following are reported:

dc_script_begin	dc_script_end	translate_off
translate_on	synthesis_off	synthesis_on
resolution_method	built_in	map_to_entity
return_port_name	label	label_applies_to
map_to_operator	infer_mux	coverage_off
coverage_on		

Tag Exceptions

The W464 Tag does not report a violation, if you do not specify the `-Xspyglass_pragma=<value>` command with the same value that exists in the Lint-Verilog Tagdeck file.

Language

Verilog, VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

Verilog

The following message appears at the location where a pragma directive *<directive>* of the specified prefix type is encountered:

[WARNING] synthesis directive '*<directive>*' is not recognized

Potential Issues

Violation may arise when a pragma directive of the specified prefix type is encountered.

Consequences of Not Fixing

If `synthesis_off` and `synthesis_on` directives are used within an expression, they have the potential to create incorrect logic by the Synopsys tools.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the offending unrecognized synthesis directive are used.

To fix the violation, Synopsys recommends use of `translate_off` and `translate_on` directives against `synthesis_off` and `synthesis_on` directives.

VHDL

The following message appears at the location where an unsupported synthesis directive is encountered:

[WARNING] Synthesis directive '<directive>' is not recognized

Potential Issues

Violation may arise when an unsupported synthesis directive is encountered.

Consequences of Not Fixing

If `synthesis_off` and `synthesis_on` directives are used within an expression, they have the potential to create incorrect logic by the Synopsys tools.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the offending unrecognized synthesis directive is used.

To fix the violation, Synopsys recommends use of `translate_off` and `translate_on` directives against `synthesis_off` and `synthesis_on` directives.

Example Code and/or Schematic**Verilog**

Consider the following example:

```
module W464_mod1(inp, outp, sel);
    input [3:0] inp;
    input [1:0] sel;
    output outp;
    reg outp;
```

```

always @(sel)
begin
  //synthesis translate on           //VIOLATION
  case(sel)
    2'b00 : outp = inp[0];
    2'b01 : outp = inp[1];
    2'b10 : outp = inp[2];
    2'b11 : outp = inp[3];
  endcase
end

endmodule

```

In the above example, when you specify -Xspyglass_pragma=synthesis in the project file, the W464 Tag reports the following violation because an unrecognized synthesis directive is present in the design.

Synthesis directive 'synthesis' is not recognized

VHDL

Consider the following example:

```

library ieee;
use IEEE.std_logic_1164.all;

entity test1 is
port (i : in std_logic;
      o : out std_logic);
end;

-- synopsys synth_off
ARCHITECTURE arc_test1 OF test1 IS
BEGIN
  o <= i;
END arc_test1;

```

The W464 Tag reports the following violation message for the above example:

Synthesis directive 'synopsys synth_off' is not recognized

Lint Tags

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

No related reports and files.

W467

Use of don't-care except in case labels may lead to simulation/synthesis mismatch

When to Use

Use this Tag to identify the usage of don't care character in the design.

Description

The W467 Tag reports violation for based numbers that contain the don't care character.

Tag Exceptions

The W467 Tag does not flag a violation for a parameter, generic, or constants that are assigned a don't-care value and are used only as case-label.

Also, for Verilog, no Tag checking is done for unused macro definitions.

Language

Verilog, VHDL

Default Weight

5

Parameter(s)

- [*ignore_wildcard_operators*](#): Default value is no. Set the value of the parameter to yes to ignore violations given in RHS side of wild operators ==? and !=? as well as inside operator LHS 'inside' RHS.

Constraint(s)

None

Messages and Suggested Fix

Verilog

The following message appears at the location where a don't care character (?) is encountered in a based number `<num>`:

[WARNING] Based number `<num>` contains a don't care (?) - might

lead to simulation/synthesis mismatch

Potential Issues

Violation may arise when a based number contains a don't care value.

Consequences of Not Fixing

There is no physical counterpart for the don't-care value. In simulation, these values are typically mapped to 'Z' which causes a tristate to be inferred. However, this behavior should be avoided as it may result in inferring spurious logic.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#)

VHDL

The following message appears at the location where a don't care character (-) is encountered in a bit string:

[WARNING] Don't-care (-) used - might lead to simulation/synthesis mismatch

Potential Issues

Violation may arise when a don't care character (-) is encountered in a bit string.

Consequences of Not Fixing

There is no physical counterpart for the don't-care value. In simulation, these values are typically mapped to 'Z' which causes a tristate to be inferred. However, this behavior should be avoided as it may result in inferring spurious logic.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the don't care value, ?, is used in the design other than in the case label.

To fix the violation, test both 0 and 1 values in a comparison. Also, in an assignment, choose either 0 or 1.

Example Code and/or Schematic

Consider the following example:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity test1 is
port(
    in1: in std_logic_vector(2 downto 0);
    out1: out std_logic_vector(2 downto 0)
);
end test1;

architecture behav of test1 is
signal sig1: std_logic_vector(2 downto 0);
begin
    process(in1)
    begin
        sig1 <= "-0-";
        out1 <= sig1 and in1;
    end process;
end behav;
```

In the above example, the W467 Tag reports a violation as a don't-care character (-) is used in signal assignment.

Default Severity Label

Warning

Tag Group

Expression Tags

Reports and Related Files

No related reports or files.

W468

Index variable is too short

Language

Verilog

Description

The W468 Tag flags variables used as array index that are narrower than the array width.

When a variable used to index a bit-select or array has a width narrower than the width of the bus or the size of the array, some elements of the bus or array will not be accessible.

Message Details

The following message appears at the location where the value of variable or signal *<name>* used to index an array or a bus signal bit-select *<vector-name>* that is narrower than the bit-width of the array or bus, is encountered:

Variable/Signal '*<vector-name>*' is indexed by '*<name>*' which can not index the full range of this vector. [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the complete hierarchical name of the containing scope excluding subprograms.

Tag Severity

Warning

Suggested Fix

If possible, use an index large enough to span the full range of the vector.

If you intend to access only a lower subset of bits, make this explicit by using a concat of the correct width with upper bits set to zero, for example.

W474

Variable assigned but not deassigned

Language

Verilog

Description

The W474 Tag flags variables that are assigned but not deassigned.

This Tag flags an `assign` statement for register data types inside an `always` block, for which a `deassign` statement does not exist.

The procedural continuous assignment statement allows an expression to be driven continuously on registers or nets. It overrides the previous procedural assignments to a register.

The `deassign` statement ends a procedural assignment to register.

The value of register remains the same until the register is assigned a new value.

NOTE: *The W474 Tag is switched off by default. You can enable this Tag either by specifying the `configure_lint_tag -enable -tag W474` command or by setting the [verilint_compat](#) Tag parameter to yes.*

Message Details

The following message appears when the W474 Tag encounters a variable `<var-name>` that is assigned but not deassigned:

variable '`<var-name>`' assigned but not deassigned

Severity

Warning

Suggested Fix

Use `deassign` statement for the variable assigned.

Examples

Consider the following example where the variable 'q' is assigned but not

Lint Tags

deassigned:

```
module test5(inp, outp);  
    input  inp;  
    output outp;  
    reg outp;  
    reg q;  
    always  
        begin  
            if(inp)  
                deassign outp;  
            else if (!inp)  
                assign outp = 0;  
            else  
                assign outp = 1;  
        end  
  
    always  
        begin  
            if(outp)  
                assign q = 1;  
        end  
endmodule
```

For this example, VC SpyGlass Lint generates the following message:
variable 'q' assigned but not deassigned

W475

Variable deassigned but not assigned

Language

Verilog

Description

The W475 Tag flags the deassignment statement for a variable that does not have a corresponding `assign` statement.

If a variable is deassigned, it should have a corresponding `assign` in the `always` block.

The W475 Tag reports message for registers which are not assigned through procedural continuous assignment statement but are deassigned.

NOTE: *The W475 Tag is switched off by default. You can enable this Tag either by specifying the `configure_lint_tag -enable -tag W475` command or by setting the [verilint_compat](#) Tag parameter to `yes`.*

Message Details

The following message appears when a variable `<var-name>` is encountered that is deassigned without being assigned:

variable '`<var-name>`' deassigned but not assigned

Severity

Warning

Suggested Fix

Use `assign` statement for the variable deassigned.

W476

Variable forced but not released

Language

Verilog

Description

The W476 Tag flags variables that are forced but are not released.

The `force` construct is a form of procedural continuous assignment like `assign`. It applies to registers, nets, a constant bit-select of a vector, a part select of a vector net, or a concatenation. It does not apply on a memory word (arrays) or a bit select or part select of a vector register.

NOTE: *The W476 Tag is switched off by default. You can enable this Tag either by specifying the `configure_lint_tag -enable -tag W476` command or by setting the [verilint_compat](#) Tag parameter to `yes`.*

Message Details

The following message appears when a variable `<var-name>` that is forced but not released, is encountered:

variable '`<var-name>`' forced but not released

Severity

Warning

Suggested Fix

Use `release` statement for all the variables that are forced.

Examples

Consider the following examples, where

```
module test(a,b);  
  input [3:0] a;  
  output [3:0] b;  
  reg [3:0] b;
```

```
wire out1, out2 ,out3;

always @(a or b)
  if(a)
    force {out1, out2, out3} = {1'b1,1'b0,1'b1};
  else
    release out1;
endmodule
```

W477

Variable released but not forced

Language

Verilog

Description

The W477 Tag flags the `release` statement for variables for which there is no corresponding `force` statement.

The W477 Tag flags registers or nets that are not forced through procedural continuous assignment statement but are released.

NOTE: *The W477 Tag is switched off by default. You can enable this Tag either by specifying the `configureLintTag -enable -tag W477` command or by setting the [verilint_compat](#) Tag parameter to `yes`.*

Message Details

The following message appears at a location where a variable `<var-name>` is encountered that is released without being forced:
variable '`<var-name>`' released but not forced

Severity

Warning

Suggested Fix

Use `force` statement for all the variables that are released.

W479

Checks if loop step statement variables are not properly incremented or decremented

Language

Verilog

Description

The W479 Tag reports a violation message if the step variable is not incremented or decremented in the step part of the for statement.

For example, in the following case the Tag reports violation:

```
for ( i = 0 ; i < 10 ; i = 1 )
```

By default, the W479 Tag does not report a violation if the step variable is not present in the initialization or condition part. Set the [strict](#) parameter to `yes` to report violation in such a case.

For example, in the following case the Tag reports violation when the `strict` parameter is set:

```
for ( i = 0 ; i < 10 ; k = 1 )
```

Message Details

The following message appears at the location of a `for` construct where the variable `<var-name>` used in the step expression is not incremented or decremented:

step variable '`<var-name>`' not assigned properly

Tag Severity

Warning

Suggested Fix

Correct the `for` loop.

Examples

The following example violates the W479 Tag as the step expression (`i=2`)

Lint Tags

does not increment or decrement the value of step variable:

```
for ( i = 0 ; i < 10 ; i = 2 )  
    ...
```

W480

Ensure that the loop index is of integer type

When to Use

Use this Tag identify the `for` constructs in which the loop index is not of integer type.

Description

The W480 Tag reports violation for those `for` constructs in which the loop index variable evaluates to a non-integer.

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location of a `for` construct where the loop index variable `<var-name>` evaluates to a non-integer value:

[wARNING] Loop index '<var-name>' is not of type integer

Potential Issues

Violation may arise for a `for` construct where the loop index variable evaluates to a non-integer value.

Consequences of Not Fixing

You can also use `reg` as a loop index. However, using `reg` may cause logic to be inferred for that value, which is wasted if the value is required only as a loop index. This may or may not be an issue, depending on whether your

synthesis tools can trim the unused logic.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the line, where non integer signal is used as for loop index. Review the loop to see if really an integer value is required.

To fix the violation, prefer using an integer index wherever possible.

Example Code and/or Schematic

Example 1

Consider the following example:

```
time i1;
realtime i2;
integer i3;
for(i1=0 ; i1<5 ; i1 = i1+1 )
for(i2=0 ; i2<5 ; i2 = i2+1 )
for(i3=0 ; i3<5 ; i3 = i3+1 )
```

In the above example, the W480 Tag flags those `for` constructs in which the loop index is of type `time` and `realtime`.

Example 2

Consider the following example:

```
reg [2:0] q;
...
for (q=0 ; q<10; q=q+1)
```

In the above example, the W480 Tag reports a violation as the loop index variable, `q`, is not an integer.

Default Severity Label

Warning

Tag Group

Loop

Reports and Related Files

No related reports or files.

W481a

Ensure that a for loop uses the same step variable as used in the condition

When to Use

Use this Tag to identify the `for` constructs where the variable used in the step expression is not used in the condition expression.

Description

The W481a Tag reports violation for the `for` constructs where the variable used in the step expression is not used in the condition expression.

For example, it is meaningless to define a `for` construct as follows:

```
for (i = 0; j < 4; i = i + 1)
```

Here, the variable `i` used in the step expression is not used in the condition expression (`j < 4`).

Default Weight

5

Language

Verilog

Parameter(s)

strict: The default value is `no`. In this case, the Tag reports violation only if none of the step variable is present in the condition statement. Set the value of the parameter to `yes` to report violation for all the step variables that are not present in the condition statement.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location of a `for` construct where the variable `<var-name>` used in the step expression is not used in the condition expression:

[WARNING] Possibly unsynthesizable loop: step variable '<var-name>' is not used in condition

Potential Issues

Violation may arise when a variable used in the step expression in a `for` construct is not used in the condition expression

Consequences of Not Fixing

Synthesis tools may not unroll loops where different variables are used in step and condition expressions as the upper bound is not in general determinable. Also this is arguably poor coding even in logic not intended for simulation.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location of the "for loop" in which step variable is not same as that of the variable used in the condition.

To fix the violation, correct the `for` loop.

Example Code and/or Schematic

Example 1

Consider the following example:

```
for (a=0,b=0; a<10 && b<10 ; a++,b++)
```

In the above example, the W481a Tag does not report any violation, even if the value of the [strict](#) parameter is set to yes. This is because the variables, a and b, are used in step as well as condition expressions.

Example 2

Consider the following example:

```
for (a=0,b=0; a<10; a++,b++)
```

In the above example, by default, the W481a Tag does not report any violation.

However, if you set the value of the [strict](#) parameter to yes, the W481a Tag reports violation for variable b, as it is used in the expression variable but is not used in the condition expression.

Example 3

Consider the following example:

Lint Tags

```
for (a=0,b=0,c=0,d=0; a<10; b++,c++,d++)
```

In the above example, the W481a Tag reports a violation for variables b, c, and d, by default as well as when the [strict](#) parameter is set to yes. This is because the variables, b, c, and d, are used in the step expression but not in the condition expression.

Default Severity Label

Warning

Tag Group

Loop

Reports and Related Files

No related reports or files

W481b

Ensure that a for loop uses the same initialization variable as used in the condition

When to Use

Use this Tag to identify the `for` constructs where the variable used in the initialization expression is not used in the condition expression

Description

The W481b Tag reports violation for the `for` constructs where the variable used in the initialization expression is not same as the variable used in the step expression.

For example, it is meaningless to describe a `for` construct as follows:

```
for (i = 0; k < 10; k = k+1)
```

Here, the variable `i` used in the initialization expression is not same as the variable `k` used in the step expression.

In SystemVerilog, there can be multiple initialization/step expressions inside a single `for` construct. In such types of `for` constructs, violation will be reported if a variable used in any of the step expression is not present in any of the initialization expression.

Language

Verilog

Default Weight

5

Messages and Suggested Fix

The following message appears at the location of a `for` construct where the variable `<var1-name>` used in the initialization expression is not same as the variable `<var2-name>` used in the step expression:

[WARNING] Unsynthesizable loop: Init variable '`<var1-name>`' is not same as step variable '`<var2-name>`'

Potential Issues

Violation may arise when a variable used in the initialization expression in a `for` construct is not same as the variable used in the step expression

Consequences of Not Fixing

Synthesis tools cannot unroll loops where different variables are used in initialization and condition expressions as the upper bound is not in general determinable. Also this is arguably poor coding even in logic not intended for simulation.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location of the `for` loop in which initialization expression is not same as that of the variable used in the step expression.

To fix the violation, correct the `for` loop.

Example Code and/or Schematic

Example 1

Consider the following example:

```
for (i = 0, j=0; j < 4; k++, l++)
```

In the above example, the W481b Tag reports a violation for `k` and `l` as both the variables are used in the step expression but are not used in the initialization expression.

Example 2

Consider the following example:

```
for (i = 0, j=0; j < 4; k++, i++)
```

In the above examples, the W481b Tag reports a violation for `k` as it is used in the step expression but is not used in the initialization expression.

Example 3

Consider the following example:

Consider the following example where the W481a Tag does not report any violation:

```
for (i = 0, k=0; j < 4; k++, i++)
```

In the above example, the W481b Tag does not report any as the variables, `i` and `k`, are used in both initialization and step expressions.

Default Severity Label

Warning

Tag Group

Loop

Reports and Related Files

No related reports or files.

W484

Possible loss of carry or borrow due to addition or subtraction

Language

Verilog

Description

The *W484* Tag reports possible loss of carry or borrow bits for assignments that are using addition and subtraction arithmetic operators.

The Tag reports assignments where the result of an addition or subtraction operation is being assigned to a bus of the same width as the operands of the addition or subtraction operation. In such cases, the carry or borrow bit may be lost.

NOTE: *For new width related changes, refer to New Width Flow Application Note.*

By default, the Tag reports a violation for disabled code in loops and conditional (*if* condition, ternary operator) statements. Set the [disable_rtl_deadcode](#) parameter to *yes* to disable violations for disabled code in loops and conditional (*if* condition, ternary operator) statements.

Consider the following example:

```
if(0)
  a[3:0] = b[3:0] + c[3:0];
```

In the above example, the *W484* Tag does not report a violation when the *disable_rtl_deadcode* parameter is set to *yes*, because the *if* statement in the above example is a disabled conditional statement.

By default, the Tag does not report a violation for the following cases:

- Where same variable is assigned to itself:

```
reg [2:0] a;
a = a + 1;
```

- Where bit-width of RHS is less than or equal to the width of the LHS:

```
reg [2:0] a;
reg [2:0] b;
reg [1:0] c;
a = b + 1;
a = c + 1;
```

Set the [strict](#) parameter to report a violation for such cases.

By default, Tag checking is disabled for RHS expressions that are static or for RHS expressions that are non-static but contain a static part.

Consider the following example:

```
wire [2:0] a, b, c;
assign a = 3'b101 + 3'b110; //RHS is static
assign a = b + c + 1; //RHS is non-static with static part
assign a = b + 5; //RHS is non-static with static part
```

To report violations in the above example, use the [check_static_value](#) parameter as follows:

- Set this parameter to `only_const` to enable Tag checking for RHS expression, which is static.
- Set this parameter to `only_expr` to enable Tag checking for RHS expression, which is a non-static expression that contains a static part.
- Set this parameter to `yes` to enable Tag checking for static expression and non-static expression that contains a static part.

NOTE: Cases of the type, $a = b + c + 1$, are considered as complex counters. Set either the [strict](#) or [check_static_value](#) parameter to report violations for such cases.

By default, violations are not reported for the counter type of cases. Set the [strict](#) parameter to report violations for such cases. Consider the following example:

```
wire [2:0] a, b, c;
assign b = c + 1; //Will flag with strict set to yes
assign a = b + c + 5; //Not a counter, will not be flagged
//with strict set to yes
```

For static expressions, the Tag reports a violation if carry or borrow bit is lost. For example:

```
wire [1:0] out;
assign out = 2'b10 + 1'b1;
assign out = 2'b10 + 2'b11;
```

The same logic is also applicable for static expressions inside concatenation. For example:

```
wire [2:0] out1;
wire c;
```

```
assign out1 = {2'b10+2'b01,c};
assign out1 = {2'b10+2'b10,c};
```

When a static value is added or subtracted from a non-static value, VC SpyGlass Lint adds the maximum non-static value to the static value, and based on this value, VC SpyGlass Lint calculates the width of the expression.

For example, consider the following code snippet:

```
wire [11:0] b;
wire [3:0] c;
assign b = 12'hFF0 + c;
assign b = 12'hFF1 + c;
```

In the above example, the maximum value of the first expression is 4095 and there is no overflow. Therefore, the Tag does not report a violation for the first expression. In addition, the maximum value of the second expression is 4096 and there is an overflow. Therefore, the Tag reports a violation for the second expression.

By default, the Tag does not report any violation inside concat operator, if guard bit is added to carry bit to accommodate overflow. For example, no violation is reported in the following case:

```
wire [2:0] b,c;
wire [3:0] result;
wire [5:0] out

assign result = {b[0],b} + {c[0],c};
// LSB added to carry
assign result = {b[2],b} + {c[2],c};
// MSB added to carry
assign out = {{2{b[2]}},b[2],b} + {{2{c[2]}},c[2],c};
// MSB added to carry
assign out = {{3{b[2]}},b[2],b[2:1]} + {{3{c[2]}},c[2],c[2:1]};
// MSB added to carry
```

Set the *strict* parameter to report such cases.

Consider a case in which RHS contains an addition operator with at least one concatenation expression. The Tag does not report a violation if the number of continued zeros at the end of concatenation expression is

greater than or equal to the width of other operand. For example, no violation is reported in the following case:

```
wire [6:0] a;
wire [3:0] b;
wire [2:0] c;
assign a = {b, 3'b0} + c;
assign a = {3'b111} + {b, 3'b0};
```

NOTE: The W484 Tag supports the generate-if, generate-for, and generate-case blocks.

By default, if the RHS expression is a binary expression and each term is a concatenation expression padded with zero bits, the Tag checks for overflow while considering zero padded bits. That is, violation is not reported if the actual width of the expression can be accommodated in zero padded bits.

When the [handle_large_expr](#) parameter is set to yes, the Tag limits this checking for expressions containing up to 25 terms and for larger expressions, the LRM width is considered. For example:

```
module top();
wire [5:0] a1;
reg a,b,c,d,e,f,g,h,i,j;
assign a1[4:0] = {4'b0,a} + ({4'b0, b} + ({4'b0, c} +
({4'b0, d} + ({4'b0, e} + ({4'b0, f} + ({4'b0, g} + {4'b0,
h})))))) ; //8 terms, no violation
assign a1[4:0] = {4'b0,a} + ({4'b0, b} + ({4'b0, c} +
({4'b0, d} + ({4'b0, e} + ({4'b0, f} + ({4'b0, g} + {4'b0,
h})))))) + {4'b0,a} + ({4'b0, b} + ({4'b0, c} + ({4'b0, d} +
({4'b0, e} + ({4'b0, f} + ({4'b0, g} + {4'b0, h})))))) +
{4'b0,a} + ({4'b0, b} + ({4'b0, c} + ({4'b0, d} + ({4'b0, e}
+ ({4'b0, f} + ({4'b0, g} + {4'b0, h})))))) + {4'b0, b};
// 25 terms, no violation
assign a1[4:0] = {4'b0,a} + ({4'b0, b} + ({4'b0, c} +
({4'b0, d} + ({4'b0, e} + ({4'b0, f} + ({4'b0, g} + {4'b0,
h})))))) + {4'b0,a} + ({4'b0, b} + ({4'b0, c} + ({4'b0, d}
+({4'b0, e} + ({4'b0, f} + ({4'b0, g} + {4'b0, h})))))) +
{4'b0,a} + ({4'b0, b} + ({4'b0, c} + ({4'b0, d} + ({4'b0,
e} + ({4'b0, f} + ({4'b0, g} + {4'b0, h})))))) + {4'b0, b}
```

Lint Tags

```
+ {4'b0, b};      //26 terms, no violation by default,
violation is reported when parameter handle_large_expr is
set
```

```
endmodule
```

In the above example, by default the Tag will not report violation for any of the assignment.

When the [handle_large_expr](#) parameter is set then the Tag will report violation for the last statement as LRM width is considered for this because it has more than 25 terms.

Message Details

The following message appears at the location where the width `<widthl>` of LHS expression `<lexpr>` is not wider than the width `<widthr>` of the RHS expression `<rexpr>`:

Possible assignment overflow: lhs width `<widthl>` (Expr: '`<lexpr>`') should be greater than rhs width `<widthr>` (Expr: '`<rexpr>`') to accommodate carry/borrow bit [Hierarchy: '`<hier-path>`']

However, if the value of the [use_natural_width](#) parameter is yes and the width of RHS is greater than the LHS width, the following violation message is displayed:

Possible assignment overflow: lhs width `<width>` (Expr: '`<lhs_expr>`') should not be less than rhs natural width `<natural_width>` (Expr: '`<rhs_expr>`') to accommodate carry/borrow bit, [Hierarchy: '`<hierarchy>`']

Where, `<hier-path>` is the complete hierarchical path.

Tag Severity

Warning

Suggested Fix

Make sure you intend to discard the carry or borrow bit.

W486

Reports shift overflow operations

When to Use

Use this Tag to identify shift overflow operations.

Description

The W486 Tag reports violation for shift overflow operations.

Width Calculation

The W486 Tag calculates the width of left operand of shift operator on the basis of the following conditions:

- If the [nocheckoverflow](#) Tag parameter is set to `yes` or W486, width is calculated as per LRM. However, for constants, natural width is considered.
- If the [nocheckoverflow](#) is set to `no`, width is calculated according to the following methods:
 - ? For plus/minus operator, value based width is considered. For example:


```
a[2:0]+b[2:0])>>1
// Width of expression, a[2:0]+b[2:0], will be 4
(7+7=14, 4 bits wide)
```
 - ? For multiplication operator, width will be sum of operand widths. For example:


```
(a[2:0]*b[2:0])<<1;
// Width of expression, a[2:0]*b[2:0], will be 6
(a1[2:0]*b1[2])<<2;
// Width of expression, a1[2:0]*b1[2], will be 3
```
 - ? For division operator, LHS width will be considered. For example:


```
(a[2:0]/b[2])<<2;
// Width of expression, a[2:0]/b[2], will be 3
```
 - ? For concat operator, width will be calculated as sum of all the operands. For example:


```
1'b0,a[2:0]}<<1;
```

```
// Width concat exp=3 bits (leading 0 ignored)
{1'b1,a[2:0]}<<2;
// Width concat exp=4 bits (1+3 bits)
```

NOTE: For new width related changes, refer to New Width Flow Application Note.

NOTE: The W486 Tag supports generate-if, generate-for, and generate-case blocks.

Language

Verilog

Default Weight

5

Parameter(s)

- [*disable_rtl_deadcode*](#): The default value is *no*. Set the value of the parameter to *yes* to disable violations for disabled code in loops and conditional (if condition, ternary operator) statements.
- [*enable_rtl_deadcode*](#): The default value is *no*. Set this parameter to *yes* to report different violations for dead codes in the RTL resulting from *if/for/while* and ternary operators that are ignored when the [*disable_rtl_deadcode*](#) parameter is enabled.
- [*nocheckoverflow*](#): The default value is *no*. Set the value of the parameter to *yes* or Tag name to check the bit-width as per the LRM.
- [*use_lrm_width*](#): The default value is *no*. Set the value of the parameter to *yes* to consider the LRM width of integer constants, which is 32 bits.
- [*check_shifted_width*](#): The default value is *no*. Set the value of the parameter to *yes* to consider the natural width of the left operand for a **left shift expression**, if left operand is constant integer and right operand is non static.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location of a left-shift operation where the width *<widthr>* of RHS expression *<rexpr>* is greater than

the width `<widthl>` of the LHS expression `<lexpr>`:

[WARNING] Rhs width '`<widthr>`' with shift (Expr: '`<rexpr>`') is more than lhs width '`<widthl>`' (Expr: '`<lexpr>`'), this may cause overflow [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path.

Potential Issues

Violation may arise when the width of the RHS expression is greater than the width of the LHS expression in a left-shift operation.

Consequences of Not Fixing

When an expression containing a left-shift is assigned to a bus with a width less than required to hold some of the most-significant shifted bits, such bits are truncated and this can lead to arithmetic functional error.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the width mismatch due to shift overflow is found.

To fix the violation, make sure you intend to discard the overflow bits.

Example Code and/or Schematic

Example 1

Consider the following example:

```
wire [2:0] net1, net2;
assign net1 = net2 << 1;
```

In the above example, the W486 Tag reports a violation as the expression containing a left-shift is assigned to a bus with a width less than required to hold some of the most-significant shifted bits.

Example 2

Consider the following examples for based numbers for which the W486 Tag does not report a violation:

```
wire [4:0] w1;
assign w1 = 5'b1 << 2;
assign w1 = 3'b1 << 2;
```


Example 3

Consider the following example:

```
module mod(in1, clk, sel, out1);
    input [2:0] in1;
    input clk, sel;
    output [2:0] out1;
    reg [2:0] out1;

    always @(posedge clk)
        if(sel)
            out1 = in1;
        else
            out1 = in1 << 1;    //violation(Rhs width (Expr: '(in1
                                << 1)') is more than lhs Lhs width (Expr: 'out1'), this may
                                cause overflow)

endmodule
```

In the above example, the W486 Tag reports a violation as the RHS width of the expression, `(in1 << 1)`, is more than the LHS width of the expression, `(out1)`. This condition may cause an overflow.

Example 4

Consider the following example:

```
module top ();

wire [4:0]a;
reg [1:0] b;
assign a[3:0] = 1 << b; //Violation by default
assign a[3:0] = 'b11 << b; //Violation by default
assign a[3:0] = 'd13 << b; //Violation by default

endmodule
```

In the above example, for all three assignments by default, the width is considered as 32 bits for the RHS and a violation is reported.

When the [*check_shifted_width*](#) parameter is set to *yes*, the natural width of left operand of the RHS expression is considered and no violation is

reported for any of the assignments.

Default Severity Label

Warning

Tag Group

Expression, Lint_Elab_Tags

Reports and Related Files

No related reports or files.

W489

The last statement in a function does not assign to the function (Verilog)
The last statement in a function does not assign to the function (VHDL)

Language

Verilog, VHDL

Description

The W489 Tag flags functions where the last statement in the function description is not assigning to the function return value.

Statements after the last function return value assignment are effectively redundant. Hence, any functionality described after the last function return value assignment is lost.

The W489 Tag does not report a violation if there is statement following the return value assignment inside the following constructs:

- if-else
- while loop
- for loop
- repeat
- case
- assign
- Other looping statements

Message Details

Verilog

The following message appears at the location of the last function return value assignment statement in a function description when other statements exist after this last assignment to the function return value:

The last statement in a function does not assign a return value

VHDL

The following message appears at the location of a function `<func-`

name> that can return without executing a return statement:

The function <func-name> may return without executing a return statement

Tag Severity

Guideline (Verilog) / Warning (VHDL)

Suggested Fix

Check the function logic again. You should either remove the statements after the last assignment to the function value, or determine what function value should be assigned after those statements.

W490

A control expression/sub-expression is a constant

Language

Verilog, VHDL

Description

The W490 Tag flags constant control expressions or sub-expressions.

Constant control expressions or sub-expressions may result in an always ON or an always OFF logic inference. Such descriptions are not recommended.

Constant control expressions or sub-expressions should only occur in hard-wired configuration settings. Use this Tag to find and check all such cases.

By default, the W490 Tag does not flag such constructs in subprogram descriptions. For VHDL designs, set the [strict](#) parameter to check for constant control expressions inside subprogram descriptions also.

NOTE: *The W490 Tag supports generate-for block.*

Message Details

The following message appears at the location of a constant control expression or sub-expression `<expr>`:

Control expression/sub-expression '`<expr>`' is a constant

Tag Severity

Warning

Suggested Fix

Fix may not be required as long as you are aware that portion of your logic may be permanently disabled/optimized out as a result.

Examples

Consider the following example:

```
module test (in, in1, out, out1, out2, clk);  
    input [3:0] in,in1;
```

```
input clk;
output out, out1, out2;

reg out, out1, out2;
wire data, node;
parameter par = 4'b1010;

assign data = 2 ? in : in1;
assign data = par ? in : in1;

always @(clk)
begin
    if(3)
        out = node ? in1 : in;
    else
        out = in1;
    while (par < 3)
        out1 = in1 & in;
    while (par[2])
        out2 = in1 & in;
end
endmodule
```

In the first conditional assignment statement, the select condition (2) is a constant. Thus, VC SpyGlass Lint generates the following message:

Control expression/sub-expression '2' is a constant

In the second conditional assignment statement, the select condition (par) is a parameter. Thus, VC SpyGlass Lint generates the following message:

Control expression/sub-expression 'par' is a constant

In the first always construct, the if statement condition (3) is a constant. Thus, VC SpyGlass Lint generates the following message:

Control expression/sub-expression '3' is a constant

In the always construct, the first while statement condition (par < 3) evaluates to a constant. Thus, VC SpyGlass Lint generates the following message:

Lint Tags

Control expression/sub-expression '(par < 3)' is a constant

In the `always` construct, the second `while` statement condition (`par[2]`) is a constant. Thus, VC SpyGlass Lint generates the following message:

Control expression/sub-expression 'par[2]' is a constant

W491

Reports case expression with a width greater than the specified value

When to Use

Use this Tag to identify case expression that have a width greater than the specified value and have a don't care (?) value in the high order bit/byte.

Description

The W491 Tag reports violation for case clause condition constants where the size specification of the constant is wider than the value specification and the high-order bit/byte is don't care (?). As a result, the constant is ?-extended.

Tag Exceptions

No Tag checking is done for unused macro definitions and unused parameters.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where the case clause expression constant *<constant>* is ?-extended:

[WARNING] Constant *<constant>* will be ?-extended

Potential Issues

Double-click the violation message. The HDL window highlights the line

where constant is ?-extended. Review the code, use the correct width and value specification rather than depending on the auto extension by simulator.

Consequences of Not Fixing

Due to extension, more cases may match the expression than intended.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location of the offending constant that is ?-extended.

To fix the violation, it is best to use the correct width and value specification, rather than depending on default extension. This is particularly important here because the implication of extension is that more cases may match than you intended.

Example Code and/or Schematic

Consider the following examples:

Specified Value	Equivalent Value	Violation Reported
8'b?0	8'b???????0	Yes as the value is ?-extended
9'h?0	9'b?????0000	Yes as the value is ?-extended
8'h?0	8'b????0000	No as the value is not ?-extended

Default Severity Label

Warning

Tag Group

Expression

Reports and Related Files

No related reports or files.

W493

A variable is not declared in the local scope, that is, it assumes global scope

Language

Verilog, VHDL

Description

The W493 Tag flags variables that are read but not declared in the local scope.

Using variables in this manner causes the variable value to be read from a more global scope, if available or for the variable to be implicitly declared, and possibly not set.

This design practice is not recommended and may result in unexpected values.

NOTE: *The W493 Tag supports generate-if and generate-for blocks.*

NOTE: *(Verilog) The W493 Tag is switched off by default. You can enable this Tag by specifying the `configure_lint_tag -enable -tag W493` command.*

Message Details

Verilog

The following message appears at the location where a variable `<var-name>` is read that is not declared in the local scope:

Temporary variable '`<var-name>`' is not declared in local scope

VHDL

The following message appears at the location where a shared variable `<var-name>` is used:

Shared variable '`<var-name>`' used. Avoid using shared variables

Tag Severity

Fatal (Verilog) / Warning (VHDL)

Suggested Fix

It is always best to avoid coding which depends on or creates side-effects. Pass values from other contexts explicitly as parameters/arguments.

W494

Inout port is not used

Language

Verilog, VHDL

Description

The W494 Tag flags inout ports that are never read or set in the design.

While such ports are allowed, the W494 Tag helps you clean up your design.

NOTE: *For Verilog, inside the nested `for` loops, signals of user-defined type like structures, interfaces, etc. are considered fully set if used on the left-hand side of an expression and fully read if used on the right-hand side of an expression.*

By default, the Tag reports violation for the completely unused inout ports. Set the value of the [strict](#) parameter to `yes` to also report violation for partially unused ports.

By default, this Tag does not process large arrays. Set the [handle_large_bus](#) parameter to `yes` to process large arrays (greater than 50,000) and report violation if not used correctly.

Message Details

The following message appears at the location where an inout port `<port-name>` is declared but never read or set in the design:

Verilog

Unused inout '`<port-name>`'. [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical name of the containing scope.

VHDL

The inout port '`<port-name>`' is not used

Tag Severity

Warning

Suggested Fix

Check to make sure you intended to ignore the output value.

W495

Inout port is never set

Language

Verilog, VHDL

Description

The W495 Tag flags inout ports that are read in the design but never set.

While such ports are allowed, the W495 tag helps you clean up your design so that you can redefine such inout ports as input ports.

It is an error in CMOS to let an input float, unless that input has a `pull-up` or `pull-down`. Since an inout has an input mode, this condition is an error.

NOTE: *For Verilog, inside the nested `for` loops, signals of user-defined type like structures, interfaces, etc. are considered fully `set` if used on the left-hand side of an expression and fully `read` if used on the right-hand side of an expression.*

By default, the tag reports violation when inout port is read but not set. However, to enable the tag to report violation for inout ports that are unread and not set, set the value of the [check_unread](#) parameter to yes.

By default, this Tag does not process large arrays. Set the [handle_large_bus](#) parameter to yes to process large arrays (greater than 50,000) and report violation if not used correctly.

Message Details

The following message appears at the location where an inout port `<port-name>` is declared that is read in the design but is never set:

Verilog

Undriven inout '`<port-name>`'. [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical name of the containing scope excluding subprograms.

VHDL

The inout port '`<port-name>`' is not set

Lint Tags

Tag Severity

Warning

Suggested Fix

If you don't care about the input value, tie it high or low, but do not let it float.

W496a

Reports comparison to a tristate in a condition expression

When to Use

Use this Tag to identify the comparisons to a tristate value in the condition expressions.

Description

The W496a Tag reports violation for comparisons to tristate signals in `if` statement conditions and conditional statements.

NOTE: *The W496a Tag supports generate-if, generate-for, and generate-case blocks.*

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a comparison (using operator `<op-name>`) with tristate value `<value>` is encountered in a control expression:

[WARNING] Comparison (`<op-name>`) to tristate value (`<value>`) is treated as false in synthesis

Potential Issues

Violation may arise when a tristate value is compared in a conditional expression.

Consequences of Not Fixing

Comparison to a tristate value does not have a physical equivalent in

synthesis and always defaults to a false value.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where a comparison with a tristate value is encountered.

To fix the violation, avoid making such comparisons in synthesizable code. If the check is being made for simulation reasons, bracket it in `translate_off` and `translate_on` statements.

Example Code and/or Schematic

Example 1

Consider the following example of an if Statement Condition:

```
always @ (en1 or in1 or in2)
  if (en1 == 2'b1z)
    out <= in1;
  else
    out <= in2;
```

In the above example, the signal, `en1`, is being compared with a tristate value, `2'b1z`, in the `if` statement condition. Also, the W496a Tag reports the following violation message for this example:

Comparison (==) to tristate value (1z) is treated as false in synthesis

Example 2

Consider the following example of a conditional statement.

```
always @ (en1 or in1 or in2)
  out <= (en1 == 1'bz) ? in1 : in2;
```

In the above example, the conditional statement condition, `en1`, is being compared with a tristate value, `1'bz`. Also, the W496a Tag reports the following violation message for this example:

Comparison (==) to tristate value (z) is treated as false in synthesis

Example 3

Consider the following example:

```
module test(out);  
output out;  
reg en1,out;  
always @ (en1)  
if (en1 == 2'b1z)  
out = 2'b10;  
else  
out = 2'b11;  
endmodule
```

In the above example, the W496a Tag reports a violation message as the comparison to tristate value, 1z, is treated as false in synthesis.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

No related reports or files.

W496b

Reports comparison to a tristate in a case statement

When to Use

Use this Tag to identify the comparisons to a tristate value in a case statement.

Description

The W496b Tag reports violation for comparisons to tristate signals in case construct control expressions.

NOTE: *The W496b Tag supports generate-case block.*

Tag Exceptions

The W496b Tag ignores casex and casez constructs.

Language

Verilog, VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a comparison with tristate value `<value>` is encountered in a case construct control expression `<expr>`:

[WARNING] Case comparison of expression: "`<expr>`" to tristate value: '`<value>`' is treated as false in synthesis

Potential Issues

Comparison to a tristate value does not have a physical equivalent in

synthesis and always defaults to a false value.

Consequences of Not Fixing

Comparison to a tristate value does not have a physical equivalent in synthesis. Synthesis tools typically default the result of the comparison to a false value.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where a tristate value is used in a case label.

To fix the violation, avoid making such comparisons in synthesizable code. If the check is being made for simulation reasons, bracket it in `translate_off` and `translate_on` statements.

Example Code and/or Schematic

Example 1

In the following example, case construct selector `sel` is being compared with a tristate value `2'bzz`:

```
case(sel)
  2'b00 : outp = inp[0];
  2'b01 : outp = inp[1];
  2'bzz : outp = inp[2];
default : outp = inp[3];
endcase
```

For this example, VC SpyGlass Lint generates the following message:

Case comparison of expression: "sel" to tristate value: 'zz' is treated as false in synthesis

Example 2

Consider the following example:

```
module top(out,in1,in2);
output out;
reg out,sel;
input in1,in2;

    always @(sel)
    case (sel)
```

Lint Tags

```
1'b0 : out = in1;  
1'bz : out = in2;  
endcase
```

```
endmodule
```

In the above example, the W496b Tag reports a violation message as the case comparison of expression to tristate value, z, is treated as false in synthesis.

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

No related reports or files.

W497

Not all bits of a bus are set

Language

Verilog

Description

The W497 Tag flags multi-bit signals that are not completely set in the design.

The W497 Tag does not check multi-bit input and inout ports.

When some bits of a multi-bit signal are not set in the design, these bits may achieve unknown values and may infer latches or flip-flops during synthesis.

For all multi-bit signals (single-dimension vectors and multidimensional arrays), the total count of unset bits is reported in the violation message. To see the details of the violating bits, please refer to the [SignalUsageReport](#). You can access the report from the *VC SpyGlass Lint_reports/lint* directory.

Parameter(s)

[*modport_compat*](#): Default value is no. Set this parameter to yes to not evaluate mod ports that are used as hier scope variables in a port connection.

Message Details

The following message appears at the location of a multi-bit signal declaration *<sig-name>* when all bits of the signal are not set in the design:

Not all bits of bus '*<sig-name>*'*<no-of_bits>* are set
[Hierarchy: '*<hier-path>*']

Where,

- *<hier-path>* is the complete hierarchical path of the signal.
- *<no_of_bits>* is the number of unset bits of vector or multidimensional signals.

Also, the following message is generated, when the W497 Tag flags a

Lint Tags

violation for vector or multidimensional signal:

Please refer to 'SignalUsageReport.rpt' for details of violating bits

In the Console, click on the above violation message to view the [SignalUsageReport](#).

Tag Severity

Warning

Suggested Fix

If the bits are not required, use a narrower bus, or tie the unused bits to a fixed value.

W498

Not all bits of a bus are read

Language

Verilog

Description

The W498 tag flags multi-bit signals that are not completely read in the design.

When some bits of a multi-bit signal are not read in the design, it may not be an error but should be examined closely as you would always expect all bits to be read.

For all multibit signals (single-dimension vectors and multidimensional arrays), the total count of unread bits is reported in the violation message. To see the details of the violating bits, please refer to the [SignalUsageReport](#). You can access the report from the *VC SpyGlass Lint_reports/lint* directory.

Parameter(s)

- [avoid_port_signal](#): Default value is no. Set this parameter to yes to enable the tag to not report a violation for port signals.
- [modport_compat](#): Default value is no. Set this parameter to yes to not evaluate mod ports that are used as hier scope variables in a port connection.

Message Details

The following message appears at the location of a multi-bit signal declaration `<sig-name>` when all bits of the signal are not read in the design:

Not all bits of bus '`<sig-name>`'`<no_of_bits>` are read
[Hierarchy: '`<hier-path>`']

Where,

- `<hier-path>` is the complete hierarchical path of the signal.
- `<no_of_bits>` is the number of unread bits of vector or multidimensional signals.

Also, the following message is generated, when the W498 Tag flags a violation for vector or multidimensional signal:

Please refer to 'SignalUsageReport.rpt' for details of violating bits

In the Console, click on the above violation message to view the [SignalUsageReport](#).

Tag Severity

Warning

Suggested Fix

No fix necessarily required, but such cases should be examined for possible errors.

W499

Ensure that all bits of a function are set.

When to Use

Use this Tag to identify the undefined bits of a function.

Description

The W499 Tag reports violation for functions returning a multi-bit value where all bits of the function return value are not assigned in the function description.

Language

Verilog

Default Weight

5

Parameter(s)

- [*ignore_auto_function_return*](#): The default value is no. Set the value of the parameter to yes to ignore name of automatic functions to be set in all branches of function body.
- [*handle_large_expr*](#): The default value is no. Set the value of the parameter to yes to process large arrays and raise violations if not correctly used.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the first line of a function description where all bit of the function return value are not assigned in the function description:

[WARNING] Not all bits of the function '<function_name>' value are set in the function body or in all branches of the function
[Hierarchy: '<hier_path>']

Potential Issues

The W499 Tag reports a violation when different bits in different parts of the function are not set or some bits in the conditional branch are not set.

Consequences of Not Fixing

This can be very error prone if not fixed. A user of the function will expect all bits of a function to be set.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the function, where, not all bits of the function are set.

Review the code and confirm the reason for not setting all bits of the function.

To fix the violation, make sure you define all bits in all possible branches in the function.

Default Severity Label

Warning

Tag Group

Function-task

Reports and Related Files

No related reports or files.

W501

A connection to a reset port should not be a static name

Language

VHDL

Description

The W501 Tag flags reset ports in component instantiations that are connected to static names (generic or constant).

Message Details

The following message appears at the location where the reset port *<rst-name>* is connected to a generic or constant in a component instantiation:

Connection to <reset> port '*<rst-name>*' is a static name
(*<type>*)

Where *<type>* can be generic or constant.

Severity

Warning

Suggested Fix

Always connect a real signal. Tie that signal off if you really want to disable the reset.

Examples

Consider the following example where the reset port *rst* of instantiation *INST1* is tied to a constant value 0:

```
...
INST1: test port map ( d => input(0),
                      clk => clk,
                      rst => '0',
                      q => s1);
```

```
INST2: test port map (d => s1,
```

```
clk => clk,  
rst => input(1),  
q => output);
```

...

For this example, VC SpyGlass Lint generates the following message:

Connection to reset port 'rst' is a static name (constant)

Also, the reset port `rst` of instantiation `INST2` is connected to signal `input(1)` which is not a constant or generic. Hence, VC SpyGlass Lint does not flag a message here.

W502

Ensure that a variable in the sensitivity list is not modified inside the always block

When to Use

Use this Tag to identify the signals in the sensitivity list that are modified inside the same always block.

Description

The W502 Tag reports violation for the signal/variable that is present in the sensitivity list and is modified in the same always block.

NOTE: *The W502 Tag supports generate-if and generate-for block.*

Language

Verilog

Default Weight

5

Parameter(s)

- [check_implicit_senselist](#): The default value is `no`. Set the value of the parameter to `yes` to report violation for `always @*` and `always_comb`.
- [report_all_messages](#): Default value is `no`. Set this parameter to `yes` or Tag name(s) to report the bit information in the violation messages.

Constraint(s)

None

Messages and Suggested Fix

The following message appears when a signal/variable `<var-name>` is encountered that is present in the sensitivity list and is being modified in that same the always block:

[WARNING] The signal/variable '`<var-name>`' is modified inside always block [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the containing

scope.

Potential Issues

Violation may arise when a variable in the sensitivity list is modified inside the `always` block.

Consequences of Not Fixing

Updating a signal from sensitivity in the same `always` block may result in erroneous simulation results.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the statement where the signal mentioned in the violation message is modified (set) inside the `always` block. The same signal is also present in the sensitivity list of the same `always` block, which can be verified using one of the following ways:

- Scroll up the HDL window, till you reach the `always` block. Verify the presence of the signal in the sensitivity list of this `always` block.
- If the `always` block is a long block, view the HDL in an editor, and, search backwards for the nearest `always` block. Verify the presence of the signal in the sensitivity list of this `always` block.

To fix the violation, ensure that the signal is not present in the sensitivity list, or the signal is not set in the `always` block.

Example Code and/or Schematic

Consider the following example:

```
module test(out1,inp1);
output out1;
input inp1;
reg out1;

always@(inp1 or out1)
begin
    out1 <= inp1;
end
endmodule
```

In the above example, signal `out1` read in the sensitivity list is getting

modified inside the same `always` block.

Default Severity Label

Warning

Tag Group

Simulation, Lint_Elab_Tags

Reports and Related Files

No related reports or files.

W503

An event variable is never triggered

Language

Verilog

Description

The W503 Tag flags `event` variables that are never triggered. Event declarations that are never triggered are redundant.

Message Details

The following message appears at the location of an event declaration `<event-name>` that is never triggered:
event '`<event-name>`' is declared but not used

Tag Severity

Warning

Suggested Fix

This may be a left-over from earlier debug. Suggest you delete the declaration to reduce the number of spurious errors in Tag-checking.

W504

Integer is used in port expression

Language

Verilog

Description

The *W504* Tag reports the port expression that uses integers.

The Tag reports a violation if during instantiating a module, a variable of type integer or a constant integer is connected to the module's port.

By default, the Tag does not report violation for integers used in expressions containing scalar values, such as, 4'h4-2, 4'h4-i, where *i* is an integer variable.

Set the value of the [strict](#) parameter to *yes* to report such violations.

The W504 Tag does not report violation if integer is used in module instantiation inside SystemVerilog module because integer ports are synthesizable in SystemVerilog.

You can run this Tag by specifying the following option in the project file:

```
configure_lint_tag -enable -tag W504
```

If you set the [report_only_overflow](#) parameter to *yes*, the *W504* Tag reports violations only for integers or constant integers where the width of port expression is greater than the width of port. By default, this parameter is set to *no*.

Message Details

The following message appears when the integer *<num>* is encountered in the port expression of the port *<port-name>*:

```
Integer <num> is used in port expression of port <port-name>
```

Severity

Warning

Suggested Fix

Do not use integers in port expression. Pass integer to lower level module by parameters.

Examples

Example 1

In the following example, the *W504* Tag reports violations because integer variable *i* and constant integer 2 are used in the expression of port *a*:

```
module m;
    integer i;
    mm1 u1 (.a(i));
    mm1 u2 (.a(2));
endmodule
```

```
module mm1(input a);
endmodule
```

Example 2

In the following example, the *W504* Tag reports violations, under the *strict* parameter, for integers used in expressions containing scalar values:

```
module top(output O);
integer i;

test    ins1 (.A(4'h4-2),.Q(O)); //Violation under strict
test    ins2 (.A(4'h3),.Q(O)); //No violation
test    ins4 (.A(4'h4-i),.Q(O)); //Violation under strict

endmodule

module test (input A,Q);
endmodule
```

W505

Ensure that the signals or variables have consistent value.

When to Use

Use this Tag to identify the signals or variables that are assigned values using both blocking and nonblocking assignments

Description

Verilog

The W505 Tag flags signals or variables that are assigned values using both blocking and nonblocking assignments.

When creating waivers, Tag ignores line numbers.

VHDL

The W505 Tag flags signals or variables that are assigned values both with and without `after` clause (or with inconsistent `after` clauses).

Tag Exceptions

For VHDL, the W505 Tag does not report signal assignments in asynchronous reset block.

Language

Verilog, VHDL

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

Verilog

The following message appears at the location where a signal or variable

`<name>` is being assigned in a block (nonblocking) mode when it has already been assigned in a nonblocking (blocking) mode:

[WARNING] variable/Signal '`<name>`' is being assigned in both blocking and nonblocking manner

Potential Issues

A violation is reported arise when a variable or a signal is assigned in both blocking and nonblocking manner.

Consequences of Not Fixing

Synthesis semantics require that the same variable or signal should not be assigned in both blocking mode and nonblocking mode. Otherwise, the pre- and post-synthesis simulation behavior of the RTL and gate-level design may differ.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line, where a signal or a variable is again assigned a value again using the blocking/nonblocking assignment and the previous assignment to that signal was done through nonblocking/blocking assignment.

To fix the violation, ensure that all assignments to the same signal or variable are made in a consistent manner.

VHDL

Message 1

The following message appears at the location where a signal `<sig-name>` is assigned value using an AFTER clause when the same signal has already been assigned value at line `<line-num>` without using an AFTER clause:

[WARNING] Signal '`<sig-name>`' is used with 'after clause' while at `<line-num>` it was used without 'after clause'

Potential Issues

A violation is reported when a signal is first used without an AFTER clause and is subsequently assigned a value using the AFTER clause.

Consequences of Not Fixing

Assigning a value to the same signal/variable both with and without AFTER clauses (or with inconsistent AFTER clauses) is not synthesizable. As delays

are ignored during synthesis, the pre- and post-synthesis simulation of the design containing AFTER constructs may differ.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

The following message appears at the location where a signal `<sig-name>` is assigned value without using an AFTER clause when the same signal has already been assigned value at line `<line-num>` using an AFTER clause:

[WARNING] Signal '`<sig-name>`' is used without 'after clause' while at `<line-num>` it was used with 'after clause'

Potential Issues

A violation is reported when a signal is first used with an AFTER clause and is subsequently assigned a value without using the AFTER clause.

Consequences of Not Fixing

Assigning a value to the same signal/variable both with and without AFTER clauses (or with inconsistent AFTER clauses) is not synthesizable. As delays are ignored during synthesis, the pre- and post-synthesis simulation

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line, where a signal is assigned a value again without the `after` clause and the previous assignment to that signal was done with the AFTER clause.

To fix the violation, ensure that all assignments to the same signal or variable are made in a consistent manner.

Example Code and/or Schematic

Example 1 (Verilog)

Consider the following example:

```
reg [3:0] reg_a;  
reg_a <= 1 ;  
reg_a = 2 ;
```

In the above example, `reg_a` is assigned using both blocking and nonblocking assignments. Hence, a violation is reported in this case.

Example 2 (VHDL)

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

signal z : std_logic;
signal a : std_logic;
signal b : std_logic;
signal c : std_logic;
z <= a or b;
z <= b and c after 10 ns;
```

In the above example, `z` is assigned with and without `after` clause. Hence, a violation is reported in this case.

Example 3

Consider the following example:

```
entity ent_test1 is
  port(a: in bit;
        b: in bit;
        c: in bit;
        z: out bit
  );
end ent_test1;

architecture behav of ent_test1 is
begin
  process(a, b, c)
  begin
    z <= a or b;
    z <= b and c after 10 ns;
  end process;
end behav;
```

In the above example, the W505 Tag reports a violation as the signal, `z`, is used first with `after` clause and then without `AFTER` clause.

Default Severity Label

Warning

Tag Group

Synthesis, Assign

Reports and Related Files

No related reports or files.

W527

Dangling else in sequence of if conditions. Make sure nesting is correct

Language

Verilog

Description

The W527 Tag flags dangling `else` statements (that is, `else` statements following a sequence of `if` statements).

It is recommended to use `begin-end` blocks to force the correct order of evaluation.

Parameter(s)

[*avoid_seq_logic*](#): Default value is no. Set the value of the parameter to yes to yes to report violations for combinational always blocks and ignore violation for sequential always blocks.

Message Details

The following message appears at the location of an `if` statement that is inferred not to have a corresponding `else` statement:

Potential dangling 'else' statement - check your logic

Tag Severity

Warning

Suggested Fix

Use `begin..end` to force the correct evaluation. This removes any ambiguity.

Examples

Consider the following example:

```
if (x)
    if (y) do_B
```

```
else do_C
```

Looking at the structure of the construct, it may appear that `do_C` block will be executed only if `x` is false irrespective of the value of `y`.

However, the Verilog semantics associate the `else` statement with the immediately preceding `if` statement rather than the outermost `if` statement. Thus, the above construct will be evaluated as follows:

```
if (x)
  if (y) do_B
  else do_C
```

The `do_C` block is evaluated only if `x` is true and `y` is false.

W528

A signal or variable is set but never read

Language

Verilog, VHDL

Description

The *W528* Tag reports variables that are set in the design but are never read. In addition, the Tag reports vector variables if all bits are not read in a scope of a process.

However, in case of indirect addressing with parameters (Verilog designs), the vector variable is assumed to be completely read if the complete range of the vector variable can be accessed. See *Examples* below for details.

While setting unused variables is allowed, the Tag helps you clean up your design of unused variables, thereby reducing clutter in the design description or possibly preventing a potential name confusion error.

NOTE: *For Verilog, inside the nested `for` loops, signals of user-defined type like structures, interfaces, etc. are considered fully set if used on the left-hand side of an expression and fully read if used on the right-hand side of an expression.*

To avoid violations for certain signals by this Tag, provide a comma-separated list of those signals in the [not_used_signal](#) parameter. You can specify full signal names or regular expressions in the parameter. For example, if you specify the list, "data_signal,bus_*", in the parameter, the Tag does not report a violation for the data_signal signal and all the signals with names starting from bus_.

For multidimensional variables, only the total count of offending bits (set but not read) is reported in the violation message. To see the details of the violating bits, please refer to the [SignalUsageReport](#). You can access the report from the *VC SpyGlass Lint_reports/lint* directory.

By default, the Tag reports a violation if any bit of a variable is set but not read in the current scope of a module or an architecture. If you set the [checkfullbus](#) parameter to yes, the Tag reports a violation only when all bits of a variable are completely unread in the current scope of a module or an architecture.

By default, this Tag does not process large arrays. Set the [handle_large_bus](#)

parameter to `yes` to process large arrays (greater than 50,000) and report violation if not used correctly.

By default, value `checkfullrecord` is set to `no`. Set this parameter to `yes` to not report violation for a record if at least one element of the record is set and read.

By default, all Tag related data is reported in the `SignalUsageReport` report. Set the value of the `disable_signal_usage_report` parameter to Tag-name to disable data generation in the `SignalUsageReport` report for the specified Tag.

By default, the value of `checkfullstruct` is set to `no`. Set this parameter to `yes` to not report a violation for structs if at least one element of the struct is read.

Message Details

Verilog

Message 1

The following message appears at the location where a variable `<var-name>` is set but is never read in the design:

variable '`<var-name>`'`<no_of_bits>` set but not read. [Hierarchy: '`<hier-path>`']

Where,

- `<hier-path>` is the complete hierarchical name of the containing scope excluding subprograms.
- `<no_of_bits>` is the number of bits of multidimensional array, which are set but not read.

Message 2

The following message is displayed when the signal size is greater than 50,000 and the `handle_large_bus` parameter is disabled:

signal '`<signal-name>`' size too big thus not processed, use '`configure_lint_tag_parameter -tag w528 -parameter HANDLE_LARGE_BUS -value yes`' for enabling handling of these signals

If the `HANDLE_LARGE_BUS` parameter is not enabled, the violation for signals of size greater than 50,000 is missed.

VHDL

The following message appears at the location of the declaration of a signal *<sig-name>* that is set but not read in the design:

The signal '*<sig-name>*' *<bits>* is set but not read

The following message appears at the location of the declaration of a variable *<var-name>* that is set but not read in the design:

The variable '*<var-name>*' *<bits>* is set but not read

Where, *<bits>* are the vector bits that are set but not read. For multidimensional array, *<bits>* is the total number of bits that are set but not read.

In addition, the following message is generated, when the *W528* Tag flags a violation for vector or multidimensional signal:

Please refer to 'SignalUsageReport.rpt' for details of violating bits

In the Console, click the above violation message to view the [SignalUsageReport](#).

Tag Severity

Warning

Suggested Fix

Check your logic. If the declaration and set are redundant, remove both to reduce clutter in Tag-check reports.

Examples (Verilog)

Example 1

Consider the following example where the signal *bigbus* is completely set in the first assign statement:

```
module test1 (a, b);
  input [13:0] a;
  output [1:0] b;
  wire [255:0] bigbus;
```

```

assign bigbus =
    {{16{4'h0}}, {16{4'h0}}, {16{4'h0}}, {16{4'h0}}};
assign b={bigbus[a[13:7]],bigbus[a[6:0]]};
endmodule

```

However, the second `assign` statement that assigns to signal `b` using indirect addressing, only 128 bits ($2^{(13-7+1)}$ or $2^{(6-0+1)}$) of signal `bigbus` can be addressed. Therefore, the *W528* Tag reports a violation message.

Now, consider the following example:

```

module test2 (a,b);
    input [15:0] a;
    output [1:0] b;
    wire [255:0] bigbus;

    assign bigbus =
        {{16{4'h0}}, {16{4'h0}}, {16{4'h0}}, {16{4'h0}}};
    assign b={bigbus[a[15:8]], bigbus[a[7:0]]};
endmodule

```

The signal `bigbus` is completely set in the first `assign` statement. In addition, the second `assign` statement that assigns to signal `b` using indirect addressing, all 256 bits ($2^{(15-8+1)}$ or $2^{(7-0+1)}$) of signal `bigbus` can be addressed. Therefore, the *W528* Tag does not report a violation message.

Example 2

Consider the following example:

```

module mod(in1, clk, out1);
    input [1:0] in1;
    input clk;
    output [1:0] out1;
    reg [1:0] out1;
    [1:0] set;

    assign set = 2'b10;

    always @(posedge clk)

```

```
out1[0] = in1[0] & set[0];
```

```
endmodule
```

In this example, the *W528* Tag, by default, reports a violation for the *set* signal. If you set the *checkfullbus* parameter to *yes* or *not_used_signal* parameter to *set*, the Tag does not report any violation.

Examples (VHDL)

Example 1

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

library ieee;
use ieee.std_logic_1164.all;

entity test is
  port (
    a :      in  bit;
    b :      in  bit;
    y :      out bit
  );
end test;

architecture test_arc of test is
  signal sig : std_logic_vector (7 downto 0);
  signal sig1 : std_logic_vector (7 downto 0);
begin
  sig <= ("11001010");
  y <= a and b;
  sig1(1) <= sig(0);
end test_arc;
```

In this example, the *W528* Tag, by default, reports a violation for *sig* and *sig1* signals. If you set the *checkfullbus* parameter to *yes*, the Tag does not report a violation for the *sig* signal.

Example 2

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

library ieee;
use ieee.std_logic_1164.all;

entity top is

end entity;

architecture behav of top is
type HADDR_RANGE_ARRAY is array (natural range <>) of
std_logic_vector (7 downto 0);
SIGNAL master_haddr : HADDR_RANGE_ARRAY (7 downto 0);
SIGNAL master_haddr1 : HADDR_RANGE_ARRAY (7 downto 0);
SIGNAL master_out : HADDR_RANGE_ARRAY (7 downto 0);
begin
    master_haddr <=
    ("11001010","11001010","11001010","11001010","11001010","110
01010","11001010","11001010");
    blk: process
    begin
    test: for i in 0 to 5 loop
        master_out (i) <=master_haddr (i) and
master_haddr1(i);
    end loop test;
    end process blk;
end behav;
```

In this example, the *W528* Tag, by default, reports a violation for `master_addr` and `master_out` signals. If you set the *not_used_signal* parameter to `master_out`, the Tag does not report a violation for the `master_out` signal. If you set the *not_used_signal* parameter to `master_*`, the Tag does not report any violation for signals with names starting from `master_`.

W546

Duplicate design unit

Language

Verilog

Description

The W546 Tag flags duplicate modules.

This error may occur as a result of including more files than required in the analysis when you have multiple revisions of a design within one directory.

NOTE: *The W546 Tag is switched off by default. You can enable this Tag by either specifying the `configure_lint_tag -enable -tag W546` command or by setting the [verilint_compat](#) Tag parameter to yes.*

Parameters

[waiver_compat](#): Default value is no. If you set the value of this parameter to yes or <Tag-name>, it ensures that the Tag does not generate the line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.

Message Details

The following message appears at the location where a duplicate module name <module-name> is encountered when a module with the same name has been encountered in file <file-name> at line <num>:

Duplicate design unit '<module-name>', previously declared in file '<file-name>' at line no '<num>'

Tag Severity

Fatal

Suggested Fix

Determine which source file contains the correct module description and exclude any other such files from the analysis.

W551

Ensure that a case statement marked `full_case` or a `priority/unique` case statement does not have a default clause.

When to Use

Use this Tag to identify the case statements that are marked either `full_case` or `priority/unique_case` and have a default clause.

Description

The W551 Tag reports violation for the following constructs with a default clause:

- case constructs with `full_case` pragma
- `priority/unique` case constructs

The W551 Tag checks for priority cases inside always blocks, initial blocks, tasks and functions in all scopes like generate block, packages, global scope, and interfaces.

The Tag reports violation when a priority modifier is used with `case`, `casex`, or `casez` statements with default clause as one of its case selection item.

Language

Verilog

Default Weight

5

Parameter(s)

[*check_case_type*](#): Default value is all. Therefore, the W551 Tag checks for priority, unique, and `full_case` cases. Set the value of the parameter to either `priority` or `unique` or `full_case` to check for priority, unique and `full_case` cases, respectively. You can also specify multiple values for the parameter so as to check for the specified cases.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the first line of a case construct with the default clause and full_case pragma or priority/unique case construct with a default clause:

[WARNING] Case statement marked <type> has a default clause

Where, <type> can be full_case or priority_case or unique_case.

Potential Issues

The W551 Tag may report a violation because of the following reasons:

- Full_case pragma or the default clause is redundant.
- In case of priority/unique case the default clause is redundant.

Consequences of Not Fixing

This Tag points to the need of intent review. If the designer meant to specify full_case then there should not be a reason for a default clause. A review at the RTL coding stage can help you uncover subtle design issues.

How to Debug and Fix

Use either the full_case pragma directive or the default clause in a case construct. For priority/unique case constructs, default clause is redundant.

To fix the violation, remove either the full_case pragma or the default clause.

Example Code and/or Schematic

Example 1

Consider the following example:

```
module test(inp, outp, sel);
    input [3:0] inp;
    input [1:0] sel;
    output outp;
    reg outp;

    always @(sel or inp)
        case(sel)
//synopsys full_case
            2'bx0 : outp = inp[0];
```

```
2'b01 : outp = inp[1];  
2'b10 : outp = inp[2];  
default: outp = inp[3];  
endcase
```

```
endmodule
```

In the above example, the W551 Tag reports a violation as the case statement specified `full_case` has a default statement.

Example 2

Consider the following example:

```
module test;  
wire a, b, c;  
logic x;  
always @ (a, b, c)  
begin  
    priority casez (c)  
    a: x = 1'b0;  
    b: x = 1'b1;  
    default: x = x;  
    endcase  
end  
endmodule
```

In the above example, the Tag reports a violation as the default clause is used with a priority case.

Default Severity Label

Warning

Tag Group

Case

Reports and Related Files

No related reports or files.

W552

Different bits of a bus are driven in different sequential blocks

Language

Verilog

Description

The W552 Tag flags bus outputs whose different bit-selects are driven in different sequential `always` constructs.

While such descriptions are allowed, they can be confusing and could lead to non-uniform handling of the bus in some conditions. Such descriptions also create a possibility of (simulation) race conditions on the bus.

For multidimensional arrays, the W552 Tag flags if different bits of packed dimensions are driven in different sequential `always` constructs. For example, the W552 reports a violation in the following case:

```
reg [1:0][3:0]out;
always@(posedge clk1)
    out[1][1] = in1[1];
always@(negedge clk1)
    out[1][0] = in1[0];
always@(negedge clk2)
    out[0] = in1;
```

NOTE: The W552 Tag supports *generate-if*, *generate-for*, and *generate-case* blocks.

NOTE: The W552 Tag does not check for unpacked arrays.

NOTE: The Tag supports the multi-line functionality.

Message Details

The following message appears at the location where a bus output *<bus-name>* is declared whose different bit-selects are driven in different sequential `always` constructs:

Bus '*<bus-name>*' is driven inside more than one sequential block. [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the complete hierarchical name of the containing

scope excluding subprograms.

Tag Severity

Warning

Suggested Fix

Wherever possible, drive all bits of a bus from the same block.

W553

Different bits of a bus are driven in different combinational blocks

Language

Verilog

Description

The W553 Tag flags bus whose different bit-selects are driven in different combinational always constructs.

While such descriptions are allowed, they can be confusing and could lead to non-uniform handling of the bus in some conditions. Such descriptions also create a possibility of (simulation) race conditions on the bus.

NOTE: *The W553 Tag supports generate-if, generate-for, and generate-case blocks.*

NOTE: *The Tag supports the multi-line functionality.*

Message Details

The following message appears at the location where a bus `<bus-name>` is first assigned whose different bit-selects are driven in different combinational always constructs:

```
net/bus '<bus-name>' is driven inside more than one
combinational block. [Hierarchy: '<hier-path>']
```

Where, `<hier-path>` is the complete hierarchical name of the containing scope excluding subprograms.

Tag Severity

Warning

Suggested Fix

Wherever possible, drive all bits of a bus from the same block.

W561

A zero-width-based number may be evaluated as 32-bit number

Language

Verilog

Description

The W561 Tag flags zero-width based numbers.

A zero-width based-number (like 0'd0) is treated as a zero and hence, may result in unwanted results.

It is possible to create a zero width-based number either by omitting the width or by using a zero-value macro to specify the width. Consider the following example in which a zero-value macro is used to specify the width:

```
'define X 0  
... a = 'X'h3
```

In the above example, the based number will be evaluated as 32 bit number, which may not be what you intended.

Message Details

The following message appears at the location where a zero-width based number *<num>* is encountered:

zero-width based number '*<num>*' may be evaluated as 32 bit number

Tag Severity

Warning

Suggested Fix

Check each such case to make sure that the behavior is what you intended.

W563

Reduction of a single-bit expression is redundant

Language

Verilog

Description

The W563 Tag flags unary reduction operations on single-bit expressions.

Unary reduction operators (unary `and` reduction operator (`&`), unary `or` reduction operator (`|`), and unary `xor` reduction operator (`^`)) reduce a vector to a single-bit. However, using the unary reduction operators on single-bit expressions is redundant.

NOTE: The W563 Tag supports *generate-if*, *generate-for*, and *generate-case* blocks.

Message Details

The following message appears at the location where a unary reduction operator `<op-name>` is used on a single-bit expression `<expr>`:

Use of unary reduction operator "`<op-name>`" on a 1-bit expression "`<expr>`" is unnecessary. [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path.

Tag Severity

Warning

Suggested Fix

Examine the logic to make sure behavior is what you intended.

W575

Logical NOT operating on a vector

Language

Verilog

Description

The W575 Tag flags logical NOT operators used on vector signals.

Logical NOT evaluates its operand as a Boolean (`zero = false`, `non-zero = true`) and returns the logical inverse.

If you intent to perform a bit-wise inversion, use the bit-wise inverse (`~`) operator.

NOTE: *The W575 Tag supports `generate-if`, `generate-for`, and `generate-case` blocks.*

Message Details

The following message appears at the location where a logical NOT operation is used on a multi-bit signal `<sig-name>` of width `<num>`:

Logical NOT(!) operator used on a multibit(`<num>`) value: `<sig-name>`

Tag Severity

Warning

Suggested Fix

Examine the logic to make sure behavior is what you intended.

W576

Logical operation on a vector

Language

Verilog

Description

The W576 Tag flags logical operations performed on vector signals.

It is particularly easy to confuse with the usage of operators, `&` and `&&`. The operations using these two operators (for example `a & b` and `a && b`) will in general return very different results. The operation using the `&` operator is a bit-wise and of two vectors which could evaluate to a vector with the number of bits set. However, the operation using the `&&` operator evaluates its arguments as Boolean (`zero = false, non-zero = true`) and returns the logical and, which will always be 1 or 0.

NOTE: *The W576 Tag supports generate-if, generate-for, and generate-case blocks.*

By default, the W576 tag reports violation for conditional expressions with logical operators. Set the value of the [IGNORE_CONDITIONAL_EXPRESSIONS](#) parameter to yes to ignore conditional expressions with logical operators.

By default, the W576 tag reports violation for the parameters. Set the value of the [ignore_parameters](#) parameter to enable the W576 tag to ignore violation for the parameters.

By default, the tag reports a violation for signals where it's not a reg/wire/logic/bit/byte or compound elements (one element is one of the aforementioned types). Set the value of the [ignore_non_vector_signals](#) parameter to yes to enable the W576 tag to not report violation for signals where it's not a reg/wire/logic/bit/byte or compound elements (one element is one of the aforementioned types).

Message Details

The following message appears at the location where a logical operator `<operator-name>` is used on a multi-bit signal `<sig-name>` of width `<num>`:

Logical operator (<operator-name>) used on a multibit (<num>) value: <sig-name>

Tag Severity

Warning

Suggested Fix

You should examine each of these cases to make sure you are using the correct operator.

W66

Ensure that a repeat construct has a static control expression

When to Use

Use this Tag to identify `repeat` constructs with non-static control expressions.

Description

The W66 Tag reports violation for `repeat` constructs with non-static control expressions.

NOTE: *The W66 Tag supports generate-if, generate-for, and generate-case blocks.*

Language

Verilog

Default Weight

5

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a non-static `repeat` construct control expression is encountered:

[WARNING] Unsynthesizable repeat loop because repeat expression is not constant.[Hierarchy: '<hier-path>']

Potential Issues

The number of times a `repeat` construct is executed should be a static constant value.

Consequences of Not Fixing

A variable value for `repeat` construct execution is not synthesizable using

simple logic.

If a variable is used in the control part of a loop, the synthesis tool will not have a deterministic way to synthesize the logic. Hence, this will not assure gate level integrity, though RTL may simulate as expected.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the repeat statement whose condition part is not constant.

Verify the variable/signal causing this non-static repeat construct control expression.

To fix the violation, you may need to recode this as multi-cycle logic. This is because there is no easy way to implement a variable bound loop in synthesizable logic.

Example Code and/or Schematic

Consider the following example:

```
module mod(i1, en, o1);
  input i1, en;
  output o1;
  reg o1;

  always@(en or i1)
    repeat(en)
      o1 = i1;

endmodule
```

In the above example, the W66 Tag reports a violation as the repeat expression is not constant, which causes an unsynthesizable repeat loop.

Default Severity Label

Warning

Tag Group

Loop

Lint Tags

Reports and Related Files

No related reports or files.

W69

Ensure that a case statement specifies all possible cases and has a default clause

When to Use

Use this Tag to identify case constructs that does not describes all possible cases and does not have a default clause.

Description

The W69 Tag reports violation for case constructs that do not have all possible clauses described and also do not have the default clause.

NOTE: *The W69 Tag supports generate-if and generate-for block.*

NOTE: *By default, the W69 Tag is switched off, and will be deprecated in a future VC SpyGlass Lint release. You can use the W71 Tag as a replacement of the W69 Tag.*

Tag Exceptions

The W69 Tag ignores case constructs with `full_case` pragma specified.

Language

Verilog

Default Weight

5

Parameter(s)

- **strict:** The default value is no. Set the value of the parameter to yes to check the case constructs in sequential always constructs. You can also specify a comma-separated list of Tag as an input to the parameter.

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears at the first line of a case construct in a combinational always construct that does not have all possible clauses

described and also does not have a default clause:

[WARNING] Case statement is missing cases and has no default in Combinational block [Hierarchy: '<hier-path>']

Where, *<hier-path>* is the complete hierarchical name of the containing scope excluding subprograms.

Potential Issues

A violation is reported when a CASE construct in a combinational always construct does not describes all possible clauses, including the default clause.

Consequences of Not Fixing

A latch is inferred, if the case construct does not describe all possible states and also does not have a default clause. While you may have intended to infer a latch, it is advisable to examine such cases to avoid creation of unexpected logic.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

The following message appears at the first line of a case construct in a sequential always construct that does not have all possible clauses described and also does not have a default clause when the `strict` Tag parameter has been set:

[WARNING] Case statement is missing cases and has no default in Sequential block [Hierarchy: '<hier-path>']

Where, *<hier-path>* is the complete hierarchical name of the containing scope excluding subprograms.

Potential Issues

A violation is reported when a CASE construct in a combinational always construct does not describes all possible clauses, including the default clause when the strict parameter is set.

Consequences of Not Fixing

A latch may be inferred, if the case construct does not describe all possible states and also does not have a default clause. While you may have intended

to infer a latch, it is advisable to examine such cases to avoid creation of unexpected logic.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the case statement with missing default is declared.

To fix the violation, you can perform any of the following tasks:

- add case clauses (or a default clause) for those cases that are not covered and specify the appropriate behavior in that default clause.
- add pragma `full_case` to the case statement, implying you know that the remaining cases can never happen.

Example Code and/or Schematic

Example 1

Consider the following example where the case construct does not have all possible cases described (case `in3=2'b10` is not described) and also does not have a default clause:

```
...
always@(in1 or in2 or in3 or mem)
begin
    case(in3)
        2'b00 : out[0] = in1 || mem[0];
        2'b01 : out[1] = in1 && mem[1];
        2'b11 : out[2] = in2 && mem[3];
    endcase
end
...
```

For this example, VC SpyGlass Lint generates the following message:

Case statement is missing cases and has no default [Hierarchy: '<hier-path>']

Example 2

Consider the following example:

```
module mod(sel,o1,o2,o4);

input [1:0] sel;
```

Lint Tags

```
output [3:0] o1, o2,o4;
reg [3:0] o1, o2,o4
reg [3:0] mem[0:3];

always@(sel)
begin
    case(sel) //violation(Case statement is missing cases
and has no default in combinational block)
        2'b00 : o1 = mem[0];
        2'b01 : o2 = mem[1];
        2'b11 : o4 = mem[3];
    endcase
end

endmodule
```

In the above example, the W69 Tag reports a violation as the case statement has missing cases and a missing default clause.

Default Severity Label

Warning

Tag Group

Case, Lint_Elab_Tags

Reports and Related Files

No related reports or files.

W701

Included file is not used

Language

Verilog

Description

The W701 Tag flags included files that are not required for analysis.

The W701 Tag checks for the presence of the following constructs only in the included file:

- Macros
- Parameters
- Task declarations
- Function declarations
- Module definitions
- User-defined primitives (UDPs)
- Timescale directive
- Interface definitions
- Signal definitions

It is recommended to remove `include` statements for unwanted files from the design to save the time wasted in analyzing such files.

Message Details

The following message appears at the location where an unused file `<file-name>` is included:

File: `<file-name>` is 'included but not used

Tag Severity

Warning

Suggested Fix

Consider removing the include statement.

W71

Ensure that a case statement or a selected signal assignment has a default or OTHERS clause

When to Use

Use this Tag to identify case constructs without the default or OTHERS clause.

Description

The W71 Tag reports violation for case constructs without the default clause (Verilog designs) and case constructs and selected signal statements without the OTHERS clause.

A default (or OTHERS) clause should always be specified in a case construct to handle unexpected situations even if the construct covers all potential situations.

NOTE: *The W71 Tag supports generate-if and generate-for block.*

Tag Exceptions

For **Verilog** designs, the W71 Tag does not report for missing default clause in the following cases:

- If the target signals in the case construct are assigned using a blocking or nonblocking assignment statement before the case statement
- Case constructs that are inside always construct that infer a flip-flop (Set check_sequential Tag parameter to report such cases).
- Case constructs with associated full_case pragma and unique / priority cases.
- Fully-specified case constructs.
- Case statements with static case select (including generate case).

For **VHDL** designs, the W71 Tag has the following exceptions:

- The syntax is reported if all case constructs are not covered.
- When selector is a port and all the case constructs are covered, then the others clause is not required. Hence, no violation is reported in such a case.

- When case selector is a variable then the others clause must be used.

Language

Verilog, VHDL

Default Weight

5

Parameter(s)

- *strict*: The default value is no. Set the value of the parameter to yes to report fully-specified case constructs without the default clause.
- *check_sequential*: The default value is no. Set the value of the parameter to yes to enable Tag checking in sequential block.
- *do_not_run_W71*: The default value is no. Set the value of the parameter to yes to block the W71 Tag. This parameter is provided to avoid duplicate violation in case the W71 Tag is run in mixed mode.

Constraint(s)

None

Messages and Suggested Fix**Verilog**

The following message appears at the location where a case construct is defined without a default clause:

[WARNING] Case statement does not have a default clause and is not preceded by assignment of target signal <block-type>
[Hierarchy: '<hier-path>']

Where, <block-type> can be either a sequential or a combinational block and the <hier-path> is the complete hierarchical name of the containing scope excluding subprograms.

Potential Issues

A violation is reported when a case construct is defined without a default clause.

Consequences of Not Fixing

If all possible cases of the case construct selector are covered, this is not directly an error. However, if the case construct selector has an undefined value (X or Z) and there is no default clause, then the design simulation may produce unexpected results.

If the width of the case construct selector changes as the design evolves, then what had once been a fully covered case construct may become only partially covered and can lead to inferred latches. Hence, it is recommended to always describe a default clause even if all possible cases of the case construct selector are described.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window is displayed. The HDL Viewer window highlights the line where the case statement with missing default is declared.

To fix the violation, add a default clause to specify default behavior. If you are specifying simulation X behavior, bracket this behavior in `translate_off` and `translate_on` pragmas.

VHDL

The following message appears at the location where a construct of `<type>` is defined without an OTHERS clause:

[WARNING] others clause not found in '`<type>`' statement

Where, `<type>` can be a CASE (for case constructs) or selected signal assignment (for Selected Signal Assignments).

Potential Issues

A violation is reported when a CASE construct or a selected signal assignment is defined without an OTHERS clause.

Consequences of Not Fixing

If all possible cases of the case construct selector are covered, this is not directly an error. However, if the case construct selector has an undefined value (X or Z) and there is no default clause, then the design simulation may produce unexpected results.

If the width of the case construct selector changes as the design evolves, then what had once been a fully covered case construct may become only partially covered and can lead to inferred latches. Hence, it is recommended to always describe a OTHERS clause even if all possible cases

of the case construct selector are described.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window is displayed. The HDL Viewer window highlights the line where the case statement with missing OTHERS clause is declared

To fix the violation, add an OTHERS clause to specify default behavior. If you are specifying simulation X behavior, bracket this behavior in `translate_off` and `translate_on` pragmas.

Example Code and/or Schematic

Verilog

Consider the following example where the case construct has all possible clauses described but does not have the default clause:

```
...
always@(in1 or in2 or mem)
begin
    case(in2)
        2'b00 : out1 = in1 || mem[0];
        2'b01 : out1 = in1 && mem[1];
        2'b10 : out1 = in1 && mem[2];
        2'b11 : out1 = in1 && mem[3];
    endcase
end
...
```

For this example, VC SpyGlass Lint generates the W71 Tag message, if you analyze the example using the `strict` Tag parameter.

VHDL

Example of CASE Construct

Consider the following example where the CASE construct does not have an OTHERS clause defined:

```
library ieee;
use ieee.std_logic_1164.all

entity ent is
```


Lint Tags

```

    port(a: in bit_vector(1 downto 0);
          b: out bit_vector(1 downto 0));
end ent;

architecture arc of ent is
begin
    process(a)
    begin
        case a is
            when "00" | "11" => b <= a;
            when "01" | "10" => b <= not a;
        end case;
    end process;
end arc;

```

For this example, VC SpyGlass Lint generates the following message:

others clause not found in 'CASE' statement

Example of Selected Signal Assignment Statement

Consider the following example where the select assignment statement does not have an OTHERS clause:

```

entity ent is
    port(a: in bit_vector(1 downto 0);
          b: out bit_vector(1 downto 0));
end ent;

architecture arc of ent is
begin
    with a select
        b <= a when "00" | "11", not a when "01" | "10";
end arc;

```

For this example, VC SpyGlass Lint generates the following message:

others clause not found in 'selected signal assignment' statement

Default Severity Label

Warning

Tag Group

Case, Lint_Elab_Tags

Reports and Related Files

No related reports or files.

W86

Not all elements of an array are set

Language

VHDL

Description

The W86 Tag flags arrays where all elements are not set.

Designs where all elements of arrays are not set, may not fully exercise all paths in to the array and thus may mask possible errors in index or bit-slice selection.

The W86 Tag will not check for memory elements having non-static index.

To see the details of the bits that are not set, refer to the [SignalUsageReport](#) report in the *VC SpyGlass Lint_reports/lint* directory.

Message Details

Arrays used in Process Sensitivity Lists

The following message appears at the location of a process' sensitivity list that has an array `<array-name>` of type `<array-type>` and all array elements are not set in the process scope (hierarchy `<hier-path>`):

Not all the elements of `<array-type>` '`<array-name>`' (Bits: `<offending-bits>`) are set. [Hierarchy: '`<hier-path>`']

Where, the `<array-type>` can be signal, port, or variable and `<hier-path>` is the complete hierarchical name of the containing scope excluding subprograms.

Arrays not used in Process Sensitivity Lists

The following message appears at the location of an array declaration `<array-name>` of type `<array-type>` that is not used in any process sensitivity list and all array elements are not set in the scope `<hier-path>`:

Not all the elements of `<array-type>` '`<array-name>`' (Bits: `<offending-bits>`) are set [Hierarchy: '`<hier-path>`']

Where, the `<array-type>` can be signal, port, or variable and

`<hier-path>` is the complete hierarchical name of the containing scope excluding subprograms.

Severity

Warning

Suggested Fix

This is an informational Tag. No action may be required if all locations (required) can actually be set.

W88

All elements of a memory are not set

Language

Verilog

Description

The W88 Tag flags memories where all their elements are not set in the design.

The W88 Tag does not check for memory elements having non-static index.

When testing logic, this indicates that the model does not fully exercise all paths into the memory and thus possible errors in address selection may be masked.

Message Details

The following message appears at the location where a memory *<mem-name>* is declared but all its elements are not set in the design:

All elements of memory '*<mem-name>*' are not set [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the complete hierarchical path of the containing process.

Tag Severity

Warning

Examples

Consider the following example where all elements of memory `mem` are not set in the design:

```
module test(in1, in2, out1);
  input [3:0] in1, in2;
  output [3:0] out1;

  reg [3:0] out1;
  reg [7:0] mem [0:3];
```

```
always@(in1)
begin
    out1 = in1 & in2;
    mem[0] = {in1,out1};
    mem[2] = {in1,in2};
end
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

All elements of memory 'mem' are not set [Hierarchy: 'test']

WhileInSubProg

Reports unsynthesizable While statements used inside subprograms

When to Use

Use this Tag to identify the unsynthesizable `while` statements used in subprograms

Description

The `WhileInSubProg` Tag flags the `while` constructs used in sub-program descriptions.

Default Weight

5

Language

VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a `while` statement `<construct-name>` is encountered in a sub-program description:

[WARNING] `<construct-name>` statement inside a subprogram body may be unsynthesizable

Potential Issues

A violation is reported when a `while` statement is used inside a sub-program.

Consequences of Not Fixing

Such descriptions are not synthesizable by some synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the offending while statement has been used inside subprogram.

No fix is required in testbenches or simulation models. In code targeted for synthesis, use for-loops if possible.

Synopsys recommends use of `translate_off/translate_on` directives against `synthesis_off/synthesis_on` directives. If `synthesis_off/synthesis_on` directives are used within an expression, they have the potential to create

Example Code and/or Schematic

Consider the following example:

```
entity e is
end e;
architecture arc of e is
function func (signal k : integer) return integer is

    variable v1,v2,v3 : integer:=7;
begin
    while v2 < 7 loop
    while v1 < 10 loop
        v1 := v1 + 1;
        v2 := k + 1;
    end loop;
    end loop;

    return v2;
end func;
signal s1,s2 : integer;
begin
    s1 <= func(s2);
end arc;
```

In the above example, the *WhileInSubProg* Tag reports a violation as a while statement is used inside a subprog.

Lint Tags

Default Severity Label

Warning

Tag Group

Synthesis

Reports and Related Files

No related reports or files.

W541

A tristate is inferred

Language

Verilog

Description

The W541 Tag flags inferred tristate nets.

NOTE: *You can enable the W541 Tag by specifying the `configure_lint_tag - enable -tag W541` command. However, this Tag will not run if you set the [fast](#) Tag parameter to `yes` and VC SpyGlass Lint product is run.*

Parameters

[CHECK_Z_ASSIGN](#): By default, this parameter is set to 0. This indicates that the tag will not report violations on Z assignment cases.

If you set this parameter to 1, the tag will report violations on Z assignment cases.

Message Details

The following message appears at the location where a tristate signal `<sig-name>` is inferred:

```
A tristate '<sig-name>' is inferred
```

Tag Severity

Info

Suggested Fix

Use muxing wherever possible, rather than tristates.

Examples

A tristate net `sig` is inferred for the following assignment:

```
assign sig = enable ? in1 : 1'bz;
```

LINT_MULTIASSIGN_BLOCKING_SIG

Signal may be multiply assigned in blocking manner (beside initialization) in the same scope.

When to Use

Use this tag to identify signals that are assigned multiple times, in blocking manner, in the same scope.

Description

The tag is equivalent to W415a and is used with the parameter *-checkblocking=yes*.

Parameter(s)

The tag consider the following parameters (same as W415a).

- *waiver_compat*
- *reportsimilarassgn*
- *ignore_reinitialization*
- *check_initialization_assignment*
- *ignore_multi_assign_in_forloop*
- *ignore_bitwiseor_assignment*

Constraint(s)

None

Messages and Suggested Fix

Signal <signal_name> is being assigned multiple times (previous assignment at line <line_no>) in same always block [Hierarchy: '<hier_name>']

Potential Issues

The violation message is reported when a signal is assigned multiple times in the same block.

Consequences of Not Fixing

Not fixing the violation may result in unexpected behavior.

How to Debug and Fix

To fix the violation, ensure that the signal is not assigned multiple times in the same block.

Example Code and/or Schematic

```
module test (a, b, c, d);  
  input a, b;  
  output c;  
  output d;  
  reg c;  
  reg d;  
  
  always @(a or b)  
  begin  
    c <= a;  
    c <= b;  
  end  
  
  always (@a or b)  
  begin  
    d = b;  
    d = a;  
  end  
  
endmodule
```

Reports and Related Files

None

Default Severity Label

Warning

LINT_MULTIASSIGN_NONBLOCKING_SIG

Signal may be multiply assigned in non-blocking manner (beside initialization) in the same scope.

When to Use

Use this tag to identify signals that are assigned multiple times in a non-blocking manner in the same scope.

Description

The tag is equivalent to W415a is used with the parameter *-checknonblocking=yes*

The tag will consider the following parameters same as W415a.

- *waiver_compat*
- *reportsimilarassign*
- *ignore_reinitialization*
- *check_initialization_assignment*
- *ignore_multi_assign_in_forloop*
- *ignore_bitwiseor_assignment*
- *ignore_nonBlockCondition*

Constraint(s)

None

Messages and Suggested Fix

Signal <signal_name> is being assigned multiple times (previous assignment at line <line_no>) in same always block [Hierarchy: '<hier_name>']

Potential Issues

The violation message is reported when a signal is assigned multiple times in the same block.

Consequences of Not Fixing

Not fixing the violation may result in unexpected behavior.

How to Debug and Fix

To fix the violation, ensure that the signal is not assigned multiple times in the same block.

Example Code and/or Schematic

```
output d;
reg c;
reg d;

always @(a or b)
begin
  c <= c
  if (a)
    c <= c+1;
  else
    c <= c-1;

end

always @(a or b)
begin
  c < = c;
  c < = a; //c<=a ? c + 1 : c-1;
end

endmodule
```

Reports and Related Files

None

Default Severity Label

Warning

Miscellaneous Tags

This section explains the Miscellaneous Tags in VC SpyGlass Lint.

ConstSig

Signal has a constant value or can only switch to a constant value.

Description

The *ConstSig* Tag flags nets that are either sourced by a constant value or driven by a flip-flop, which has a D input held to a constant value.

The *ConstSig* Tag flags the following types of constructs:

- Nets that are always held to a constant value
- Nets connected to outputs of flip-flops whose D inputs are held to the same constant value in all conditions
- Tristate signals with enable held to a constant value
- Direct assignments to constants, as shown in the following example, provided the `strict_constsig` Tag parameter has been set to 1. For example:

```
assign a = 1'b0;
```

By default, the *ConstSig* Tag reports the most optimized assignment to constant value. When you set the `strict_constsig` Tag parameter to 1, the *ConstSig* Tag flags all assignments to constant value. Consider the following example:

```
...  
wire a, b;  
supply0 s1;  
assign a = s1;  
assign b = a;  
...
```

Here, the *ConstSig* Tag flags the following message by default as assignment of supply net s1 to net b is the most optimized assignment to constant value:

Net 'top.b' is held to constant 0

However, when you set the `strict_constsig` Tag parameter to 1, the *ConstSig* Tag flags both assignments:

Net 'top.a' is held to constant 0

Net 'top.b' is held to constant 0

Message Details

The following message appears at the location where net <net-name> is connected to a constant value variable or net <name> of the value <value>:

Net '<net-name>' is held to constant '<value>'

The following message appears at the location where net <net-name> is directly connected to constant value <value> when the `strict_constsig` Tag parameter is set to 1:

Net '<net-name>' is held to constant '<value>'

The following message appears at the location where a register <flop-name> that can only switch to the same constant value <const> in all conditions, is first assigned:

Register '<flop-name>' can only switch to <const>

One of the following messages appears at the location where a tristate signal <sig-name> that has a constant enable <en-name>, is first assigned:

Tristate enable '<en-name>' (driving '<sig-name>') is permanently on

Tristate enable '<en-name>' (driving '<sig-name>') is permanently off

Severity

Warning

Verilog Examples

Consider the following example:

```
module test(in1, clk, reset, out1, out2);
    input [1:0] in1;
    input clk, reset;
    output out1, out2;

    reg out1, out2;
    wire wa, wb;
```

```
wire [1:0] en;
wire tbus;

assign wa = 1'b0;
assign wb = wa & in1[0];

always @(posedge clk)
    out1 <= wb;

always @(posedge clk or posedge reset)
    if (reset)
        out2 = 0;
    else
        out2 = 1'b1;

assign en = 1;
assign tbus = en[0] ? in1[0] : 1'bz;
assign tbus = en[1] ? in1[1] : 1'bz;
endmodule
```

The signal `wa` is directly assigned to a constant. However, VC SpyGlass Lint does not flag direct assignments.

The signal `wb` is held to a constant 0 as the signal `wa` is always 0. Therefore, VC SpyGlass Lint generates the following message:

Net 'test.wb' is held to constant 0

When you set the `strict_constsig` Tag parameter is set to 1, signal `wa` is also flagged as follows:

Net 'test.wa' is held to constant 0

The D pin of flip-flop `out1` is connected to signal `wb` that is held to constant 0 and hence the flip-flop output can only switch to 0 in all conditions (as there is no reset sequence). Therefore, VC SpyGlass Lint generates the following message:

Register 'test.out1' can only switch to 0

While the D pin of flip-flop out2 is held to a constant (1), the flip-flop output is different (0) in its reset state. Therefore, VC SpyGlass Lint does not flag this situation.

As the enable signal en of tristate signal tbus is assigned to a constant 1, the lower bit en[0] is always on and the higher bit en[1] is always off. Therefore, VC SpyGlass Lint generates the following messages:

Tristate enable 'test.en[1]' (driving 'test.tbus') is permanently off

Tristate enable 'test.en[0]' (driving 'test.tbus') is permanently on

When you set the `strict_constsig` Tag parameter is set to 1, constant value assignment to signal en is also flagged as follows:

Net 'test.en[1]' is held to constant 0

Net 'test.en[0]' is held to constant 1

VHDL Examples

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

entity test is
  port (in1 : in std_logic_vector(0 to 1);
        clk, reset: in std_logic;
        out1, out2 : out std_logic);
end test;

architecture logic of test is
  signal wa, wb : std_logic;
  signal tbus : std_logic;
  signal en : std_logic_vector(0 to 1);
```

```
begin
    wa <= '0';
    wb <= wa and in1(0);
    en <= "01";

    process (clk)
        begin
            if (clk'event and clk = '1') then
                out1 <= wb;
            end if;

            if (clk'event and clk = '1') then
                if (reset = '1') then
                    out2 <= '1';
                else
                    out2 <= wb;
                end if;
            end if;

            if (en(1) = '1') then tbus <= in1(1);
            else tbus <= 'Z';
            end if;

            if (en(0) = '1') then tbus <= in1(0);
            else tbus <= 'Z';
            end if;
        end process;
    end logic;
```

The signal wa is directly assigned to a constant. However, VC SpyGlass Lint

does not flag direct assignments.

The signal `wb` is held to a constant 0 as the signal `wa` is always 0. Therefore, VC SpyGlass Lint generates the following message:

Net 'test.wb' is held to constant 0

When you set the `strict_constsig` Tag parameter is set to 1, signal `wa` is also flagged as follows:

Net 'test.wa' is held to constant 0

The D pin of flip-flop `out1` is connected to signal `wb` that is held to constant 0 and hence the flip-flop output can only switch to 0. Therefore, VC SpyGlass Lint generates the following message:

Register 'test.out1' can only switch to 0

While the D pin of flip-flop `out2` is held to a constant (1), the flip-flop output is different (0) in its reset state. Therefore, VC SpyGlass Lint does not flag this situation.

The enable signal `en` of tristate signal `tbus` is assigned to a constant 1. Therefore, VC SpyGlass Lint generates the following message:

Tristate enable 'test.en[0]' (driving 'test.tbus') is permanently on

When you set the `strict_constsig` Tag parameter is set to 1, constant value assignment to signal `en` is also flagged as follows:

Net 'test.en[1]' is held to constant 0

Net 'test.en[0]' is held to constant 1

DeadCode

Code does not contribute to functionality of the design

Description

The *DeadCode* Tag flags nets that are either not driven or do not drive anything. The Tag does not report violation for any instance. Instead, the Tag reports violation for output net of the instance that does not contribute to the overall functionality of the design.

The *DeadCode* Tag will not report violation, if the net comes from fan-in or fan-out of a black box as shown in the following example:

```
wire w1,w2;  
blackbox bb1(w1,w2);
```

Message Details

Following message appears at the location where a net <net-name> that does not drive anything, is declared or first assigned:

The net '<net-name>' does not drive anything

Following violation message appears at the location where a net, <net-name> that is not driven, is declared:

The net '<net-name>' is not driven by anything

Following violation message appears at the location where a net, <net-name> that has no effect on any of the outputs, is declared.

Change on net <net-name> has no effect on any of the outputs

Following violation message appears at the location where a net, <net-name> whose inputs have no effect on the net, is declared:

None of the inputs have any effect on net <net-name>

Severity

Warning

Examples

Consider the following example:

```
module test(inout1, inout2, inout3);
```

Miscellaneous Tags

```

    inout inout1, inout2, inout3;

    wire data, enable;
    wire L1_1, L1_2, q_net, L1_3;
    wire L2_1, L2_2, L2_3;
    wire L3_1, L3_2;
    wire L4_1, L4_2;

    nor (L1_1, inout1, inout3);
    or  (L1_2, L1_3, L2_3);
    nand (data, L1_1, L1_2);
    LATCH L1 (.EN(enable), .D(data), .Q(q_net));

    and (L2_1, inout1, inout3);
    or  (L2_2, inout1, inout3);
    bufif1 TRIBUF1 (enable, L2_1, L2_2);
    LATCH L2 (.EN(enable), .D(data), .Q(L2_3)); // loop

    nand (L3_2, L3_1, !L3_2);
    LATCH L3 (.EN(enable), .D(L3_2), .Q(L3_1)); // loop

    nand (L4_1, !L4_1, L4_2);
    LATCH L4 (.EN(!L4_1), .D(L4_1), .Q(L4_2)); // loop
    assign inout2 = L4_2;

endmodule

module LATCH (EN, D, Q);
    input EN, D;

```

```
output Q;

reg     Q;

always @ (D or EN ) begin
    if (EN) Q <= D;
end
endmodule
```

VC SpyGlass Lint generates the following message for net `L1_3` as it is not driven by anything:

The net 'test.L1_3' is not driven by anything

VC SpyGlass Lint generates the following message for net `q_net` as it does not drive anything:

The net 'test.q_net' does not drive anything

VC SpyGlass Lint generates the following message for net `L3_2` as it has no effect on any of the outputs:

Change on net 'test.L3_2' has no effect on any of the outputs

VC SpyGlass Lint generates the following message for net `L4_2` as there is no effect of any input on this net:

None of the inputs have any effect on net 'test.L4_2'

VC SpyGlass Lint generates the following message for net `L3_1` as a change in this net has no effect of any of the outputs:

Change on net 'test.L3_1' has no effect on any of the outputs

VC SpyGlass Lint generates the following message for net `L4_1` as there is no effect of any input on this net:

None of the inputs have any effect on net 'test.L4_1'

Mux01

Connected MUXes should not have a common select.

Description

The *Mux01* Tag flags connected MUXes with common select.

The *Mux01* Tag works on only MUXes inferred from RTL description or MUX gate instances from a library specified by using the `read_file -type gateslib <file>` command in a project file.

In case, there is a path connecting two MUXes, this path will never be used by one of the input signals of the source MUX, due to the presence of a common select.

Message Details

The following message appears at the location where the destination MUX (output net <net2-name>) is inferred if this MUX is connected to a source MUX (output net <net1-name>) and both MUXes have the same select:

Some inputs of mux '<net1-name>' do not propagate to connected mux '<net2-name>', due to common select

Severity

Warning

Reset_check11

Reports asynchronous resets that are used as both active-high and active-low

When to Use

Use this tag to detect incorrect usage of asynchronous reset signals.

Description

The details of the tag are covered under the following topics:

- [Reason for the Reset_check11 tag Violation](#)
- [Functioning of the Reset_check11 tag](#)

NOTE: *The Reset_check11 tag will be deprecated in a future release.*

Reason for the Reset_check11 tag Violation

The *Reset_check11* tag reports asynchronous reset signals that are used at both levels, that is active high and active low, to set or reset flip-flops/sequential-elements.

For example, in the following figure, the `rst` reset is used as active high at `q1` and active low at `q3`:

Miscellaneous Tags

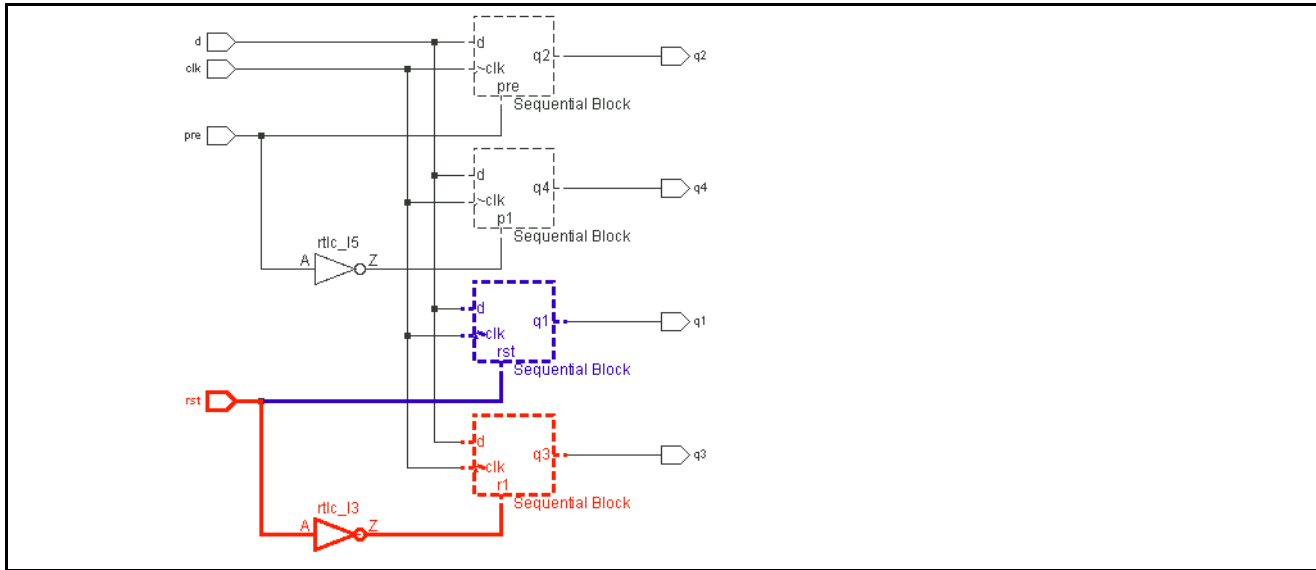


FIGURE 4. Schematic of the Reset_check11 tag Violation

This tag reports any one active-high usage and one active-low usage for an asynchronous reset in a design. You can expand the schematic to view other occurrences of such active-high or active-low usages.

Functioning of the Reset_check11 tag

The *Reset_check11* tag functions in the following manner:

1. It applies both active and inactive value on every reset signal and performs simulation every time for each reset twice, that is, one with the active value and another with the inactive value.
2. It then checks for the existence of two flip-flops such that one flip-flop can be asserted with the active value and the other flip-flop can be asserted with the inactive value.

If such flip-flops exist, the *Reset_check11* tag reports a violation.

tag Exceptions

The *Reset_check11* tag does not report a violation if a reset signal reaches a sequential element with an undetermined phase.

For example, this tag does not report a violation in the following scenario:

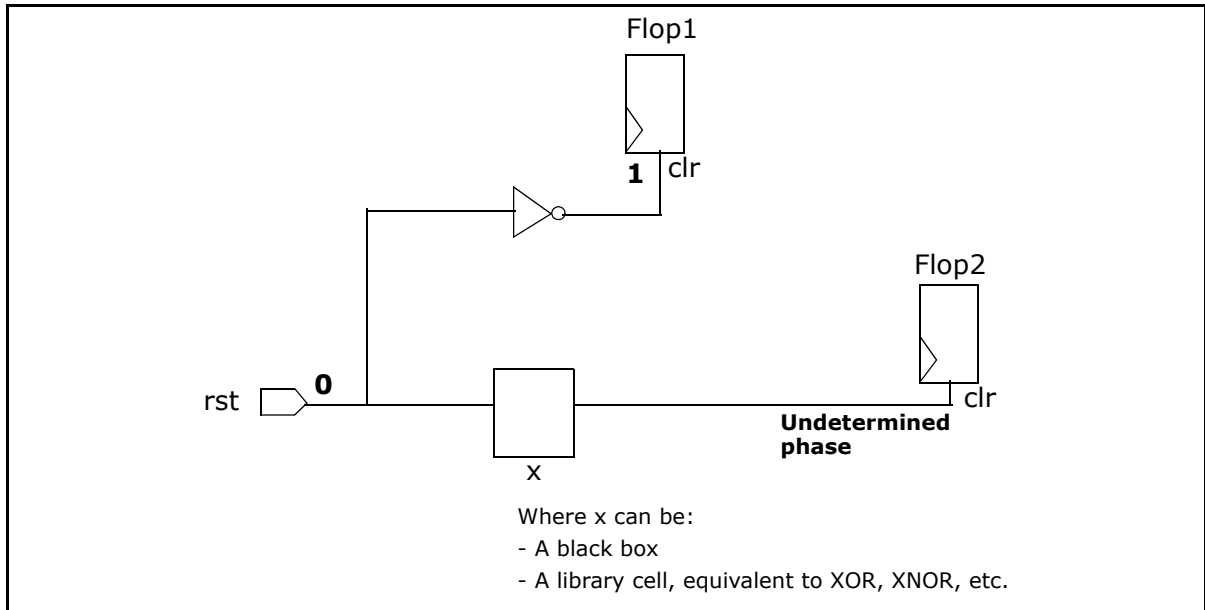


FIGURE 5. Reset_check11 tag Exception

Parameter(s)

- [*use_inferred_resets*](#): Default value is no. Set this parameter to yes to use auto-generated reset information.
- [*report_instance_pin*](#): Default value is no. Set this parameter to yes to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- [*filter_named_resets*](#): Default value is clk, clock, scan. Specify a list of strings to automatically infer asynchronous resets that do not match the specified strings.
- [*reset_reduce_pessimism*](#): Default value is filter_unused_synchronizer, same_data_reset_flop. Set the value of this parameter to all_potential_resets to detect all the potential resets in a design.

Messages and Suggested Fix

The following message appears when the asynchronous signal `<reset-name>` is used at both active high and active low levels:

[WARNING] Asynchronous reset signal '`<reset-name>`' is used as active-high at '`<inst1-name>`' and active-low at '`<inst2-name>`'

The arguments of the above message are explained below:

Argument	Description
<code><reset-name></code>	Hierarchical name of the asynchronous reset signal
<code><inst1-name></code>	In case of netlist designs, if the reset_reduce_pessimism parameter is set to <code>yes</code> , <code><inst1-name></code> is the name of the instance where the reset signal is used as active-high. Otherwise, it is the name of the output net of the instance where the reset signal is used as active-high.
<code><inst2-name></code>	In case of netlist designs, <code><inst2-name></code> is the name of the instance where the reset signal is used as active-low. Otherwise, it is the name of the output net of the instance where the reset signal is used as active-low.

Potential Issues

This violation appears if your design contains an asynchronous reset that is used at both active-high and active-low levels.

Consequences of Not Fixing

If you do not fix this violation, the design does not initialize properly. As a result, functional analysis may not work properly in this case.

How to Debug and Fix

To fix this violation, use only one active level for a particular asynchronous reset in a design.

Example Code and/or Schematic

Consider the following example:

```
module test(rst,q1,q2,d1,d2,clk);
    input rst,d1,d2,clk;
```

```

output q1,q2;
reg q1,q2;
always@(posedge clk or negedge rst) //active low clear
    if(!rst)
        q1 <= 0;
    else
        q1 <= d1;
always@(posedge clk or posedge rst)
    // active high clear
    if(rst)
        q2 <= 0;
    else
        q2 <= d2;
endmodule

```

In the above example, the asynchronous reset signal `rst` is used as active-high at `q2` and as active-low at `q1`. Therefore, the *Reset_check11* tag reports a violation.

The following figure shows the schematic of this violation:

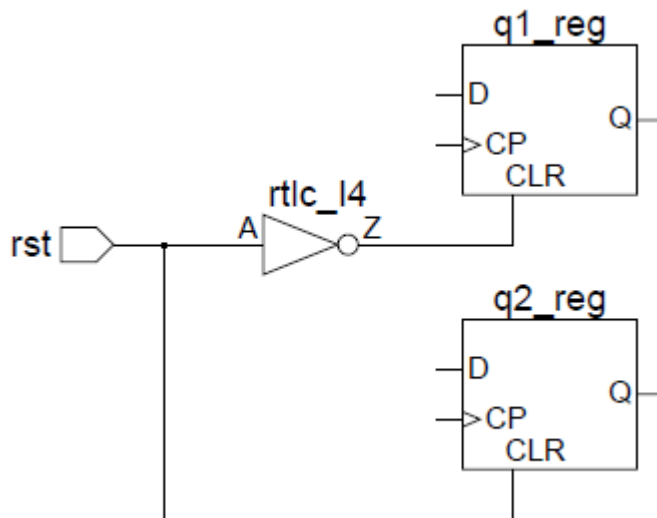


FIGURE 6. Schematic of the *Reset_check11* tag Violation

Miscellaneous Tags

Tag Severity

Warning

Reports and Related Files

No report or related file

MoreLint Tags

This section explains the MoreLint Tags in VC SpyGlass Lint.

AlwaysCombExhaustive-ML

The conditions of an if/unique-if/priority-if construct inside an always_comb block should be exhaustive.

Language

Verilog

Description

The *AlwaysCombExhaustive*-ML flags if constructs in combinational always_comb constructs where the conditions are not exhaustive.

The *AlwaysCombExhaustive*-ML flags those if constructs in a combinational always_comb construct where all if conditions (including unique-if and priority-if) of the assigned variable/signal are not covered.

If all conditions are not covered, there is a possibility of generating a latch by the synthesis tools.

Message Details

The following message appears at the location of an if construct in a combinational always_comb construct when all if conditions of the assigned variable/signal are not covered:

not all conditions are covered for the combinational logic

Consequences of Not Fixing

If all conditions are not covered, the synthesis tool may generate a latch. However, the designers do not want to infer a latch.

Additionally, this may result in additional violation at synthesis sTage because the type of the always block does not match the type of logic inferred.

Severity

Warning

AlwaysFalseTrueCond-ML

Checks the use of always-false and always-true conditions

Description

The *AlwaysFalseTrueCond-ML* Tag reports conditions that are either always true or always false.

The following conditional constructs are covered:

- case
- if
- for
- while
- ternary
- generate

This Tag supports all the unary, binary, and ternary Verilog operators.

Default Weight

10

Language

Verilog

Parameter(s)

- [*evaluate_for_loop_index*](#): The default value is no. Set the value of the parameter to yes to enable the *AlwaysFalseTrueCond-ML* Tag to consider loop index of Verilog for construct as a static value and report a violation if any of the loop iterations violates the Tag. However, if the loop index is compared with a constant value, the Tag reports a violation only if the constant value is out of the range of for loop. Also, if the for loop index is compared with a variable, Tag reports a violation only by considering the complete range of variable.
- [*report_always_cond*](#): The default value is yes and Tag reports conditions that are either always true or false. Set the value of the parameter to true or false to enable the Tag to report always-true or always-false conditions.

- *ignore_cond_having_identifier*: The default value is no. Set the value of this parameter to yes to ignore parameters, constant and macro conditions during Tag checking.
- *ignore_genvar_condition*: The default value is no. Set the value of the parameter to yes to ignore conditions associated with genvars.
- *report_less_severity*: The default value is no. In this case, all violations are reported with Warning severity. Set the value of the parameter to yes to report the parameters, constant, and macro conditions with Info severity. Others are reported with Warning severity.

Constraint(s)

None

Messages and Suggested Fix

The following message is reported when conditions are either always true or always false:

[WARNING] Condition <condition-string> can never be <true|false> in module <hierarchical-module-name>

Potential Issues

The violation message states the issue.

Consequences of Not Fixing

Not fixing the violation can result in unreachable code blocks in the design.

How to Debug and Fix

To avoid this:

- The conditions that are never true can be removed as the corresponding code will never be executed.
- The conditions that are never false can be removed as corresponding code will always be executed.

Example Code and/or Schematic

Consider the following code snippet, common for all the examples:

```
input [3:0] in;
input [3:0] in2;
```

```
input [3:0] in3;  
const int a = 1;  
int c = 1;
```

Example 1

In the following example, a violation is reported if the `report_always_cond` parameter is set to yes, both, or true because `a` is constant:

```
if(32>>a)
```

Example 2

In the following example, no violation is reported because `c` is non-static:

```
if(32>>c)
```

Example 3

In the following example, a violation is reported if the `report_always_cond` parameter is set to yes, both, or false because `in` is four bits:

```
(in > 18)
```

Example 4

In the following example, no violation is reported because `in` and `in3` are four bits:

```
(in < 18 - in3)
```

Example 5

In the following example, a violation is reported if the `report_always_cond` parameter is set to yes, both, or false because `in` and `in3` are four bits:

```
(in - 30 > 14 + in3)
```

Example 6

In the following example, a violation is reported if the `report_always_cond` parameter is set to yes, both, or false, however no violation is reported for the `in2 > in3` condition:

```
if((32<<((in2 > in3) +2)) < 0)
```

MoreLint Tags

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

AMSKeyword-ML

Verilog-AMS keyword found.

Language

Verilog

Description

The AMSKeyword-ML Tag flags use of Verilog-AMS keywords as object names.

The list of Verilog-AMS keywords checked is as follows:

above	abs	absdelay	abstol	access
acos	acosh	ac_stim	aliasparam	analog
analysis	asin	asinh	atan	atan2
atanh	branch	ceil	connect	connectTags
continuous	cos	cosh	cross	ddt
ddt_nature	ddx	discipline	discrete	domain
driver_update	endconnectTags	enddiscipline	endnature	endparameters
exclude	exp	final_step	flicker_noise	floor
flow	from	genvar	ground	hypot
idt	idtmod	idt_nature	inf	initial_step
laplace_nd	laplace_np	laplace_zd	laplace_zp	last_crossing
limexp	ln	log	max	merged
min	nature	net_resolution	noise_table	paramset
potential	pow	resolveto	sin	sinh
slew	split	sqrt	string	tan
tanh	timer	transition	units	white_noise
wreal	zi_nd	zi_np	zi_zd	

The AMSKeyword-ML Tag checks the object names in case-sensitive manner. Hence, the above keywords when used in uppercase or mixed-case are not flagged. Also, only the first occurrence of a keyword is flagged.

NOTE: *The AMSKeyword-ML Tag is switched off by default.*

Message Details

The following message appears at the location where a Verilog-AMS keyword `<keyword>` is used as an object name:

verilog-AMS keyword '`<keyword>`' found

Severity

Warning

AnsiPort-ML

The module declaration should use Verilog ANSI style ports only

Description

The *AnsiPort-ML* Tag reports violation if module declaration do not use Verilog ANSI style ports. The Tag checking is performed for all the ports of all the modules of the RTL/Netlist Verilog design. The Tag reports only one violation per module.

Language

Verilog

Messages and Suggested Fix

The following message appears when a module does not use Verilog ANSI style ports:

[2-MEDIUM] Preferably use Verilog ANSI style ports declarations in module <module_name>. [Hierarchy: <hierarchy name>]

How to Fix

Clicking the violation message points to the module header.

Example Code and/or Schematic

Example 1

Consider the following example:

```
module m1(a); //Violation
input a;
wire a;
endmodule
```

For the above example, the Tag reports a violation.

Example 2

Consider the following example:

```
module m2(a); //Violation
```



```
input wire a;  
endmodule
```

For the above example, the Tag reports a violation.

Example 3

Consider the following example:

```
module m3(a); //Violation  
input a;  
endmodule
```

For the above example, the Tag reports a violation.

Example 4

Consider the following example:

```
module m4(input a); //Violation  
wire a;  
endmodule
```

For the above example, the Tag reports a violation.

Example 5

Consider the following example:

```
module m5(input wire a); //No Violation  
endmodule
```

For the above example, the Tag does not report violation.

Example 5

Consider the following example:

```
module m6(a,b); //Violation  
input a,b;
```

```
wire a,b;  
endmodule
```

For the above example, the Tag reports one violation per header.

Default Severity Label

2-Medium

Reports and Related Files

None

ArrayPortConnect-ML

Detect port connections used in multidimensional signals with non-static indexes

Language

Verilog

Tag Description

The *ArrayPortConnect-ML* tag reports a violation for port connections that use multidimensional signals with non-static indexes.

Parameter(s)

None

Constraints

None

Message Details

Message 1

[warning]Port '<port name>' of instance '<module>:<instance>' is connected to multi dim signals '<signal name>' with non-static index '<index expression>' [Hierarchy: <hier-name>]

Potential Issues

The violation message is reported when port connections are used in multidimensional signals with non-static indexes.

Consequences of Not Fixing

Not fixing the violation may result in redundant mux selections in the synthesized netlist.

How to Debug and Fix

To debug the violation, click the violation message. VC SpyGlass highlights the RTL line with instantiation.

Examples

Example 1

Consider the following example:

```
module test(clk, rstb, addr_cns40, num_invalid, addr_r0);

input clk, rstb;
input [4:0] addr_cns40;
input [2:0] num_invalid;
output [11:0] addr_r0;
reg [4:0] addr_buf [0:7];

test_mod test_mod_0 (.A(addr_0),
.B(addr_buf[num_invalid])); //violation
endmodule
```

Example 2

Consider the following example:

```
module test(clk, rstb, addr_cns40, num_invalid, addr_r0);

input clk, rstb;
input [4:0] addr_cns40;
input [2:0] num_invalid;
output [11:0] addr_r0;
reg [4:0] addr_buf [0:7];

reg [4:0] a_sep_in;
always @(*) begin
case (num_invalid) //synopsys infer_mux_override
3'd0 : a_sep_in <= addr_buf[0];
```

MoreLint Tags

```
3'd1 : a_sep_in <= addr_buf[1];  
3'd2 : a_sep_in <= addr_buf[2];  
3'd3 : a_sep_in <= addr_buf[3];  
3'd4 : a_sep_in <= addr_buf[4];  
3'd5 : a_sep_in <= addr_buf[5];  
3'd6 : a_sep_in <= addr_buf[6];  
3'd7 : a_sep_in <= addr_buf[7];  
endcase  
end
```

```
test_mod test_mod_0 (.A(addr_0), .B(a_sep_in)); //no violation  
endmodule
```

Default Severity Label

Warning

Reports and Related Files

No related reports or files.

AsgnNextSt-ML

Next state variable is not assigned in the default state of Finite-state machine (FSM)

Language

VHDL

Description

The AsgnNextSt-ML Tag flags a violation, if the next state variable is not assigned in the default case or if there is no default case (others clause) specified for the case construct.

Next state variable should be assigned in the default state of an FSM. If the FSM somehow goes into an undesired state, then the next state variable should be assigned a proper state value to make sure it does not stay there.

Message Details

The following message appears at the location of an others clause of a case construct in an FSM description if the others clause does not assign a value to the next state variable:

Next state variable should be assigned in default state of FSM

Severity

Guideline

AsgnToOneBit-ML

Assigning a 0 or 1 (32-bits) to a 1 bit is not allowed

Language

Verilog

Description

The AsgnToOneBit-ML Tag flags assignment of 0 or 1 to single-bit wires or registers.

Assignment of 0 or 1 to a single-bit wire or register is assigning a 32-bit value to a single-bit object. Such assignments are generally coding oversights and should be removed.

Message Details

The following message appears at the location where a single-bit wire or register is assigned value 0 or 1:

1-bit is assigned a 0 or 1 (32 bits)

Severity

Warning

Examples

Consider the following example where single-bit register bit-select out[2:2] and wire o1 are being assigned value 0 and 1 respectively:

```
module top(in1, out1);  
    input in1;  
    output [2:0] out1;  
  
    reg [2:0] out1;  
  
    wire o1;  
  
    assign o1 = 0;  
  
    always@( in1 )
```

```
begin
    out1[2:2] = 1;
end
endmodule
```

For this example, SpyGlass generates the AsgnToOneBit-ML Tag message for each assignment.

AsgnOverflow-ML

Detects assignment overflow

When to Use

Use this Tag to detect overflows in assignment statements.

Description

The AsgnOverflow-ML Tag reports violation for overflows in assignment statements.

NOTE: *Since functionality of this Tag is already covered by W164a Tag of the VC SpyGlass Lint product, the AsgnOverflow-ML Tag is switched off and will be removed in next major release.*

The AsgnOverflow-ML Tag handles different assignment types as follows:

■ `<int_range> <- <int_range> OP <int_range>`

The range will be evaluated for RHS and then compared with the range of LHS.

■ `<non_int_range> <- <non_int_range> OP <non_int_range>`

The sizeof for RHS and LHS will be calculated as per the package.

■ `<non_int_range> <- <int_range> OP <int_range>`

The range will be evaluated for RHS, which will then be converted in size and the size will then be compared with the sizeof of LHS.

■ `<int_range> <- <non_int_range> OP <non_int_range>`

■ `<non_int_range> <- <int_range> OP <non_int_range>`

■ `<int_range> <- <int_range> OP <non_int_range>`

The size of RHS will be calculated from the package and then compared with size of LHS.

Here, integer types without range specified will be treated as `<non_int_range>` and the sizeof will be calculated instead of range.

NOTE: *The AsgnOverflow-ML Tag supports generate-if and generate-for blocks.*

Tag Exceptions

The subprogram body is not checked as the size of the arguments may

depend on the actuals passed in the subprogram call.

Default Weight

10

Language

VHDL

Parameter(s)

- [nocheckoverflow](#): The default value is no. Set the value of the parameter to yes to calculate the width according to the LRM (numeric_std lib). You can also specify Tag name as an argument to the parameter to enable LRM based checking. To know more about the Tags of width calculation, refer to the [nocheckoverflow](#) parameter description.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location of an assignment statement with an overflow condition:

[WARNING] LHS: '<expr1>' (<size1>) does not match RHS: '<exprr>' (<size2>) in assignment [Hierarchy: '<hier-path>']

Where, the size of LHS expression <expr1> is <size1> and the size of RHS expression <exprr> is <size2> in hierarchy <hier-path>.

Potential Issues

Violation can arise when there is an overflow condition in an assignment statement.

Consequences of Not Fixing

Overflow in assignment statements is an error condition and results in possible loss of data. Connecting a bus which is wider than the target would mean that excess bits will be ignored (if an input bus) or assigned unknown values (if an output bus). In such a case simulation as well as synthesis may fail.

How to Debug and Fix

Select the message that you want to debug.

In the HDL window, it takes you to the line, where, overflow of bits is detected.

To fix the violation, review the RTL code mentioned in message, as some data loss can occur at this point.

Example Code and/or Schematic

Consider the following example flagged by the AsgnOverflow-ML Tag:

```
signal SIG1, SIG2: std_logic_vector (2 downto 0);
signal SIG3: std_logic_vector (4 downto 0);
SIG3 <= SIG1 * SIG2;
```

Here, in the assignment statement, the sizeof RHS is 6 (3+3) whereas the sizeof LHS is 5. Thus, the size of LHS is less than the size of RHS, causing an overflow condition.

Now consider the following example that has an assignment statement of all integer types with range specified:

```
signal A1: integer range -100 to 100;
signal A2: integer range -100 to 100;
signal A3: integer range -199 to 199;
signal A4: integer range -200 to 200;
A3 <= A1 + A2;
A4 <= A1 + A2;
```

In the first assignment, the RHS (that is, A1+A2) can have values from -200 to 200 and range of LHS (that is, A3) is -199 to 199. Thus, the range of LHS is less than the range of RHS, causing an overflow condition. In the second assignment, the range of LHS (that is, A4) is -200 to 200, which is equal to range of RHS (that is, A1+A2).

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

AssignPatInInst-ML

Detect assignment patterns in parameter override or port connection area

When to Use

Use this tag to detect assignment patterns in parameter override or port connection area.

Description

The *AssignPatInInst-ML* tag reports a violation when assignment patterns are used in parameter override or port connection area of an instance.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Assignment Pattern(s)
'<comma_separated_assign_pattern_list>' used in <port connection | parameter override> area at instance
'<instance_name>' [Hierarchy: <hierarchy>]

Potential Issues

A violation is reported when assignment patterns are used in parameter override or port connection area of an instance.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

Click the violation message, SpyGlass points to the RTL lines where the assignment patterns are used in the instance (multi-line message).

Example Code and/or Schematic

Example 1

Consider the following example:

```
module test ( input x, output [7:0] y, output [7:0] z);
  sub#(.LUT('{h64, 'ha8})) // violation ('{h64,'ha8},
    '{h72, 'hcc})
  i_sub (.c ('{ 'h72, 'hcc}), .x(), .y(), .z(z));
endmodule

module sub #(parameter [1:0][7:0] LUT = '{h1, 'h0}) // no-violation, not
in an instance
  ( input [1:0][7:0] c, input x, output [7:0] y, output [7:0]
    z);
  assign y = LUT[x];
  assign z = c[x];
endmodule
```

Example 2

Consider the following example:

```
module test (input in, output out);
  Inf inst(.in_p('{h72, 'hcc})); // violation
endmodule

interface Inf(input [1:0][7:0] in_p);
  reg r;
endinterface
```

Example 3

Consider the following example:

```
module test (input in, output out);
  parameter PP = 1;
```

MoreLint Tags

```

if(PP == 0)
  mod inst('.in_p('{h72, 'hcc}), .oupt_p(out)); // no-violation, in a dead
  block
else
  mod inst('.in_p(in), .oupt_p(out));
endmodule
module mod(input [1:0][7:0] in_p, output oupt_p);
  assign oupt_p = ~in_p;
endmodule

```

Example 4

Consider the following example:

```

module test (input in, output out); // top module
  assign out = in;
endmodule

module mod (input in, output out); // unused
  mod_2 inst('.in_p('{h72, 'hcc}), .oupt_p(out)); // no-violation, in an
  unused module
endmodule

module mod_2(input [1:0][7:0] in_p, output oupt_p);
  assign oupt_p = ~in_p;
endmodule

```

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files

AutomaticFuncOnly-ML

Detects non-automatic functions

Description

The *AutomaticFuncOnly-ML* Tag reports a violations for all the functions that are not declared as automatic functions.

Language

Verilog

Messages and Suggested Fix

The following message appears when functions are not declared as automatic functions:

[WARNING] Non automatic function '<function_name>' detected

Example Code and/or Schematic

Example 1

Consider the following example:

```
module top ();  
    function logic myFunc;//violation, declared in module  
    input logic b;  
    myFunc = b;  
    endfunction  
endmodule
```

In the above example, the Tag reports a violation because function is declared in module.

Example 2

Consider the following example:

MoreLint Tags

```
module top ();  
function automatic logic myFunc; //no violation  
input logic b;  
myFunc = b;  
endfunction  
endmodule
```

In the above example, the Tag does not report a violation.

Example 3

Consider the following example:

```
function logic myFunc; //violation, declared outside module  
input logic b;  
myFunc = b;  
endfunction  
  
module top ();  
endmodule
```

In the above example, the Tag reports a violation because function is declared outside module.

Example 4

Consider the following example:

```
package test;  
function logic myFunc; //violation, declared in package  
input logic b;  
myFunc = b;
```

```
endfunction  
endpackage
```

```
module top ();  
endmodule
```

In the above example, the Tag reports a violation because function is declared in package.

Example 5

Consider the following example:

```
interface test ();  
function logic myFunc;//violation, declared in package  
input logic b;  
myFunc = b;  
endfunction  
endinterface  
module top ();  
endmodule
```

In the above example, the Tag reports a violation because function is declared in package.

Default Severity Label

Warning

Reports and Related Files

None

MoreLint Tags

AutomaticFuncTask-ML

Use automatic functions/tasks in modules and interfaces

Description

The *AutomaticFuncTask-ML* tag reports a violation if the functions/tasks inside modules and interfaces are not declared as automatic. Variables used in functions and tasks are static by default. There is risk of inadvertently overriding the variables used by them. Declaring the functions and tasks helps in creating local copies of the variables.

Language

Verilog

Parameter(s)

[*check_names_space*](#): Set the value of the parameter to *module*, *interface*, *package* to enable the *AutomaticFuncTask-ML* tag to report violations only for functions in module scope, package scope and interface scope.

Message Details

The following message is reported when the functions/tasks inside modules and interfaces are not declared as automatic:

[ERROR] <function | task> '<function-name | task-name>' should be declared as automatic [Hierarchy: <hier-path>]

Example and /or Schematic

Consider the following code snippet:

```
`define FUNC_AUTOMATIC automatic

module mod1 (clk);

function func1;
input b;
assign func1 = ~b;
```

MoreLint Tags

```
endfunction

task tsk1;
input b;
output a;
assign a = ~b;
endtask

input clk;
reg a,b,c,d;
always @(*) assign b = func1(a);
always @(*) tsk1(c,d);

endmodule
```

For the above example, the Tag reports the following violations:

Task 'tsk1' should be declared as automatic [Hierarchy: :mod1]

Function 'func1' should be declared as automatic [Hierarchy: :mod1]

Default Severity Label

Error

AvoidShortedPort-ML

Avoid connecting one net to different logical inputs of a same instance, except GND, VDD nets

Language

Verilog+VHDL

Message Details

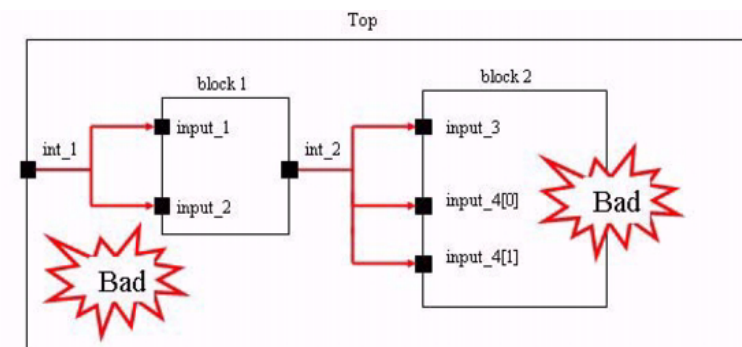
The following message appears for net `<net-name>` that is connected multiple input pins `<pin-name-list>` of instance `<inst-name>`:

Net '`<net-name>`' is driving multiple inputs `<pin-name-list>`
[Instance `<inst-name>`]

Suggested Fix

Avoid driving several inputs of module/entity by single net.

Example



Tag Severity

Note

BitDataType-ML

Bit data type signal has been used in the design.

Language

Verilog

Description

The BitDataType-ML Tag flags if a signal of bit data type is declared and used in the design.

Bit type signals support only two states, 0 or 1. Therefore, simulation synthesis mismatch may occur if the value, x, is propagated to a bit type signal.

By default, the BitDataType-ML Tag flags five messages per module. Set the [allviol](#) Tag parameter to *yes* or *<Tag-name>* to report all the violation messages.

Message Details

The following message appears at the location where a bit data type is used in the module *<module-name>*:

Bit data type signal used in the module '<module-name>'

Potential Issues

The violation message is reported when the bit data type signal is used in the design.

Consequences of Not Fixing

Not fixing the violation can result in It can result in mismatch in simulation and synthesis.

Severity

Warning

BitRangeUsedParam-ML

Bit Range Specified for Parameter.

When to Use

Use this Tag to ensure that bit range is specified for a parameter.

Description

This Tag checks whether bit range specification is used for parameters. Parameters are constants and do not belong to either the variable or the net group in Verilog. Therefore, bit range specification is not allowed.

Language

Verilog

Default Weight

10

Parameters

None

Constraints

None

Messages and Suggested Fix

[WARNING] Bit range is specified for parameter/localparam <parameter name>.

Potential Issues

The violation message is reported when bit range is specified for a parameter or localparam.

Consequences of Not Fixing

Not fixing the violation message may result in unexpected results.

How to Debug and Fix

Click the violation message to highlight the associated RTL description.

To fix the violation, review the bit range description.

Example Code and/or Schematic

Example 1

Consider the following code:

```
module top(input logic in, output logic out);
    reg b;
    parameter a = 20;

    assign b = a [0]; //Violation on "a[0]"
endmodule
```

Example 2

Consider the following code:

```
module top(input logic in, output logic out);
    reg b,c;
    parameter a = 20;

    assign b = a [5:0]; // Violation on "a [5:0]"
    assign c = a [0+:2]; // Violation on "a [0+:2]"
endmodule
```

Example 3

Consider the following code:

```
package my_pkg;
    parameter a = 20;
    parameter pb = a[5:0]; // violation

endpackage

module top(input logic in, output logic out);
    reg b;
    import my_pkg::*;
    assign b = a [5:0]; // Violation on "a [5:0]"

endmodule
```

Example 4

```
//Inc.v
```

```
parameter a = 20;

//Test.v
module top(input logic in, output logic out);
    reg b;
    `include "inc.v"
    assign b = a [0]; //Violation on "a [0]"
    assign out = in;

endmodule
```

Example 5

Consider the following code:

```
module top(input logic in, output logic out);

    function my_func2;
        reg b;
        parameter a = 345;
        assign b = a [0]; //Violation on "a [0]"
    endfunction

    assign out = in;
endmodule
```

Example 6

Consider the following code:

```
module top(input logic in, output logic out);
    parameter param_c = 46;
    parameter [5:0] param_a = param_c; //no Violation on "[5:0]
    param_a"
    assign out = in;
endmodule
```

Severity

Warning

MoreLint Tags

Reports and Related Files

None

BitOrder-ML

Bit order specification should follow recommended convention.

Language

Verilog

Description

The BitOrder-ML Tag flags array specifications that are not in the recommended format.

By default, the BitOrder-ML Tag requires that the vector size, word size, and memory size specifications are in the [MSB:LSB] format. Use the [vector_size](#), [word_size](#), or [mem_size](#) Tag parameters to check for the vector size, word size, and memory size specifications, respectively, in the [LSB:MSB] format.

Arithmetic operations on arrays work on the assumption that the arrays are defined in a definite format. Such operations on the reverse arrays require complicated conversions.

By default, this Tag detects a recommended bit order only. Set the [range_bound](#) parameter to any or a combination of the `zero_bound`, `positive_bound`, and/or `negative_bound` values to specify the type of boundary value to be used in the range index. This enables the Tag to check the negativity and positivity of boundary values. You can select multiple options as well. See the [range_bound](#) parameter section for details.

Message Details

The following message appears at the location where a vector, word, or memory `<name>` is defined when its size specification `<size>` is not in the recommended format:

Declare '`<type>`' bit order '`<size>`' as per convention

Where `<type>` can be Vector Size, Word Size, or Memory Size.

Severity

Warning

Example

Consider the following example:

```
Line1:    module TOP();  
Line2:  
Line3:    reg [0:4] vec1;  
Line4:    reg [4:0] vec2;  
Line5:    reg [4:0] mem1 [4:0];  
Line6:    reg [0:5] mem2 [4:6];  
Line7:  
Line8:    endmodule
```

In the above example, by default, the value of *vector_size*, *word_size*, and *mem_size* is set to down.

The Tag reports violation for the following:

- In Line3, for bit-order [0:4], because *vector_size* is set to down.
- In Line6, for bit-order [0:5] and [4:6], because *word_size* and *mem_size*, respectively, is set to down.

Also, the Tag does not report violation for the following cases:

- In Line4, for bit-order [4:0], because *vector_size* is set to down.
- In Line5, for bit-order [4:0] and [4:0], because both *word_size* and *mem_size*, respectively, is set down.

BreakInMultiDimForeach-ML

Reports if break or continue statement is used in a multi-dimensional foreach loop.

When to Use

Use this tag to detect break or continue statements used in a multi-dimensional foreach loop.

Description

The *BreakInMultiDimForeach-ML* tag checks if a break or continue statement is used in a multi-dimensional foreach loop.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] '<break or continue>' statement is used inside multi-dimensional '<multi-dim signal>' foreach loop

Potential Issues

The violation message is reported when a break or continue statement is used inside multi-dimensional foreach loop

Consequences of Not Fixing

Not fixing the violation may cause synthesis and simulation mismatch because different tools interpret break/continue statements differently

How to Debug and Fix

Click the violation message to highlight the RTL line of break or continue statement.

Example Code and/or Schematic

Example 1

```
module top (array);  
    output logic array[3][2];  
  
    always_comb begin  
        foreach(array[i,j]) begin  
            array[i][j] <= i + j;  
            break; // violation  
        end  
    end  
endmodule
```

Example 2

```
module top (array);  
    output logic array[3][2];  
  
    always_comb begin  
        foreach(array[i,j]) begin  
            array[i][j] <= i + j;  
            continue; // violation  
        end  
    end  
endmodule
```

Example 3

```
module top (array,out);

output logic array[3][2];
output logic out;
logic [2:0] array1;
wire sel;

always_comb begin

for(int k= 0; k < 2; k++) begin
    break;// no violation
    foreach(array[i,j]) begin
        array[i][j] <= i + j;
        break; // violation
    end
end

foreach(array[i,j]) begin
    for(int k= 0; k < 2; k++) begin
        array[i][j] <= i + j;
        break; // no violation because this is related for loop
    end
    break; // violation
end
end
endmodule
```


MoreLint Tags

Default Severity Label

Warning

Reports and Related Files

None

CAPA-ML

Copy-paste should not be done within a design file

Language

Verilog, VHDL

Description

The CAPA-ML Tag detects duplicate code sections.

By default, the CAPA-ML Tag checks for duplicate code sections as follows:

- Checks in modules (Verilog) and architectures (VHDL)

You can use the [capa_scope](#) Tag parameter to specify a different scope for checking. You can specify the scope as `module` (default) or `file`.

- Checks code sections of 5 lines or more.

Use the [capa_size](#) Tag parameter to specify a different number.

- Checks for 100% match between two code sections.

Use the [capa_similarity](#) Tag parameter to specify a different match number.

By default, the CAPA-ML Tag does not check within a block to detect duplicate sections of code. Set the [capa_checkwithinblock](#) Tag parameter to `yes` to enable Tag checking within a block.

NOTE: *By default, the CAPA-ML is switched off. Set the [runcapaTag](#) Tag parameter to run the CAPA-ML Tag.*

By default, the value of the [capa_max_line_count](#) tag parameter is set to 0 and the CAPA-ML tag checks all the available modules/architecture. And if this parameter is enabled, the tag will count the actual line count of modules/architectures by skipping empty lines and the lines that have only comments. Set the value of the [capa_max_line_count](#) tag parameter to a positive integer to specify the maximum line number count of the modules/architectures that should be considered while tag checking. If the line count of a module/ architecture is greater than the given value, then the tag will not check that module/ architecture.

Message Details

The following message appears at the first line of a code section (from line

MoreLint Tags

`<num1> to <num2>`) when a matching code section is found from line `<num3> to line <num4>`:

Duplicate code found in lines (`<num1> to <num2>`) and (`<num3> to <num4>`)

Severity

Guideline

Suggested Fix

Avoid copying of code within a design file. Prefer to use function, task or sub-block instead.

CheckAnalogPin-ML

Analog output pins should be connected to analog input pins only

Tag Description

The CheckAnalogPin-ML tag reports violation if output analog pin of a cell is not connected to input analog pin of different cell.

Tag-checking is only done if source pin direction is output and destination pin direction is input.

For <28 NM architectures, a pin is considered as an analog pin only if the `is_analog` attribute of the pin is set to `true`.

Tag Exceptions

This tag does not report violation in the following cases:

- If any analog output pin is connected to top-level port
- If an analog output pin is connected to an analog output/inout pin of a different cell
- If an analog output pin is connected to a non-analog input pin of the same cell

Language

Verilog, VHDL

Message Details

The following message appears if output analog pin of a cell is not connected to input analog pin of different cell:

Output pin <output-pin-name> of instance <hier-instance-name1> (Cell: <master-name>), type ANALOG, is connected to Input pin <input-pin-name> of instance <hier-instance-name2> (Cell: <master-name>), type <pin-type>, which is NOT Analog type

Tag Severity

Error

CGFromSpecifiedCell-ML

Clock gated only with user specified cells

Tag Description

The *CGFromSpecifiedCell-ML* tag reports a violation when a cell on the clock tree is not from user specified cell-list given in the file specified using the [clockgate_filename](#) parameter.

This tag propagates the above clock in the design and checks the cells present in the clock path. Tag checking would be similar to [CGFromSpecifiedLib-ML](#) with added support of cell names. This tag reports violation only for those cells that are present in the user-specified library but not present in the user-specified cell-list of the corresponding library. In addition, if any PAD cell is detected on a clock-tree, the tag stops traversing further and does not report any violations on that clock tree.

Sanity violation messages are reported when user-specified file could not be accessed. When user does not specify a file, violation would not be reported for any cells present in the clock path.

The tag generates a comma separated report named `ClockPathCellInfo.rpt` listing the unwanted cells used in the clock tree.

NOTE: The *CGFromSpecifiedCell-ML* tag does not report a violation for SpyGlass-generated buffers and inverters.

NOTE: The *CGFromSpecifiedCell-ML* tag does not report a violation if the fan-out of a net does not reach the clock pin of a memory.

Language

Verilog + VHDL

Parameter(s)

- [clockgate_filename](#): By default, the value of this parameter is set to `null.txt`. You can specify the name of file that contains cell names.

- [disable_seq_clock_prop](#): By default, the value of this parameter is set to no. Set this parameter to yes, to prevent automatic propagation of clocks through flops.

Message Details

The following message appears if a cell is detected in the clock tree of a clock *<clk-name>*:

```
cell '<comma separated cell-names along with the path
name>' detected in clock tree of clock '<clk-name>'
(Instance: '<inst-name>') does not belong to user specified
library
```

The following violation message (of severity Note) is reported when user-specified file can not be accessed:

User specified file *<file-name>* could not be accessed. Rule checking aborted

The following message appears if no cell is mentioned in the input file specified by using the [clockgate_filename](#) parameter:

cells from the user library are not provided. Rule checking aborted

Tag Severity

Error/Note

CGFromSpecifiedLib-ML

Clock gated only with a cell from user specified library

Tag Description

The *CGFromSpecifiedLib-ML* tag reports a violation when cell on the clock tree is not from the user-specified library given in the file specified using the *clockgate_filename* parameter.

The *CGFromSpecifiedLib-ML* tag begins fan-out traversal from user-specified clocks and checks only those cells whose fan-out traversal reaches clock pin of sequential element. It ignores behavioral logic and no violation would be reported for behavioral logic present in the clock path. In addition, if any PAD cell is detected on a clock-tree, the tag stops traversing further and does not report any violations on that clock tree.

Violation is reported for each instance of the cell present in the clock path. Sanity violation message would be reported when user-specified file cannot be accessed.

The *CGFromSpecifiedLib-ML* tag generates a comma-separated report named `ClockPathCellInfo.rpt` listing the unwanted cells used in the clock tree.

NOTE: *The CGFromSpecifiedLib-ML tag does not report a violation for SpyGlass-generated buffers and inverters.*

NOTE: *The CGFromSpecifiedLib-ML tag does not report a violation if the fan-out of a net does not reach the clock pin of a memory.*

Language

Verilog + VHDL

Parameter(s)

- *clockgate_filename*: By default, the value of this parameter is set to `null.txt`. You can specify the name of file that contains library names.

- [*disable_seq_clock_prop*](#): By default, the value of this parameter is set to no. Set this parameter to yes, to prevent automatic propagation of clocks through flops.

Message Details

The following message appears if a cell is detected in the clock tree of a clock *<clk-name>*:

```
cell '<comma separated cell-names along with the path
name>' detected in clock tree of clock '<clk-name>'
(Instance: '<inst-name>') does not belong to user specified
library
```

The following sanity violation message (of severity Note) is reported when user-specified file can not be accessed:

User specified file *<file-name>* could not be accessed. Rule checking aborted

Tag Severity

Error/Note

CheckAsyncHold-ML

Report violations for flop with an asynchronous hold.

When to Use

Use this Tag to detect flip-flops with an asynchronous hold.

Description

This Tag checks for asynchronous flops which has an asynchronous hold. This Tag detects assignments only inside always blocks, which assign a signal to the same signal inside an if, else-if or a conditional operator.

Also, Tag will report violations when the holding assignment is inside an else-if block, only if all previous if and else-if blocks have asynchronous behavior. If the holding assignment is inside a conditional operator all the conditions of conditional operators should have asynchronous behavior.

If the condition of an if, else-if block or conditional operator contains in sensitivity list, that block is considered to have an asynchronous behavior.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Asynchronous hold detected on signal '<Signal name>'.

Potential Issues

The violation message is reported when an asynchronous hold is detected on a signal.

Consequences of Not Fixing

Not fixing the violation may result in unexpected Tag behavior.

How to Debug and Fix

Click the violation message to highlight the asynchronous hold assignment in the HDL viewer.

Example Code and/or Schematic

Case 1: violating scenario with if-else

```
always @(posedge clk or posedge pause or negedge reset_n)
begin
    if(!reset_n) begin
        shift_reg <= 1'b0;
    end
    else if (pause) begin
        shift_reg <= shift_reg; // Asynchronous Hold
    end
    else begin
        shift_reg <= input_reg;
    end
end
```

Case 2: non violating scenario with if-else

```
always @(posedge clk or posedge pause) begin
    if(reset) begin
        shift_reg <= 1'b0;
    end
    else if (pause) begin
        shift_reg <= shift_reg;
    end
    else begin
        shift_reg <= input_reg;
    end
end
```

MoreLint Tags

In the above example, Tag will not report violation because all previous if, else-if blocks do not have asynchronous behavior. That is, signal reset is not in the sensitivity list

Case 3: violating scenario with conditional operator

```
always @(posedge clk or posedge pause or negedge reset_n)
begin
```

```
    shift_reg <= (!reset_n) ? 1'b0 : ((pause) ? shift_reg :
    input_reg);
```

```
end
```

Case 4: non violating scenario with conditional operator

```
always @(posedge clk or posedge pause) begin
    shift_reg <= (!reset_n) ? 1'b0 : ((pause) ? shift_reg :
    input_reg);
end
```

In the above example, Tag will not report violation because all previous conditions of conditional operators do not have asynchronous behavior. That is, signal reset_n is not in the sensitivity list.

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No reports or related files.

CheckAlwaysCombSenseList-ML

Report violations for signal which are read inside an always comb block and not in the implicit sense list of that always_comb block.

Description

This Tag checks whether the signals read inside an always_comb block are present in the implicit sensitivity list of the always_comb block. The implicit sensitivity list of an always_comb includes the expansions of the longest static prefix of each variable or select expression that is read within the block or within any function called within the block with the following exceptions:

- Any expansion of a variable declared within the block or within any function called within the block.
- Any expression that is also written within the block or within any function called within the block.

The Tag does not report violations when the a variable is written and then is read inside an always_comb block, considering it is a temporary variable. However, the Tag reports a violation if the same variable is read and then written.

Language

Verilog

Message Details

The following violation message is reported when the signal is read in the design:

Signal '<Signal Name>' is read inside always_comb block which is not present in the sensitivity list [Hierarchy: '<hierarchy>']

Severity

Warning

Examples

Example 1

```
module sr (input logic en, clk, din, sh_en, output logic
```

MoreLint Tags

```
[3:0] dout);  
logic [3:0]dout_c;  
always_ff@(negedge en, posedge clk) dout <= !en? 0 : dout_c;  
always_comb dout_c = sh_en ? {dout_c, din} : dout;  
//violation on dout_c  
endmodule
```

Example 2

```
always_comb  
begin  
    b = a;  
    c =b; //no violations for b  
end  
  
always_comb  
begin  
    c = b; //violation for b  
    b = a;  
end
```

CheckLocalParam-ML

Reports violation if parameters are used in modules or interfaces

When to Use

Use this Tag to report violation if parameters are used in the modules or interfaces

Description

In Verilog, it is recommended to use 'localparam' inside modules or interfaces instead of 'parameter'. This Tag reports a violation message if parameters are used instead of 'localparam's in modules or interfaces.

Language

Verilog

Default Weight

5

Parameters

None

Constraints

None

Messages and Suggested Fix

[WARNING] Use localparam instead of parameter to declare "<parameter name>" in module or interface "<Module or Interface name>". [Hierarchy: < Hierarchy >]

Potential Issues

The violation message is reported when parameter is used, instead of localparam, for module or interface.

Consequences of Not Fixing

How to Debug and Fix

The violation message enables you to cross-probe to the HDL description.

Click on the violation message to highlight the line where the parameter declaration inside module/interface in the design.

To fix the violation message, use localparam instead of parameter for module or interface.

Example Code and/or Schematic

Consider the following RTL code:

```
parameter ROOT_OK1 = 1;
localparam ROOT_OK2 = 2;

interface simple;
parameter NOT_OK = 0; // violation on this declaration
endinterface

module test;
parameter NOT_OK = 0; // violation on this declaration
localparam OK = 1;
endmodule
```

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No reports or related files.

CheckImplicitPort-ML

Reports if instances are using implicit port connections in the design

Description

The *CheckImplicitPort-ML* Tag reports violation if a module instance or an interface instance has been instantiated with implicit port connectivity created using `".*"` construct. The Tag considers module instances and interface instances in the design.

The Tag reports violation:

- If implicit port connection is used in a module instance
- If implicit port connection is used in an interface instances

Language

Verilog

Parameter(s)

[ignore_file_ext](#): By default, the parameter is set to no. Set the parameter to yes to enable the tag to not report a violation for the files having extensions specified through this parameter.

Messages and Suggested Fix

The following message appears when a module instance or an interface instance has been instantiated with implicit port connectivity created using `".*"` construct:

```
[WARNING] implicit port connection used in instance <instance name-string>(.*) [Hierarchy: '<hierarchy name>']
```

How to Fix

Clicking the violation message points to the line where implicit port connections are used in the design.

Example Code and/or Schematic

Example 1

```
module addr (input [1:0] in0, input [1:0] in1, output [2:0]
```


MoreLint Tags

```

sum);
    assign sum = in0 + in1;
endmodule

module top (in0, in1, in2, sum);
    input [1:0] in0, in1, in2;
    output [2:0] sum;
    addr addr (.*); //violation
endmodule

```

In the above example, the Tag reports a violation for the addr (.*).

Example 2

Consider the following example:

```

module addr (input [1:0] in0, input [1:0] in1,output [2:0]
sum);
    assign sum = in0 + in1;
endmodule

module top (in0, in1,in2, sum);
    input [1:0] in0, in1,in2;
    output [2:0] sum;
    addr addr (.*); //violation
    addr add (.in0,in1, .sum); // No Violation
    addr ad (in0, in1,sum); //No Violation
endmodule

```

In the above example, the Tag reports a violation for addr (.*) and does

not report for the 'add' and 'ad' instances.

Example 3

Consider the following example:

```
interface TestInterface (input Clk);
    logic [7:0] In1, Data;
    modport Slave (input In1, inout Data);
endinterface

module test;
    logic Clk;
    TestInterface TheBus(.*); //violations
    tempModule TheRAM (.Bus(TheBus.Slave));
endmodule

module tempModule (TestInterface.Slave Bus);
endmodule
```

In the above example, the Tag reports a violation for the 'TheBus(*)' interface instance.

Example 4

Consider the following example:

```
primitive udp_body (a,b,c);
output a;
input b,c;
table
    ? 1      : 1;
```

MoreLint Tags

```

        1  ?      : 1;
        0  0      : 0;
    endtable
endprimitive

module udp_body_tb();
    reg b,c;
    wire a;
    udp_body udp (a,b,c);
endmodule

```

In the above example, the Tag does not report for UDP instances.

Example 5

Consider the following example:

```

module addr (input [1:0] in0,input [1:0] in1,output [2:0]
sum);
    assign sum = in0 + in1;
endmodule

module top (in0, in1,in2, sum);
    input [1:0] in0, in1,in2;
    output [2:0] sum;

    addr add (.in0(in0),.*); //violation
endmodule

```

In the above example, the Tag reports a violation for the 'add' instance.

Example 6

Consider the following example:

```
`include "top.v"
module test(input shortint in1,in2, output int out1);
assign out1 = ((1{in1,in2}))+1);
endmodule

top.v :
module addr (input [1:0] in0,input [1:0] in1,output [2:0]
sum);
    assign sum = in0 + in1;
endmodule

module top (in0, in1,in2, sum);
    input [1:0] in0, in1,in2;
    output [2:0] sum;

    addr add (.*) ; //violation
endmodule
```

In the above example, the Tag reports a violation for add(.*) instance in the top.v file.

Example 7

Consider the following example:

MoreLint Tags

```

module addr (input [1:0] in0,input [1:0] in1,output [2:0]
sum);
    assign sum = in0 + in1;
endmodule

module top (in0, in1,in2, sum);
    input [1:0] in0, in1,in2;
    output [2:0] sum;
    genvar itr;
    generate
        for (itr = 1 ; itr <= 2; itr = itr+1)
            addr aa(.*) ; //violation
    endgenerate
endmodule

```

In the above example, the Tag reports a violation for the aa(.*) instance in the generate for.

Example 8

Consider the following example:

```

module addr (input [1:0] in0,input [1:0] in1,output [2:0]
sum);
    assign sum = in0 + in1;
endmodule

module top (in0, in1,in2, sum);
    input [1:0] in0, in1,in2;
    output [2:0] sum;

```

```
if (1)
    addr a(.*); // violation
else
    addr b(.*); // no violation
endmodule
```

In the above example, the Tag reports a violation for the a(*) instance in the if condition because and does not report violation for the dead code in the else part b(*)).

Example 9

Consider the following example:

```
module addr (input [1:0] in0,input [1:0] in1,output [2:0]
sum);
    assign sum = in0 + in1;
endmodule

`define MACRO 1
module top (input [1:0] in0,input [1:0] in1,output [2:0]
sum);
    `ifdef MACRO
        addr a(.*); // violation
    `endif
Endmodule
```

In the above example, the Tag reports a violation for the for macro cases.

Default Severity Label

Warning

MoreLint Tags

Reports and Related Files

None

CheckModulesWithoutPorts-ML

Reports violation for modules without any port.

When to Use

Use this Tag to check for modules without any port.

Description

The *CheckModulesWithoutPorts-ML* Tag reports violation for modules that have no port declarations.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[ERROR] modules without any port should not be specified
[Hierarchy: <module hierarchy>]

Potential Issues

The violation message is reported when the design has modules without port declarations.

Consequences of Not Fixing

NA

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the module declaration in the design.

To fix the violation, ensure the module has the respective port declaration.

Example Code and/or Schematic

Example 1

Consider the following snippet:

```
module s_209 ();    //violation no port

wire in1 [31:0] [6:0];
reg out1 [31:0] [7:0];

always begin
    for (int i=0; i<4; i++)
        begin
            for(int j=5;j<4; j++)
                begin
                    out1[i][7] = 1'b0;
                    out1[i][6:0] = in1[i];
                end
            end
        end
    end
endmodule
```

In the above example, the Tag reports a violation for the s_209 module as the module does not have a port declaration.

Example 2

Consider the following snippet:

```
module s_209 (in1, out1); //no violation
input wire in1 [31:0] [6:0];
output reg out1 [31:0] [7:0];

always begin
    for (int i=0; i<4; i++)
        begin
            for(int j=5;j<4; j++)
                begin
                    out1[i][7] = 1'b0;
                    out1[i][6:0] = in1[i];
                end
            end
        end
    end
```

```
        end  
    end  
endmodule
```

In the above example, the Tag does not report a violation as the s_209 module includes the port declaration.

Example 3

Consider the following snippet:

```
module test_top(); //violation no ports  
subcomponent subcomponent_instance_name ( );  
endmodule  
  
module subcomponent(); //violation no ports  
endmodule
```

In the above example, the Tag reports a violation as the modules, test_top and subcomponent have no port declaration.

Default Severity Label

None

Tag Group

Not Applicable

Reports and Related Files

No reports or related files.

CheckPortType-ML

Complex port types are not allowed in module interface

When to Use

Use this tag to check complex port types in the module interface.

Description

The *CheckPortType-ML* tag reports violations when the following port types are found in the module interface:

- SV port types
 - ☐ Struct
 - ☐ Union
 - ☐ Interface Mod-Pod
 - ☐ Interface
 - ☐ Scope Variable
- Multi-dimensional port types
Example: `a [1:0] [1:0]`
- Scalar port types with zero range dimension
Example: `a [0:0]`

The tag results in better RTL by only allowing scalar or single dimension Verilog style array for design top ports specification. It detects such scenarios and notifies you of the complex port types used in the module interface.

Parameter(s)

- *check_all_modules*: By default, the parameter is set to `no` and the tag checks for complex port types only at the top module interface. Set the parameter to `yes` to enable the tag to check on all the module levels.
- *ignore_dim_port_types*: By default, the parameter is set to `no`. Set the value of this parameter to `yes` to not report violations for multi-dimensional ports and zero-range ports.

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Port '<port_name>' of type '<port_type>' is not allowed in module '<module_name>' port declaration

Potential Issues

The violation message is reported when complex port types are used in the module interface.

Consequences of Not Fixing

Not fixing the violation may result in complex debugging issues.

How to Debug and Fix

Click the violation message to highlight the RTL line where a complex port type is used.

Example Code and/or Schematic

Example 1

```
typedef struct {  
  logic [3:0] data;  
  reg [4:0] addr;  
  logic [21:0] inst;  
} instr;  
typedef union  
{  
  logic [4:0][3:0] data;  
  reg [4:0] addr;  
  instr IW0[1:0];  
  logic [21:0] inst;
```

MoreLint Tags

```

    } uni;
    interface chip_bus();
    logic [2:0]d1;
    reg d2 [3:0];
    endinterface
    module top#(parameter width = 20481) (
        chip_bus bus,// violation ('Interface')
        input uni un,// violation ('Union')
        input instr IW [7:0],// violation ('Struct')
        inout d1,// No violation (sclar)
        output reg [1:0][1:0]array,// violation ('Multi-Dim')
        output [3:0] addr,// No violation (single dimension)
        output [0:0] addr1,// violation ('Scalar Zero Range
        Dimension')
        output [2:0] dout[3:0][4:0]); // violation ('Multi-Dim')
    endmodule

```

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No reports or related files.

CheckRegAndWire-ML

All reg / wire uses logic, but excluding multi-drive signals

When to Use

Use this Tag to detect registers and wires in the design.

Description

The CheckRegAndWire-ML Tag reports violations for registers and wires used in the design.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Signal '<signal name >' is declared as '<data type >'. [Hierarchy: '<hierarchy>']

Potential Issues

The violation message is reported for all the registers and wires in the design.

Consequences of Not Fixing

None

How to Debug and Fix

Click the violation message to highlight the line in the RTL where the instance is present.

Example Code and/or Schematic

Example 1

```
module TOP(input in, output out);
```

MoreLint Tags

```

        wire in; //violation
        reg out; //violation
    endmodule

```

Example2

```

module t21211 (input a, output z,output pin);
    wire a; // violation
    reg pin; // violation

    assign z = a;
    always @(*)
        pin = a ;

endmodule

```

Example 3

```

module abc();
    wire a,c; //violation
    reg x,y,v; //violation
    reg z; //violation

    always@(c)
    begin
        x <= a;
        z = y;
    end
endmodule

```

Example 4

```

module assignment1();
    int i;
    wire out1[3:0]; //ve_array //violation
    wire [3:0]out2; //ve_net //violation
    wire [3:0]out3[3:0]; //ve_array //violation

```

```
wire [3:0][3:0]out4; //ve_net //violation
wire out5[3:0][3:0]; //ve_array //violation
wire [3:0][3:0]out6[3:0]; //ve array //violation

assign out1[0] = 1'b1;

assign out2[0] = 1'b1;
assign out2[1:0] = 2'b11;
assign out2 = 3'b111;

assign out3[0] = 4'b1111;
assign out3[0][1:0] = 2'b11;
//assign out3 = logic1;

assign out4[0] = 4'b1111;
assign out4[0][1:0] = 2'b11;

assign out5[0][0] = 1'b1;

assign out6[0] = out4;
assign out6[0][0][1:0] = 2'b11;
endmodule
```

Example 5

```
interface x1;
wire d,clk; //violation
reg q,q1; //violation

modport flop(input d,clk, output q);
modport flop1(input d,clk, output q1);
endinterface

module dflop(x1.flop i);
wire x;
always@(posedge i.clk)
i.q<=i.d;
```


MoreLint Tags

```
always@(negedge x);  
endmodule
```

Example 6

```
module conditional_operator();  
    wire out; //violation  
    reg enable,data; //violation  
    assign out = (enable) ? data : 1'b0;  
endmodule
```

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No reports or related files.

CheckShiftOperator-ML

Logical and arithmetic shift operation detection on signed and unsigned operand(s) or expression(s) respectively

Language

Verilog

Description

The CheckShiftOperator-ML Tag reports a violation when a logical (left or right) shift operation is done on a signed operand and an arithmetic (left or right) operation is done on an unsigned operand. The operand can be a simple net or an expression.

The CheckShiftOperator-ML Tag conforms to the following guidelines to check whether the resulting expression or an operand is signed or unsigned:

- Decimal numbers are signed.
- Based numbers are unsigned, except where the 's' notation is used in the base specifier, such as, in 4'sd12.
- Bit-select results are unsigned, regardless of the operand.
- Part-select results are unsigned, regardless of the operand. This is true even if the part-select specifies the entire vector.
- Concatenate results are unsigned, regardless of the operand.
- Comparison results (1, 0) are unsigned, regardless of the operand.
- Real numbers converted to integers by type conversion are signed.
- Expression having both signed and unsigned operands are treated as unsigned.

Message Details

The following message appears when the CheckShiftOperator-ML Tag detects a logical or arithmetic shift operation on signed and unsigned operands or expressions, respectively:

```
<Shift_operation_type> shift operation applied on  
<operand_type> operand '<operand_name>' in module '<mod_name>'
```

Where,

- `<shift_operation_type>` represents an arithmetic or logical shift operation.
- `<operand_type>` represents the sign of the operand, which is unsigned for arithmetic operation and signed for logical operation.
- `<operand_name>` is the name of the operand on which the shift operation is performed.
- `<mod_name>` is the name of the module to which the operand belongs.

Severity

Warning

Examples

Logical Shift

Consider the following examples for the logical shift operation:

Example 1:

```
reg signed [3:0] a;
out = a << 4;
```

In the above case, the CheckShiftOperator-ML Tag flags a violation for signal, a.

Example 2:

```
reg signed [3:0] a;
reg [3:0] b;
reg [3:0] c;
out = (a+(b-c)) << 6;
```

In the above case, the CheckShiftOperator-ML Tag does not flag any violation for the signed signal, a.

Example 3:

```
reg signed [3:0] d;
reg [3:0] b, c, a;
out = ((a + b)+c - {d})>>4;
```

In the above case, the CheckShiftOperator-ML Tag does not flag any violation for the signed signal, d.

Example 4:

```
reg signed [3:0] a;  
reg [3:0] b, c;  
out = (c - b) >> a;
```

In the above case, the CheckShiftOperator-ML Tag does not flag any violation.

Example 5:

```
reg signed a;  
reg c;  
out = $signed(c) >> 2;
```

In the above case, the CheckShiftOperator-ML Tag flags a violation for the signed signal, c.

Arithmetic Shift

Consider the following examples for the arithmetic shift operation:

Example 1:

```
reg [3:0] a;  
out = a <<< 4;
```

In the above case, the CheckShiftOperator-ML Tag flags a violation for signal, a.

Example 2:

```
reg signed [3:0] a;  
reg signed [3:0] b;  
reg [3:0] c;  
out = (a + (b - c)) <<< 6;
```

In the above case, the CheckShiftOperator-ML Tag flags a violation for the unsigned expression, (a + (b - c)).

Example 3:

```
reg signed [3:0] a, b, c;  
reg [3:0] d;  
out = ((a + b) + c - (d)) >>> 4;
```

In the above case, the CheckShiftOperator-ML Tag flags a violation for the unsigned signal, d.

Example 4:

```
reg [3:0] a;  
reg signed [3:0] b, c;  
out = (c - b) <<< a;
```

In the above case, the CheckShiftOperator-ML Tag does not flag any violation.

Example 5:

```
reg signed a;  
reg c  
out = $unsigned(a) >>> 3;
```

In the above case, the CheckShiftOperator-ML Tag flags a violation for the unsigned signal, a.

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No reports or related files.

CheckSignedFunc-ML

Most significant bit should not be concatenated with 0 or 1 in signed function.

When to Use

Use this Tag to detect the expressions that are concatenated with 0 or 1 in signed function.

Description

The CheckSignedFunc-ML Tag reports violations for expressions, which are concatenated with 0 or 1 in signed function.

Language

VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Expression <Expression body> is concatenated with <0 or 1>.

Potential Issues

The violation message is reported for expressions, which are concatenated with 0 or 1 in the signed function.

Consequences of Not Fixing

Not fixing the violation may result in unexpected Tag behavior.

How to Debug and Fix

To debug the violation message, click the violation message. This will highlight the violating instance in the HDL viewer.

To fix the violation,

Example Code and/or Schematic

Consider the following RTL code:

```
entity signed_unsigned is
  port (
    i_rst_l : in std_logic;
    i_clk    : in std_logic;
    i_a      : in std_logic_vector(4 downto 0);
    i_b      : in std_logic_vector(4 downto 0)
  );
end signed_unsigned;
architecture behave of signed_unsigned is
  signal rs_SUM_RESULT : signed(4 downto 0) := (others =>
'0');
  signal ru_SUM_RESULT : unsigned(4 downto 0) := (others =>
'0');
  signal rs_SUB_RESULT : signed(4 downto 0) := (others =>
'0');
  signal ru_SUB_RESULT : unsigned(4 downto 0) := (others =>
'0');

begin
  p_SUM : process (i_clk, i_rst_l)
  begin
    if i_rst_l = '0' then          -- asynchronous reset
      (active low)
        rs_SUM_RESULT <= (others => '0');
        ru_SUM_RESULT <= (others => '0');
    elsif rising_edge(i_clk) then

      ru_SUM_RESULT <= unsigned('0' & i_a) + unsigned(i_b);
      --No Violation for unsigned function
      rs_SUM_RESULT <= signed('0' & i_a) + signed(i_b);--
      Violation
      rs_SUM_RESULT <= signed('1' & i_a) + signed(i_b); --
      Violation
      rs_SUM_RESULT <= signed(i_a & '1') + signed(i_b); --No
      Violation
```

```
        rs_SUM_RESULT <= signed(i_b) + signed('1' & i_a); --  
Violation  
        rs_SUM_RESULT <= signed(i_b) + signed(i_a & '1' & i_b);  
--No Violation  
    end if;  
    end process p_SUM;  
end behave;
```

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No reports or related files.

CheckTimeUnitandPrecision-ML

Delay used without timeunit and timeprecision in module

Description

The *CheckTimeUnitandPrecision-ML* Tag reports a violation when `timeunit` and `timeprecision` are not used in a module if intra- or inter-delays are used in a module.

Message Details

The following message appears violation when `timeunit` and `timeprecision` are not used in a module if intra or inter delays are used in a module:

[WARNING] Delay used without timeunit and timeprecision in module '<module-name>'

Examples

Consider the following example code:

```
module test1;
    timeunit 1ns;
    timeprecision 100ps;
    wire foo;
    assign #5 foo = 1'b0; // no violation
endmodule

module test2;
    wire foo;
    assign #5 foo = 1'b0; // violation
endmodule
```

Default Severity Label

Warning

ChkCarriageReturn-ML

Carriage return should be last line of a file

Language

Verilog, VHDL

Description

The ChkCarriageReturn-ML flags design files where the last line is not an empty line.

Message Details

The following message appears when the last line in a design file is not an empty line:

```
carriage return is not the last line of the file
```

Severity

Warning

CheckAssignToVecBits-ML

Use explicit assignments only for bit-selects, part-selects and full vectors.

Language

VHDL

Description

The CheckAssignToVecBits-ML Tag flags assignments to bit-selects, part-selects, and full vectors using concatenation as in the following examples:

```
library ieee;
use ieee.std_logic_1164.all

...
Vectored_net[31:30] <= 2'b0; --correct
Vectored_net[29:20] <= count10[9:0]; --correct
Vectored_net[19:0] <= count20[19:0]; --correct
ASSIGN vectored_net <= 2'b0 & count10 & count20;
...
```

NOTE: *The CheckAssignToVecBits-ML Tag is switched off by default.*

Message Details

The following message appears for an assignment to a bit-select, a part-select, or a full vector using concatenation:

Assignment to bit-selects, part-selects, and full vectors
should not be done using concatenation

Severity

Warning

Suggested Fix

Avoid using concatenation in assignments to bit-selects, part-selects, and full vectors.

CheckDelayTimescale-ML

Delay used without timescale compiler directive

When to Use

Use this Tag to detect delays used without timescale compiler directive in the design.

Description

The CheckDelayTimescale-ML Tag reports a violation if a delay is used in a module and no timescale units are specified using the 'timescale compiler directive.

The Tag reports all the modules that are described above the timescale declaration and use delay as the scope of a timescale starts from where it is specified.

Since the scope of a timescale starts from where it is specified, therefore, all the modules that are described above the timescale declaration and that use delay are reported.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a delay is specified without a 'timescale compiler directive.

[WARNING] Delay used without timescale compiler directive

Potential Issues

Violation may arise when a delay is used without timescale compiler directive in the design.

Consequences of Not Fixing

Using a delay without a timescale compiler directive may lead to simulation-synthesis mismatch. It can also cause difference in values for different simulation tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the instance where the delay is used without the declaration of the timescale directive.

To fix the violation, specify a timescale compiler directive for a delay.

Example Code and/or Schematic

Consider the following example:

```
module test;  
  wire foo;  
  assign #5 foo = 1'b0;  
endmodule  
`timescale 1ns/100ps  
module top;  
  wire foo;  
  assign #5 foo = 1'b0;  
endmodule
```

In the above example, the CheckDelayTimescale-ML Tag reports a violation as no timescale compiler is specified.

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No reports or related files.

CheckParamSensList-ML

Always block uses parameters in the sensitivity list

Language

Verilog

Description

The CheckParamSensList-ML Tag flags parameters used in the always construct's sensitivity list.

Use the following parameter to customize the Tag behavior:

strict: Default value is `no` and this Tag does not report a violation for *localparam*. Set the *strict* parameter to `yes` to report violations for *localparam*.

Message Details

The following message appears at the location where parameter `<param>` is used in the sensitivity list of an always construct:

Parameter '`<param>`' found in the sensitivity list

Severity

Warning

CheckSyncReset-ML

Synthesis directive `sync_set_reset` must be set for all synchronous reset description.

Language

Verilog

Description

The CheckSyncReset-ML Tag flags synchronous reset descriptions without the synthesis directive, `sync_set_reset`.

Message Details

The following message appears at the location where the synchronous reset signal `<rst-name>` is used without the `sync_set_reset` directive set:

```
sync_set_reset directive is not set for synchronous reset  
'<rst-name>'
```

Severity

Warning

CheckSynthPragma-ML

Pragma option must be set on the command line if `translate_on/off` blocks are present in a testcase.

Language

Verilog, VHDL

Description

The CheckSynthPragma-ML Tag flags `translate_on/translate_off` pragma specifications of unspecified non-default pragma family.

The CheckSynthPragma-ML Tag flags a `translate_on/translate_off` pragma specification when the pragma family name is not one of the default family names (`synopsys` for Verilog and `synopsys` and `pragma` for VHDL) and has not been specified with the `-Xspyglass_pragma=synthesis` command.

Message Details

The following message appears at the location of a `translate_on/translate_off` pragma specification with an unspecified non-default pragma family name `<name>`:

pragma '`<name>`' required on command line for `translate_off/on`

Severity

Warning

ChkSensExprPar-ML

Expression uses parenthesis '()' in the sensitivity list

Language

Verilog

Description

The ChkSensExprPar-ML Tag flags sensitivity list expressions that use parenthesis.

Message Details

The following message appears at the location of a sensitivity list that contains an expression `<expr>` using parenthesis:

Expression '`<expr>`' uses parenthesis '()' in the sensitivity list

Severity

Warning

Examples

The following example uses parenthesis in the sensitivity list and hence, is flagged by the ChkSensExprPar-ML Tag as highlighted:

```
module test(in, clk, reset, out);  
    input in, clk, reset;  
    output out;  
  
    reg out;  
  
    always @ (posedge(clk) or negedge(reset))  
        out = in;  
  
endmodule
```

For the above example, the ChkSensExprPar-ML Tag generates the following two messages:

Expression 'clk' uses parenthesis '()' in the sensitivity list
Expression 'reset' uses parenthesis '()' in the sensitivity list

CheckSwappedIndex-ML

Do not use bus signals with swapped indexes while instantiation

Language

Verilog

Tag Description

The CheckSwappedIndex-*ML* tag reports violation when the indexes of the bus signal passing in the instantiation are swapped relative to the port indexes of the module.

Parameter(s)

None

Constraints

None

Message Details

The following message appears when the indexes of the bus signal passing in the instantiation are swapped relative to the port indexes of the module:

```
[warning] Indexes of the bus signal 'sig_name' are swapped  
relative to the indexes of corresponding port 'port_name' in  
the instance 'instance_name' of module 'module_name'.  
[Hierarchy: 'hierarchy']
```

Potential Issues

The violation message is reported, when the bus signals are used with swapped indexes during instantiation.

Consequences of Not Fixing

Not fixing the violation may result in incompatible read/writes.

How to Debug and Fix

To debug the violation, click the violation message. VC SpyGlass highlights the RTL line of the instance having the swapped indexes.

Examples

Example 1

Consider the following example:

```
module sub(  
  input logic [5:0][1:0] sub_in,  
  output logic [1:0][5:0] sub_out  
);  
assign sub_out = sub_in;  
endmodule  
  
module test(  
  input logic [1:0][5:0] mod_in,  
  output logic [1:0][5:0] mod_out  
);  
sub inst(.sub_in(mod_in), .sub_out(mod_out)); //  
violation    ([5:0][1:0] sub_in != [1:0][5:0] mod_in)  
endmodule
```

Example 2

Consider the following example:

```
module sub(  
  input logic [1:0] sub_in [5:0],  
  output logic [1:0] sub_out [5:0]  
);  
assign sub_out = sub_in;  
endmodule  
  
module test(  
  input logic [5:0] mod_in [1:0],
```

MoreLint Tags

```
output logic [1:0] mod_out [5:0]
);
sub inst(.sub_in(mod_in), .sub_out(mod_out)); // this will be a syntax
error. index swapping between packed & unpacked is illegal (hence not
going to consider it in this rule)
endmodule
```

Example 3

Consider the following example:

```
module sub(
input logic sub_in [2:0][3:0],
output logic sub_out [2:0][3:0]
);
assign sub_out = sub_in;
endmodule
```

```
module test(
input logic mod_in [3:0][2:0],
output logic mod_out [2:0][3:0]
);
```

```
sub inst(.sub_in(mod_in), .sub_out(mod_out)); // this will be a syntax
error. index swapping in unpacked part is illegal (hence not going to
consider it in this rule)
```

```
endmodule
```

Example 4

Consider the following example:

```
module sub(
input logic [5:0][1:0] sub_in,
output logic [1:0][5:0] sub_out
);
```

```
assign sub_out = sub_in;
endmodule
```

```
module test(
input logic [3:0][5:0] mod_in,
output logic [1:0][5:0] mod_out
);
sub inst(.sub_in(mod_in), .sub_out(mod_out)); // no-violation (this is a
width mismatch, hence not going to consider it in this rule)
endmodule
```

Example 5

Consider the following example:

```
module sub(
input logic [3:0][2:0][1:0] sub_in,
output logic [1:0][5:0] sub_out
);
assign sub_out = sub_in;
endmodule

module test(
input logic [3:0][4:0] mod_in,
output logic [1:0][5:0] mod_out
);
sub inst(.sub_in(mod_in), .sub_out(mod_out)); // no-violation ( range
count mismatch - Index elements(packed) of [3:0][4:0] mod_in are not
matched with [3:0][2:0][1:0] sub_in , hence not going to consider it in this
rule)
endmodule
```

ChkUndefMacro-ML

Macro is not defined before undefining it.

Language

Verilog

Description

The ChkUndefMacro-ML flags macros that are undefined before they are defined.

The ChkUndefMacro-ML checks the usage of the `undef` statement with respect to the `define` statement.

Message Details

The following message appears at the location where a macro `<name>` is being undefined when the macro definition statement was not yet encountered:

Improper usage of 'undef' for macro '<name>'

Severity

Info

Suggested Fix

Do not un-define a macro before defining it.

ConstBitCase-ML

Detect constant bits in case expression and case items

Language

Verilog

Tag Description

The *ConstBitCase-ML* tag reports violation for the following scenarios:

- When the bit value is constant in all case items and has the same value as the case expression.
- When the bit value of the case item is fixed and conflicts with the value of the case expression.

Parameter(s)

None

Constraints

None

Message Details

Message 1

[warning]Redundant constant bit(s) are present in case expression '<case_expression>' at '<bit position>' [Hierarchy: <hier-name>]

Message 2

[warning]Constant bit(s) at '<bit position>' in case item '<case item>' are not matched with case expression '<case_expression>' [Hierarchy: <hier-name>]

Potential Issues

The violation message is reported for the following:

- When the bit value is constant in all case items and has the same value as the case expression.

- When the bit value of the case item is fixed and conflicts with the value of the case expression.

Consequences of Not Fixing

Not fixing the violation may add unnecessary resources in synthesized netlist.

How to Debug and Fix

To debug the violation, click the violation message. VC SpyGlass highlights the RTL line that has the case block.

Examples

Example 1

Consider the following example:

```
module test(i1, i2, i3, i4, i5, i6, i7, i8, sel, outt);
input  [3:0] i1, i2, i3, i4, i5, i6, i7, i8;
input  [5:0] sel;
output [3:0] outt;
reg [3:0] outt1;
always @(*) begin
    case ({s1,s2,s3,1'b0,s5,s6}) // Violation - 3rd bit is
    constant
        6'b1000_00 : outt1 = i1;
        6'b0100_00 : outt1 = i2;
        6'b0010_00 : outt1 = i3;
        6'b0000_10 : outt1 = i5;
        6'b0000_01 : outt1 = i6;
        default : outt1 = 4'h0;
    endcase
end
endmodule
```

Example 2

Consider the following example:

```
case({s1,s2,s3,1'b0,s5,s6})//No violation
```

```
  A : outt1 = i1;
  B : outt1 = i2;
  C : outt1 = i3;
  D : outt1 = i5;
  default : outt1 = 4'h0;
endcase
```

```
module test(i1, i2, i3, i4, i5, i6, i7, i8, s1, s2, s3, s5,
s6, outt);
input  [3:0] i1, i2, i3, i4, i5, i6, i7, i8;
input  s1,s2,s3,s5,s6;
output [3:0] outt;
reg [3:0] outt1;
always @(*) begin
case({s1,s2,s3,1'b0,s5,s6})
6'b1000_00 : outt1 = i1;
6'b0100_00 : outt1 = i2;
6'b0010_00 : outt1 = i3;
6'b0000_10 : outt1 = i5;
6'b0000_01 : outt1 = i6;
6'b0001_00 : outt1 = i7; //violation - 3rd bit conflict
with case expression
default : outt1 = 4'h0;
endcase
end
```

MoreLint Tags

endmodule

Default Severity Label

Warning

Reports and Related Files

No related reports or files.

ConstDrivenNet-ML

All internal net of module should not be assigned a constant value.

Language

Verilog, VHDL

Description

The ConstDrivenNet-ML Tag flags internal nets that are assigned a locally static constant value.

The ConstDrivenNet-ML Tag flags internal nets that are assigned a constant value at the time of declaration or in an assignment statement (Verilog) or in a concurrent assignment statement (VHDL).

The ConstDrivenNet-ML Tag does not check assignment statements in sequential descriptions.

Message Details

The following message appears at the location where wire *<name>* is assigned a locally static constant value *<value>*:

Verilog

```
wire <name> is assigned a constant value <value>
```

VHDL

```
signal <name> is assigned a constant value <value>
```

Severity

Info

ConstWithoutValue-ML

Avoid constants with unspecified value (deferred constants).

Language

VHDL

Description

The ConstWithoutValue-ML Tag flags deferred constants as in the following examples:

```
library ieee;  
use ieee.std_logic_1164.all  
  
constant with_val :  
    std_logic_vector(2 downto 0) := "010";  
constant no_val :  
    std_logic_vector(2 downto 0);
```

NOTE: *The ConstWithoutValue-ML Tag is switched off by default.*

Message Details

The following message appears at the location where a deferred constant `<const-name>` is declared:

Avoid deferred constant '`<const-name>`'

Severity

Warning

Suggested Fix

Avoid deferred constants if your design methodology does not allow them.

CloseCaseWithX-ML

Others clause does not have X assignment to destination signals

Language

VHDL

Description

The CloseCaseWithX-ML Tag flags signal/variable assignments in the others clause, if the others clause does not have X assignment to destination signals/variables.

All case statements should be closed with

```
when others => <dest-signal-name> <= 'X';
```

rather than with the following statement

```
when others => null;.
```

This methodology helps in saving area and power when all the possible choices are not covered. This methodology also allows X propagation.

Examples

Consider the following examples:

```
case i is
  when 0 => d1 <= '0';
  when 1 => d2 <= '1';
  when others =>
    d1 <= 'X';
    d2 <= '1';
    d3 <= '0';
end case
```

In the above example, d2 and d3 are not assigned X in the Others clause. Hence, these are flagged.

```
case i is
  when 0 => d1 <= '0';
  when 1 => d2 <= '1';
  when others =>
    d1 <= 'X';
```

MoreLint Tags

```

        d2 <= 'X';
    end case

```

In the above example, no violation is reported as X is assigned to destination signals in the others clause.

```

case i is
    when 0 => d1 <= '0';
    when 1 => d2 <= '1';
    when others =>
end case

```

```

case i is
    when 0 => d1 <= '0';
    when 1 => d2 <= '1';
    when others => NULL;
end case

```

In the above examples, X is not assigned in the others clause. Hence, violation is reported.

```

case i is
    when 0 => d1 <= '0';
    when 1 => d2 <= '1';
end case

```

In the above example, the others clause is not specified for the case construct. Hence, violation is reported.

Message Details

The following message appears at the location where the case construct does not have 'X' assignment to signal *<sig-name>*:

Case does not have 'X' assignment to signal '*<sig-name>*'

The following message appears at the location where the case construct does not have others clause for signal *<sig-name>*:

Case does not have others clause for signal '*<sig-name>*'

Severity

Warning

Suggested Fix

Close case with when others => dest_signal <= 'X'

ComplexExpr-ML

Complex Expression should be avoided.

Language

Verilog, VHDL

Description

The ComplexExpr-ML flags expressions with very large number of operators.

By default, the ComplexExpr-ML flags expressions with more than 8 operators. Use the [OperatorCount](#) Tag parameter to specify a different limit.

NOTE: *During normal and precompile run, the ComplexExpr-ML Tag shows different line numbers for expression nodes.*

Message Details

The following message appears at the location of an expression `<expr>` that contains `<num>` operators when `<num>` exceeds the value of the OperatorCount Tag parameter:

Expression '`<expr>`' containing `<num>` operators exceeds specified limit

Severity

Warning

ConcatenationInArrayAssign-ML

Use assignment patterns in unpacked array assignments

When to Use

Use this tag to check if concatenation is used in an unpacked array assignment.

Tag Description

The *ConcatenationInArrayAssign-ML* tag reports violations, if the LHS of an assignment is an unpacked array variable and the RHS is a concatenation.

Parameter(s)

None

Constraints

None

Message Details

[warning] Concatenation used in unpacked array assignment
'<assignment-statement>' [Hierarchy: '<hierarchy>']

Potential Issues

The violation message is reported when the LHS of an assignment is an unpacked array variable and the RHS is a concatenation.

Consequences of Not Fixing

Not fixing the violation may result in wrong logic generated by the synthesis tools.

How to Debug and Fix

To debug the violation, click the violation message and SpyGlass highlights the assignment in the RTL line.

Examples

Example 1

Consider the following example:

MoreLint Tags

```
module test ();
wire a_array [2:0];
wire b_array [2:0];
reg a;
reg b;
assign a_array = {a,a,a}; // violation
assign b_array = '{b,b,b}; // no-violation
endmodule
```

Example 2

Consider the following example:

```
module test ();
wire a_array [7:0];
wire b_array [7:0];
reg a;
reg b;
assign a_array [2:0] = {a,a,a}; // violation
assign b_array [2:0] = '{b,b,b}; // no-violation
endmodule
```

Example 3

Consider the following example:

```
module test ();
wire a_array [7:0];
wire b_array [7:0];
reg a;
reg b;
assign a_array [1] = {a}; // no-violation for bit select
assign b_array [1] = '{b}; // no-violation
endmodule
```

Example 4

Consider the following example:

```
typedef struct
{
  logic a_array [7:0];
} my_struct;

module test ();
  my_struct st [1:0];
  reg a;
  reg b;
  assign st[0].a_array[2:0] = {a,a,a}; // violation
  assign st[1].a_array[2:0] = '{b,b,b}; // no-violation
endmodule
```

Example 5

Consider the following example:

```
module test (input clk, output reg out);
  reg a_array [2:0];
  reg b_array [2:0];
  reg a;
  reg b;
  always@ (posedge clk)
  begin
    a_array = {a,a,a}; // violation
    b_array = '{b,b,b}; // no-violation
  end
endmodule
```

Example 6

Consider the following example:

```
module test ();
wire [2:0] a_array;
reg a;
assign a_array = {a,a,a}; // no-violation, packed array
endmodule
```

Example 7

Consider the following example:

```
module test();
wire [3:0] a_array [1:0][2:0];
reg a;
assign a_array[0][0] = {a,a,a,a}; // no-violation, assigning to packed part
assign a_array[1] = {a,a,a}; // violation, assigning to
unpacked part
endmodule
```

Example 8

Consider the following example:

```
function sum; // active function
input p, q;
begin
logic a_array [2:0];
logic b_array [2:0];
reg a;
reg b;

a_array = {a,a,a}; // violation
b_array = '{b,b,b}; // no-violation
```

```
sum = p + q;
```

```
end
```

```
endfunction
```

```
module test (input p, q, output out);
```

```
assign out = sum (p, q); // function call
```

```
endmodule
```

Default Severity Label

Warning

Tag Group

morelint

Reports and Related Files

No related reports or files.

CondSigAsgnDelay-ML

Delays should not be used in conditional signal assignment

Language

VHDL

Description

The CondSigAsgnDelay-ML Tag flags delays used in conditional signal assignment statements.

While delays are allowed in RTL description, they are ignored during synthesis. Hence, you should ensure that your design does not depend on delays to achieve correct functionality since this behavior will not necessarily map into post-synthesis behavior.

Message Details

The following message appears at the location where a delay is specified in a conditional signal assignment statement:

```
Conditional signal assignment uses delay
```

Severity

Warning

Examples

Some of the examples where the CondSigAsgnDelay-ML Tag flags a message are as follows:

```
library ieee;  
use ieee.std_logic_1164.all  
  
sig1 <= '0' after 2ns when sel = 1  
      else '1' when sel = 2;  
  
wsig1<= '0' when sel = 1  
      else unaffected when sel =2  
      else 'X' after 1ns;
```

ConflictVar-ML

Flag where conflicting variable names across different processes are used

Language

VHDL

Description

The ConflictVar-ML flags same variables used in different process constructs in the same design unit.

Using the same names reduces readability.

Message Details

The following message appears at the location where variable `<var-name>` is used when the variable was already used in a different process construct in the same design unit at line `<line-num>`:

variable '`<var-name>`' already declared at line `<line-num>`

Severity

Warning

CoveragePragma-ML

The coverage off pragma should have corresponding the coverage on pragma

Language

Verilog

Description

The CoveragePragma-ML Tag flags unmatched coverage off pragma specifications.

The CoveragePragma-ML Tag flags the following cases:

- No matching coverage on pragma specification found
- No coverage on pragma specification found between two coverage off pragma specifications

Message Details

The following message appears at the location of coverage off pragma when no matching coverage on pragma is found in the file or between this coverage off pragma and the next coverage off pragma at line *<num>*:

No matching "coverage on" pragma found in the same file or prior to the subsequent "coverage off" pragma used at line *<num>*

Severity

Info

DetectParamTruncate-ML

Parameter value is truncated

When to Use

Use this Tag to detect truncation of parameter values due to LRM width calculation and sign bit of the value.

Description

The *DetectParamTruncate-ML* Tag reports violations for value assignment for parameters where the intended value is truncated due to LRM width constraints or sign operators.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Parameter '<param_name>' value is truncated from '<initial_value>' to '<truncated_value>' in assignment [hierarchy: '<hier_path>']

Potential Issues

The violation message is reported when a parameter value is truncated due to LRM width calculation and sign bit of the value.

Consequences of Not Fixing

Not fixing the violation may result in unexpected Tag behavior.

How to Debug and Fix

To debug the violation message, click the violation message to highlight line of the parameter assignment in the HDL viewer.

Example Code and/or Schematic

Example 1

```
module test();
```

MoreLint Tags

```

parameter A = 4'sd5;
parameter B = 5'sd8;

parameter C = A * B; // violation, because intended value
is 40, but truncated to 8 due to width is restricted to 5
in A*B

parameter D = $signed (A*B); // violation, because
intended value is 40, but truncated to 8 due to width is
restricted to 5 in A*B

endmodule

```

Example 2

```

module test();
parameter A = 4'sd5;
parameter B = 5'sd10;

parameter C = A * B; // violation, because intended value
is 50, but truncated to -14 due to width is restricted to
5 in A*B and A*B is a signed.

parameter D = $signed (A*B); // violation, because
intended value is 50, but truncated to -14 due to width is
restricted to 5 in A*B and A*B is a signed.

endmodule

```

Example 3

```

module test();
parameter A = 4'd5;
parameter B = 5'd8;

parameter C = A * B; // violation, because intended value
is 40, but truncated to 8 due to width is restricted to 5
in A*B

parameter D = $signed (A*B); // violation, because
intended value is 40, but truncated to 8 due to width is

```

```
restricted to 5 in A*B
endmodule
```

Example 4

```
module test();
parameter A = 4'd5;
parameter B = 5'd10;
parameter C = A * B; // violation, because intended value
is 50, but truncated to 18 due to width is restricted to 5
in A*B
parameter D = $signed (A*B); // violation, because
intended value is 50, but truncated to -14 due to width is
restricted to 5 in A*B and sign operator.
endmodule
```

Example 5

```
module test();
parameter A = 4'sd2;
parameter B = 5'sd5;
parameter C = A * B ; // no violation, because intended
value is 10, and with 5 bits 10 can be represented
parameter D = $signed (A*B) // no violation, because intended value is 10
and with 5 bits 10 can be represented
endmodule
```

Example 6

```
module test();
parameter A = 4'd2;
parameter B = 5'd5;
```

MoreLint Tags

```

parameter C = A * B ; // no violation, because intended value is 10, and
                        // with 5 bits 10 can be represented and A*B is unsigned
parameter D = $signed (A*B) // no violation, because intended value is 10
                        // and with 5 bits 10 can be represented and A*B is unsigned
endmodule

```

Example 7

```

module test();
parameter A = 4'sd2;
parameter B = 5'sd10;

parameter C = A * B ; // violation, because intended value
                        // is 20, and truncated to -8 since A*B is signed
parameter D = $signed (A*B) // violation, because intended
                        // value is 20 and truncated to -8 since A*B is signed
endmodule

```

Example 8

```

module test();
parameter A = 4'd2;
parameter B = 5'd10;

parameter C = A * B ; // no violation, because intended value is 20, since
                        // A*B is unsigned 5 bits can hold 20.
parameter D = $signed (A*B) // violation, because intended
                        // value is 20 and truncated to -8 since A*B is signed due to
                        // $signed operator
endmodule

```

Example 9

```

module test();
parameter A = 4'sd5;

```

```
parameter B = 5'd8;  
parameter C = A * B; // violation, because intended value  
is 40, but truncated to 8 due to width is restricted to 5  
in A*B  
parameter D = $signed (A*B); // violation, because  
intended value is 40, but truncated to 8 due to width is  
restricted to 5 in A*B  
endmodule
```

Example 10

```
module test();  
parameter A = 5'sd18; // violation, because intended value  
is 18, but truncated to -14 due to sign based number.  
endmodule
```

Example 11

```
module test();  
parameter A = 5'sd18; // violation, because intended value  
is 18, but truncated to -14 due to sign based number.  
parameter B = 4'sd5;  
parameter C = A * B // violation, because intended value  
cannot be correctly determined as parameter A is  
truncated, but assigned value truncated to -6 due to sign  
of the expression. Correct A value first, then the  
intended value can be determined correctly.
```

Default Severity Label

Warning

Tag Group

None

MoreLint Tags

Reports and Related Files

None

DefineTypeDefStructure-ML

Always define a typedef when using a structure or union

Description

Any structure or union should always be defined as a user-defined type. Therefore, any struct or union variables declared without typedef are reported by the *DefineTypeDefStructure-ML* Tag. For example:

```
struct { logic a; logic b;} myVar;
```

Instead, structs and unions should be declared as a user-defined type typedef, as follows:

```
struct {logic a; logic b;} myVar_t;  
myVar_t myVar;
```

Language

Verilog

Messages and Suggested Fix

The following message appears when struct or union variables are declared without typedef :

[2-MEDIUM] Un-typedefed <struct/union name> detected in module < module_name> [Hierarchy: <hierarchy name>].

Example Code and/or Schematic

Example 1

Consider the following example:

```
module top ();  
    struct {  
        byte a;  
        reg b;
```



```

        } struct_inst = '{10,0}; //Violation because typedef
is not used
endmodule

```

In the above example, the Tag reports a violation because typedef is not used.

Example 2

Consider the following example:

```
typedef struct { //No violation here

```

```

    byte a;
    reg b;
} struct0;
module top ();
    struct0 struct0_inst= '{10,0};
endmodule

```

In the above example, the Tag does not report a violation because typedef is used.

Example 3

Consider the following example:

```

module top ();
    union packed {
        reg a;
        reg b;
    } union_inst; //Violation because union is not typedef
endmodule

```

In the above example, the Tag reports a violation because union used is not typedef.

Example 4

Consider the following example:

```
module top ();  
    struct packed {  
        byte a;  
        reg b;  
    } struct_inst;  
    assign struct_inst.a = 10; //Violation struct is not  
typedef  
endmodule
```

In the above example, the Tag reports a violation because struct used is not typedef.

Default Severity Label

2-Medium

Reports and Related Files

None

DetectBlackBoxes-ML

Black box instance detected.

Description

The DetectBlackBoxes-ML tag flags black box instances of a module in a design unit.

The DetectBlackBoxes-ML tag generates a violation message for each black box instance found in the design unit.

Message Details

The following message appears at the location where an instance <inst-name> of a module <mod-name> is detected as black box:

Instance '<inst-name>' of module '<mod-name>' detected as black box

Severity

Warning

Suggested Fix

Avoid using black box instance inside a design unit.

DiffDelayInNonBlock-ML

Checks if all the non-blocking assignments have delays and those delays are equal

Description

A delay must be present in all the non-blocking assignments and the delay values should be the same.

The *DiffDelayInNonBlock-ML* Tag reports violation if any of the non-blocking assignments do not have a delay. Also, even if the delays are present but they are not equal, the Tag still reports a violation and indicates that the delays are not same.

NOTE: *Dynamic values assigned as delays are not considered.*

Language

Verilog

Parameters

[ignore_async_blocks](#): By default, the value of this parameter is set to no and the tag reports a violation for asynchronous blocks within the whole always block. Set the value of this parameter to yes to not report a violation for asynchronous blocks.

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears when multiple delay values are found in the sequential statements:

[WARNING] Multiple delay values detected in sequential statements

Message 2

The following message appears when delay clause is not found in the non-blocking assignment:

[WARNING] Missing delay clause in the non blocking assignment

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

Add a delay in those assignments where delay is missing. Make the delay value same as used in the previous assignment if multiple delay value is reported.

Example Code and/or Schematic**Example 1**

In the following example, a violation is reported as no delay is present:

```
always @(posedge CLK or negedge Rst)
    if (!Rst) Q <= 1'b0;
    else Q <= #3 D;
```

Example 2

In the following example the *DiffDelayInNonBlock-ML* Tag reports a violation because the delay values are not the same, as it should be 3 in this case.

```
always @(posedge CLK or negedge Rst)
    if (!Rst) Q <= #3 1'b0;
    else Q <= #4 D;
```

Example 3

In the following example the *DiffDelayInNonBlock-ML* Tag reports a violation because the delay values are not the same, as it should be 3 in this case:

```
module top1();
    always @(posedge CLK or negedge Rst)
        if (!Rst) Q1 <= #3 1'b0;
        else Q1 <= #3 D;
endmodule
module top2();
    always @(posedge CLK or negedge Rst)
```

```
        if (!Rst) Q2 <= #4 1'b0;
        else Q2 <= #4 D;
    endmodule
```

Example 4

In the following example the *DiffDelayInNonBlock-ML* Tag does not report any violation because it is not a non-blocking statement:

```
always @(posedge CLK or negedge Rst)
    if (!Rst) Q = 1'b0;
    else Q = #3 D;
```

Example 5

In the following example the *DiffDelayInNonBlock-ML* Tag does not report any violation because it is not a sequential block:

```
always @(D or Rst)
    if (!Rst) Q = 1'b0;
    else Q = #3 D;
```

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

DisallowForceOnEquiNet-ML

Reports equivalent hierarchical signals due to which multiple drivers happen on same signal

When to Use

Use this tag to detect equivalent hierarchical signals that are driven by multiple assignments.

Description

This tag detects scenarios when two or more hierarchical signals that are driven by multiple assignments. These two or more hierarchical signals are separated by just buffers and connection operators.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a signal <SignalName1> is found to be driven by other statements as well on equivalent signal <SignalName2>

[WARNING] Signal <SignalName1> has driver via Force/Assign statement <ForceDriver1>. Same/Equivalent signal <SignalName2> has driver via another Force/Assign statement <ForceDriver2>

Potential Issues

Violation may arise when a signal has multiple simultaneous drivers.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic shows the multiple driven net and a corresponding load for that net. This signifies an error in the logic.

To fix the violation, ensure that variable/signal does not have multiple drivers, unless logic is intentionally used.

Example Code and/or Schematic

Consider the following example:

```
module top(in1, in2, out1, out2);
    input  in1, in2;
    output out1,out2;
    assign u1.in = in1;
    initial force u1.u2.in = in2;
    initial force out2 = u1.out;
    sub1 u1 (.in(),
    .out(out1));
endmodule
module sub1(in,out);
    input in;
    output out;
    sub2 u2(.in(in), .out(out));
endmodule
module sub2(in,out);
    input in;
    output out;
    wire out=in;
endmodule
```

In the above example, the DisallowForceOnEquiNet-ML reports a violation as the signal, u1.in and u1.u2.in, are equivalent hierarchical signals.

Severity

Warning

DisallowForceOnPortConn-ML

Reports simulation synthesis mismatch due to force assignment.

When to Use

Use this tag to detect equivalent hierarchical signals that are on same signal.

Description

This tag detects scenarios where there is broken connection due to force statement as force statement will override the original connection.

However, the tag does not report violation for force statement when driver on force statement is the same as the original driver.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a signal <Signal> is found to be driven by more than one statements:

```
[ERROR]signal <Signal> has conflicting drivers with force statement driver <ForceDriver> and regular connection driver <RegularDriver>
```

Potential Issues

Violation may arise when a signal has multiple simultaneous drivers.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic shows the multiple driven net and a corresponding load for that net. This signifies an error in the logic.

To fix the violation, ensure that variable/signal does not have multiple drivers, unless tristate logic is intentionally used.

Example Code and/or Schematic

Consider the following example:

```
module test(in1, in2, out1, out2);
  input in1, in2;
  output out1,out2;
  initial force u1.in = in2;
  initial begin force out2 = u1.out;
  end sub1
  u1 (.in(in1),
    .out(out1));
  endmodule
module sub1(in,out);
  input in;
  output out;
  wire out=in;
endmodule
```

In the above example, the DisallowForceOnPortConn-ML reports a violation as the signal, u1.in, has multiple drivers, in1 and in2.

Severity

Error

DisallowSVAwaysLatch-ML

Reports the usage of SystemVerilog `always_latch`

Description

The *DisallowSVAwaysLatch-ML* Tag reports violation when `always_latch` is used in RTL.

The Tag reports violation when `always_latch` is used in the following scenarios:

- Inside modules
- Inside used interfaces
- Inside generate blocks (only used scenarios in the design are considered)

NOTE: *The Tag does not report violations inside unused interfaces.*

Language

Verilog

Messages and Suggested Fix

The following message appears if `always_latch` is used in RTL:

[ERROR] `always_latch` usage detected in module '<module-name>'
[Hierarchy: '<hierarchy>']

Example Code and/or Schematic

Consider the following example:

```
module test (input r, d, output logic q);
    always_latch    // violation
    if (r)
        q <= 0;
endmodule
```

For the above example, the Tag reports a violation because `always_latch` is used in RTL.

Default Severity Label

Error

Reports and Related Files

None

DiffTimescaleUsed-ML

Design should not have different timescales for different modules

Language

Verilog

Description

The DiffTimescaleUsed-ML Tag flags different timescales used in the design.

If different timescales are used in different modules of the design, then in a single simulation, delay values in one module will be different from that of other module.

By default, the DiffTimeScaleUsed-ML Tag reports one violation per timescale, except the first timescale. That is, if a timescale is used multiple times in the design, the *DiffTimescaleUsed-ML* Tag reports a violation for only the first instance of the timescale. Set the value of the [report_all_diff_timescale](#) parameter to `yes` to report violations for all the timescales, except the first timescale.

NOTE: *The first timescale read in the design is considered as the base timescale. Therefore, no violation is reported for the first timescale.*

Message Details

The following message appears at the location where a module has a timescale value `<timescale2>` when a different timescale value `<timescale1>` has already been found in file `<file-name>`, line `<num>`:

Different timescale `<timescale2>` is used. Previous timescale used : `<timescale1>` [FileName = `<file-name>` , Line number=`<num>`]

Severity

Warning

Examples

Example 1

The DiffTimescaleUsed-ML Tag flags the following example where different timescales values have been specified for modules top and lower:

```
`timescale 1 ns / 100 ps
module top(a, b, c);
    input a, b;
    output c;

    lower inst (.a(a), .b(b), .c(c));
endmodule

`timescale 1 ns / 10 ps
module lower (a, b, c);
    input a, b;
    output c;
endmodule
```

Example 2

Consider the following example:

```
module a;
endmodule

`timescale 1s/1ms
module b;
endmodule

`timescale 1s/1ms
module c;
endmodule

`timescale 100ns/1ns // Violation by default and when
report_all_diff_timescale is set
module d;
endmodule
```

MoreLint Tags

```
`timescale 100ms/1ms // Violation by default and when
report_all_diff_timescale is set
module e;
endmodule

`timescale 100ns/1ns // Violation ONLY when
report_all_diff_timescale is set
module f;
endmodule

`timescale 100ms/1ms // Violation ONLY when
report_all_diff_timescale is set
module g;
endmodule

`timescale 1ms/1ns // Violation by default and when
report_all_diff_timescale is set
module h;
endmodule
```

In the above example, by default, the DiffTimescaleUsed-ML Tag reports violation for only third, fourth, and last timescale. If you set the value of the [report_all_diff_timescale](#) parameter to yes, the DiffTimescaleUsed-ML Tag reports violation for third, fourth, fifth, sixth and seventh timescales.

DirectiveCheck-ML

Dumps information about ifdef, undef, and include directives used in design

When to Use

Use this Tag to view information about ifdef, undef, and include directives used in the design.

Description

The DirectiveCheck-ML Tag generates information about ifdef, undef, and include directives used in the design.

Default Weight

10

Language

Verilog

Parameter(s)

- ***dump_ifdef_data***: The default value is `no`. Set the value of the parameter to `yes` to report information about ifdef, undef, and include directives.
- ***check_ifdef_define_include***: The default value is `no`. Set this Tag parameter to `yes`, to enable the Tag to report ``define`, ``ifdef`, and ``include` constructs used in the design.
- ***directivecheck***: By default, the `directivecheck` parameter is set to "" and does not report violations for the following compiler directives.
 - ☐ ``ifdef`
 - ☐ ``undef`
 - ☐ ``else`
 - ☐ ``include`
 - ☐ ``ifndef`
 - ☐ ``elsif`
 - ☐ ``define`

Set the `directivecheck` parameter to a comma-separated list of directives to report violations for the above mentioned compiler directives.

NOTE: *If the `directivecheck` parameter is enabled, then it is prioritized (over other parameters) and other parameters are neglected.*

- [*ignore_directive_names*](#): By default, the parameter is set to `""`. Set the parameter to a comma or space-separated list of PERL regular expressions to provide the names of the compiler directives that should be ignored while checking the tag.

NOTE: *This parameter works only if the [*directivecheck*](#) parameter is enabled.*

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears at the location where the macro `<macro-name>` used in `ifdef` is defined in filename `<file-name>`, line number `<num>`:

[WARNING] The macro '`<macro-name>`' used in '`ifdef`' is defined in Filename '`<file-name>`' LineNumber '`<num>`'

Potential Issues

A violation is reported if an `ifdef` directive is used in the design.

Consequences of Not Fixing

This is an informational Tag and aids in tool transitioning. Therefore, there is no direct implication of not fixing this violation.

You can use this Tag to highlight the compiler directives used in the design as different directives could behave different across tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the instance where the `ifdef` directive is used in the design.

To fix the violation, cross-check the usages of the `ifdef` compiler directive in the design and fix in case of any unintentional usage.

Message 2

The following message appears at the location where the macro `<macro-name>` used in `ifdef` is not defined in the current scope:

[WARNING] The macro '`<macro-name>`' used in '`ifdef`' is not defined in current scope

Potential Issues

A violation is reported when an undefined macro is used in the `ifdef` directive.

Consequences of Not Fixing

This is an informational Tag and aids in tool transitioning. Therefore, there is no direct implication of not fixing this violation.

You can use this Tag to highlight the compiler directives used in the design as different directives could behave different across tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the instance where the `ifdef` directive is used in the design.

To fix the violation, cross-check the usages of the `ifdef` compiler directive in the design and fix in case of any unintentional usage.

Message 3

The following message appears at the location where file `<file-name>` is included in the file:

[WARNING] File '`<file-name>`' is included in the file

Potential Issues

A violation is reported when an `include` directive is used in the design.

Consequences of Not Fixing

This is an informational Tag and aids in tool transitioning. Therefore, there is no direct implication of not fixing this violation.

You can use this Tag to highlight the compiler directives used in the design as different directives could behave different across tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the

instance where the `include` directive is used in the design.

To fix the violation, cross-check the usages of the `include` compiler directive in the design and fix in case of any unintentional usage.

Message 4

The following message appears at the location where macro `<macro-name>` undefined in the file:

[WARNING] Macro '`<macro-name>`' undefined in the file

Potential Issues

A violation is reported when an `undef` directive is used in the design.

Consequences of Not Fixing

This is an informational Tag and aids in tool transitioning. Therefore, there is no direct implication of not fixing this violation.

You can use this Tag to highlight the compiler directives used in the design as different directives could behave different across tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the instance where the `undef` directive is used in the design.

To fix the violation, cross-check the usages of the `undef` compiler directive in the design and fix in case of any unintentional usage.

Message 5

[WARNING] Directive '``<directive_type>
<directive_name_if_available>`' is used

Potential Issues

A violation is reported when directives that have been specified through the `directivecheck` parameter are used in the design.

Consequences of Not Fixing

This is an informational tag and aids in tool transitioning. Therefore, there is no direct implication of not fixing this violation.

You can use this tag to highlight the compiler directives used in the design as different directives could behave different across tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the instance where the directives (that have been specified through the directivecheck parameter) are used in the design.

To fix the violation, cross-check the usages of the specified directives in the design and fix in case of any unintentional usage.

Example Code and/or Schematic

Consider the following example:

```
`define MACRO 4
module mod1(in1,clk,out1);
  input in1,clk;
  output out1;
  `define PARAM = 5;
  parameter PAR = 10;
  assign out1 = in1;
endmodule
```

```
module mod2(in1,clk,out1);
  input in1,clk;
  output out1;
  reg r;
  parameter PAR = 10;
  integer i;
  assign out1 = in1;
  always @(posedge clk)
  begin
    for(i=0;i<3;i=i+1)
      r = in1;
  end
endmodule
```

```
module mod3(in1,clk,out1);
  input in1,clk;
  output out1;
  `define PARAM = 2;
  parameter PAR = 10;
  assign out1 = in1;
```

MoreLint Tags

```
endmodule
```

In the above example, the DirectiveCheck-ML Tag reports a violation as the macro, PARAM is defined more than once.

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

DisallowCaseX-ML

Do not use casex constructs in the design

When to Use

Use this Tag to identify casex constructs used in the design.

Description

The DisallowCaseX-ML Tag reports violation for casex constructs used in the design.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the first line of a casex construct:

[**WARNING**] 'casex' statement used [Hierarchy: '<hier-path>']

Potential Issues

Violation may arise when a casex construct is used in the design.

Consequences of Not Fixing

Use of casex statements may increase chances of overlapping conditions and may deteriorate circuit quality. Such constructs may also lead to mismatches between pre-and post-synthesis simulation. <R&D>

How to Debug and Fix

Select the message that you want to debug.

In the HDL window, it takes you to the line where offending casex construct

has been defined.

To fix the violation, review the code and ensure that the casex construct is not used in the design.

Example Code and/or Schematic

Consider the following example:

```
module top(in, out);
    input  [7:0] in;
    output out;

    mod1 mod1(in, out);

endmodule

module mod1 (in, out);

    input [7:0] in;
    output out;

    reg out;

    always begin

        casex(in)
            8'b00001xx0 : out=1;
            8'b1xxx00xx : out=0;
            8'b00000000 : out=1;
        endcase
    end
endmodule
```

In the above example, the DisallowCaseX-ML Tag reports a violation as a casex statement is used in an always block in a lower level module.

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

DisallowCaseZ-ML

Do not use casez constructs in the design

When to Use

Use this Tag to identify casez constructs used in the design.

Description

The DisallowCaseZ-ML Tag flags casez constructs used in the design.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the first line of a casez construct:

[WARNING] 'casez' statement used [Hierarchy: '<hier-path>']

Potential Issues

Violation may arise when a casez construct is used in the design.

Consequences of Not Fixing

Use of casez statements may increase chances of overlapping conditions and may deteriorate circuit quality. Such constructs may also lead to mismatches between pre- and post-synthesis simulation.

How to Debug and Fix

Select the message that you want to debug.

In the HDL window, it takes you to the line where offending casez construct has been defined

To fix the violation, review the code and ensure that the casez construct is not used in the design.

Example Code and/or Schematic

Consider the following example:

```
module top(in, out);
    input  [7:0] in;
    output out;

    mod1 mod1(in, out);

endmodule

module mod1 (in, out);

    input [7:0] in;
    output out;

    reg out;

    always begin

        casez (in)
            8'b00001xx0 : out=1;
            8'b1xxx00xx : out=0;
            8'b00000000 : out=1;
        endcase
    end
endmodule
```

In the above example, the DisallowCaseZ-ML Tag reports a violation as a casez statement is used in an always block in a lower level module.

Default Severity Label

Warning

MoreLint Tags

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

DisallowDWComp-ML

Do not instantiate DesignWare component used for multiplication, division

Language

Verilog, VHDL

Description

The DisallowDWComp-ML Tag flags instantiation of user-specified DesignWare components in the design.

By default, the DisallowDWComp-ML Tag flags instantiation of the DesignWare components in the design:

DW*mult*	DW*mac*	DW*prod*
DW*div*	DW*sqrt*	DW*_square*

Use the [dw_name](#) Tag parameter to specify your own DesignWare component naming convention.

Using such instances may cause problems in logic equivalence checking, due to the complexity of checking through multipliers and dividers.

Message Details

The following message appears at the location where an instance *<inst-name>* of an unsupported DesignWare Component *<comp-name>* is encountered:

Instantiation '*<inst-name>*' of DesignWare component '*<comp-name>*' may cause problem in logic equivalence checking

Severity

Warning

DisallowMult-ML

Multiplication operator (*) should not be used in the design

Language

Verilog, VHDL

Description

The DisallowMult-ML Tag flags the use of multiplication operator (*) in the design.

Multiplication operation may cause problems in logic equivalence checking, due to the complexity of checking through multipliers.

The DisallowMult-ML Tag ignores the following cases in Verilog and VHDL, where:

Verilog

- One of the operands is a loop variable
- One of the operands is a constant power of 2
- Both the operands are constants
- One of the operands is 0 or 1

VHDL

- Both operands of multiplication operator are constants.
- Multipliers in function and procedure body as size of the arguments may depend on the actuals passed in the function or procedure call.
- One of the operands is a constant power of 2
- One of the operands is 0 or 1

Message Details

The following message appears at the location where a multiplication operator is encountered in expression `<expr>` in hierarchy `<hier-path>`:

Multiplication operation in expression '`<expr>`' may cause problems in logic equivalence checking [Hierarchy: '`<hier-path>`']

Severity

Warning

Examples

Verilog

Example1

```
parameter MAXBITS = 8;
for (k=1;k<2;k=k+1)
out[(k * MAXBITS-1) -: 8] = in;
```

In the above example, the Tag does not reports a violation as $k * \text{MAXBITS}$ is not acting as a multiplier

Example2

```
task my_task;
input in1,in2;
output out1;
begin
    out1=in1 * in2;
end
endtask
```

Example3

```
always @(in1 or out1)
begin
    prod = out1 * in1;
    prod = out1 * 8;
    prod = 2 * 8;
```

In the above example, the Tag does not flags a violation for $prod = out1 * 8$ as 8 is 2 to the power 3.

Also, the Tag does not flags a violation for $prod = 2 * 8$ as $2 * 8$ is a constant and no multiplier is inferred for this.

VHDL

Example1

```
library ieee;
use ieee.std_logic_1164.all
```

```
entity ent is
port( a : in integer;b: in integer;c:out integer);
end ent;
...
```

```
architecture arc of ent is
```

```
begin
```

```
c <= b * a;
```

```
...
```

Example2

```
entity e is
```

```
port(a: in integer range 0 to 65535;c:out integer);
```

```
end e;
```

```
...
```

```
architecture a of e is
```

```
begin
```

```
c <= a * 65536;
```

```
c <= 2 * 4 ;
```

```
...
```

In the above example, the Tag does not flag a violation for $c \leq a * 65536$ as 65536 is 2 to the power 16.

Also, the Tag does not flag a violation for $c = 2 * 4$ as $2 * 4$ is a constant and no multiplier is inferred for this.

DisallowTimeArr-ML

Array of time is not allowed

Language

Verilog

Description

The DisallowTimeArr-ML Tag flags arrays of time variables.

Arrays of time variables are used in simulation and hence, are allowed in RTL description. However, they are ignored during synthesis. Hence, you should ensure that your design does not depend on arrays of time variables to achieve correct functionality since this behavior will not necessarily map into post-synthesis behavior.

Message Details

The following message appears at the location where an array of time variables *<name>* is declared:

Array of time '*<name>*' used

Severity

Warning

DisallowVal-ML

Flag where the values 'U', '-', 'W', 'H' and 'L' are used

Language

VHDL

Description

The DisallowVal-ML Tag flags use of U, -, W, H, and L values.

Message Details

The following message appears at the location where value *<value>* is declared:

value '<value>' used in design

Severity

Warning

DisallowXInCaseZ-ML

Do not use "x" in casez statements

When to Use

Use this Tag to identify x used in casez constructs

Description

The DisallowXInCaseZ-ML Tag reports violation for x used in casez constructs.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where x is used in a casez construct:

[WARNING] 'x' used in casez statement [Hierarchy: '<hier-path>']

Potential Issues

Violations may arise when x is used in a casez construct.

Consequence of Not Fixing

In a casez construct, if the case selector or case branch has an x, there would be mismatch between simulation and synthesis.

How to Debug and Fix

Select the message that you want to debug.

MoreLint Tags

In the HDL window, it takes you to the line where 'X' has been used within the casez construct.

To fix the violation, review the code and ensure that X is not used in casez constructs.

Example Code and/or Schematic

Consider the following example flagged by the DisallowXInCaseZ-ML Tag where X is used in the casez construct:

```
module mod1 (in, out);
  input [1:0] in;
  output out;
  reg out;
  always@( in )
  begin
    casez ( in )
      2'b0x : out=1;
      2'bx0 : out=0;
      2'b01 : out=1;
      default: out=0;
    endcase
  end
endmodule
```

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

DuplicateCase-ML

The conditions of a case/unique-case/priority-case construct should not be duplicated.

Language

Verilog

Description

The DuplicateCase flags a warning when a condition of case/unique-case/priority-case construct overlaps another condition present elsewhere in the same construct. Duplicate cases are redundant - the first matching case will always be executed in simulation and only the first such case will be implemented in synthesis. This indicates a probable error since subsequent matching cases will never be executed or implemented.

NOTE: *The Tag supports the multi-line functionality.*

Message Details

The following message appears when a case condition <condition1> at line <line1> overlaps another case condition <condition2> at line <line2>:

Case Condition '<condition1>' at line number '<line1>' overlaps condition '<condition2>' at line number '<line2>'

Severity

Warning

DuplicateCaseLabel-ML

Ensure that a case construct does not have duplicate case clauses with the `parallel_case` pragma specified

When to Use

Use this Tag to identify case constructs that have duplicate case clauses with the `parallel_case` pragma specified.

Description

The DuplicateCaseLabel-ML Tag reports violation for case constructs that have duplicate case clauses with the `parallel_case` pragma specified.

The DuplicateCaseLabel-ML Tag flags a case constructs when the `parallel_case` pragma is specified and more than one case clauses overlap completely or partially (due to the don't care bits). If the case selector is one-hot signal then the DuplicateCaseLabel-ML Tag flags a violation only if there is an overlap in case branches for any of the possible value of the case selector.

Default Weight

5

Language

Verilog

Parameter(s)

- [*attach_violation_tag*](#): The default value is no. Set the value of the parameter to yes to attach signature (tag) to the violation message to make violation message unique.
- [*ignore_dup_label_stmt*](#): The default value is no. Set the value of the parameter to yes to ignore the duplicate case labels having the same statement block.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location of a case construct clause `<case-label2>` when a overlapping case clause `<case-label1>` was already encountered and the `parallel_case` pragma specified for the case construct:

```
[WARNING] Case '<case-label1>' covered more than once at  
'<case-label2>'
```

Potential Issues

Violations may arise when a case construct does not have a duplicate case clauses with the `parallel_case` pragma specified.

Consequences of Not Fixing

Duplicate cases are redundant; the first matching case will always be executed in simulation and only the first such case will be implemented in synthesis. This indicates a probable error since subsequent matching cases will never be executed or implemented.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the second instance of the duplicate case label used in a case statement with a `parallel_case` pragma. To search for the first usage of the case label, scroll up the HDL Viewer window or search backwards for the second expression given in the violation message.

To fix the violation, carefully examine offending case items and re-code to avoid overlap.

Example Code and/or Schematic

Example 1

Consider the following example:

```
module cases(a, b, c, o1);  
  input [3:0]c;  
  input [3:0]a;  
  input [1:0]b;  
  
  output [3:0]o1;  
  reg [3:0]o1;
```

MoreLint Tags

```

always@(c or b)
begin
    case(b)    // synopsys parallel_case
        2'b00 : o1 = a || c;
        2'b01 : o1 = a & c;
        2'b00 : o1 = a | c;
        2'b11 : o1 = a ^ c;
        default : o1 = 0;
    endcase
end

endmodule

```

In the above example, the DuplicateCaseLabel-ML Tag reports a violation as a parallel case pragma is specified and the case, '2'b00', is covered more than once in the case statement.

Example 2

Consider the following example:

```

always @ (*)
case(p)//synopsys parallel_case
    3'b101,
    3'b101:
        begin
            assign out = 2;
        end

    3'b111:
        begin
            assign out = 1;
        end

    3'b111:
        begin
            assign out = 1;
        end
endcase

```

In the above example, if you set the value of the [*ignore_dup_label_stmt*](#) parameter to `yes`, the *DuplicateCaseLabel-ML* Tag ignores the duplicate case labels having the same statement block and, therefore, reports violation for only 3'b111. However, if the value of the `ignore_dup_label_stmt` parameter is set to `no`, the Tag reports violation for both 3'b101 and 3'b111.

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

DuplicatePort-ML

Do not define duplicated ports

Tag description

The *DuplicatePort-ML* tag flags when a port is defined more than once.

Defining a port more than once is a strong indication of some typo. Either the two distinct ports have been named similarly, due to a typo, or the same port has been specified twice, by mistake. For example, the *DuplicatePort-ML* tag reports violation in the following cases:

Case 1

```
module testve_vlg040 (  
    idle,  
    idle  
);  
input idle;  
endmodule
```

Case 2

```
module testve_vlg040_case2 (  
    idle[0],  
    idle[0],  
    idle[1]  
);  
input [1:0]idle;  
endmodule
```

You should correct the typo because:

- Some tools might not proceed without this being fixed
- If two distinct ports have been named similarly due to a typo, the design intent will not be captured correctly.
- If the same port has been specified twice by mistake, there could be an error if somebody tries to instantiate the module by specifying connections by position.

Language

Verilog

Message Details

The following message appears if a port `<port-name>` has been defined more than once:

Port '`<port-name>`' is defined more than once

Tag Severity

Error

EnableXPropagation-ML

Use "?:" construct instead of "if/else" construct to enable X-propagation

When to Use

Use this Tag to detect the `if-else` or `case` statements in design that does not enable X-propagation.

Description

The *EnableXPropagation-ML* Tag reports those `if-else` or `case` statements in design that does not enable X-propagation. The `"?:"` construct should be used instead of the `"if/else"` construct to enable X-propagation.

For `case` statement and `if-else` chain, this Tag reports violations if `x` is not assigned in the `default/else` branch to all the output variables that are assigned in any of the `if-else-if` branch/case label.

Default Weight

5

Language

Verilog

Parameter(s)

- [*report_nonstatic_assign*](#): Default value is `yes` and the Tag reports violation if non static value is assigned to the target signal in the `else` part or `default` clause. When this parameter is set to `no`, the Tag does not report violation for such scenarios.
- [*report_sequential_always*](#): Default value is `no` and the Tag does not report violations inside sequential `always` blocks. When this parameter is set to `yes`, the Tag to also report violations inside sequential `always` blocks.

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message is reported when the if-else statements in the design do not enable X-propagation:

[WARNING] Replace if-else construct with ?: construct to enable X-propagation

Message 2

The following message is reported when the case statements in the design do not enable X-propagation:

[WARNING] Default clause should have X assignment

Example Code and/or Schematic

Example 1

Consider the following example:

```
if(en) q <= d1;
else q <= d2; // violation
```

The code in the above example should be written as:

```
q = en?d1:d2;
```

Example 2

Consider the following example:

```
if(en1) q <= d1; //violation
else if(en2) q<= d2;
else q <= d3;
```

The code in the above example should be written as:

```
q = en1?d1:(en2?d2:d3);
```

Example 3

Consider the following example:

```
case (sel[7:0]) // violation
    8'h00: c = 1'b1;
    8'h01: d = 1'b1;
    default: begin
```

MoreLint Tags

```
        c = 1'bx;  
        d = 1'b0;  
    end  
endcase
```

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

EntityCompMismatch-ML

Inconsistent case between entity/module name vs component name

Language

VHDL

Description

The EntityCompMismatch-ML Tag flags entity/module names that do not have the same case as the component name.

The EntityCompMismatch-ML Tag reports case-inconsistency between entity/module name and the component name.

The EntityCompMismatch-ML Tag also checks Verilog module name and its corresponding component in VHDL, for case sensitivity.

Message Details

The following message appears at the location where entity name *<ent-name>* and component-name *<comp-name>* have case-inconsistency:

Entity name '*<ent-name>*' and component name '*<comp-name>*' have case-inconsistency

Severity

Info

EnumBaseComparison-ML

Ensures that two variables of different enumerated data types are not compared

When to Use

When enum variable is used in a Verilog or System Verilog design.

Description

This tag reports a violation if the variables of two different enumerated data types are being compared. Assignment of a variable (from enum data type) to another variable (from another enum data type) is already a syntax error.

Default Weight

10

Language

Verilog, System Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is reported if variable `<var1-name>` is being compared with variable `<var2-name>` that are of two different data types:

[ERROR] Enum variable `<var1-name>` (base type: `<type-name1>`) is compared with another enum variable `<var2-name>` (base type: `<type-name2>`) in module `<hier_module_name>`

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

Double click the violation message. The HDL viewer window highlights the line where violation is detected. Fix the violation by changing the RTL code such that only the same type of enum variables are compared.

Example Code and/or Schematic

Example 1

Consider the following example:

```
package definesPkg;
    typedef enum    {FALSE, TRUE} bool;
    typedef enum    {CROSS, TICK} bool_t1;
    typedef enum    {NO, YES} bool_t2;
endpackage

import definesPkg::*;

module simple_package();
    wire clk;
    bool_t1 bool1 ;
    assign bool1 = definesPkg::CROSS;

    bool_t2 bool2 ;
    assign bool2 = definesPkg::NO;

    //bool_t1 bool3 = definesPkg::NO;
    //Syntax Error for above as 'NO' belongs to bool_t2

    bool_t1 bool3 = bool_t1'(definesPkg::NO);
    //No Syntax Error or no Tag violation for above as type
    casting is done

    always @ (clk)
        if(bool1 == bool2)    // Violation through Tag
EnumBaseComparison-ML
```



```

        ;
endmodule

```

In the above example, the *EnumBaseComparison-ML* Tag reports a violation for the comparison of variables of two different enumerated data types `bool1` and `bool2`.

Example 2

Consider the following example:

```

typedef enum logic {
    A0,
    A1
} Enum_type_1;

typedef enum logic {
    B0,
    B1
} Enum_type_2;

module test ();
    Enum_type_1 a ;
    wire test_equals, test_wildcard_equals, test_inside;
    assign test_equals = (a == B1); // Violation
    assign test_wildcard_equals = (a ==? B1); // Violation
    assign test_inside = (a inside {B1}); // Violation
endmodule

```

Default Severity Label

Error

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

EnumDeclInGenerate-ML

Report all enumeration declaration inside generate (for, case, if), nested generate

When to Use

Use this tag to check if an enumeration declaration is specified inside a generate block.

Description

The *EnumDeclInGenerate-ML* tag reports a violation if there is any enum declaration specified inside a generate block. The tag checks the following types of generate blocks:

- generate for
- generate case
- generate if

The tag does not report a violation for enum declarations inside inactive generate-if and generate-case branches by default. Set the `lint_enable_inactive_rtl_check` project file option to `yes` to enable the *EnumDeclInGenerate-ML* tag to report a violation for enum declarations inside inactive generate-if and generate-case branches.

NOTE: *The `lint_enable_inactive_rtl_check` option is an existing switch which enables inactive rtl checks of all other tags as well.*

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[warning] <file-name> <line-number> <rule-weight> Enum declaration <enum-declaration> inside generate block

[Hierarchy: <top-name: block-name>]

Potential Issues

The violation message is reported if any enum declaration is specified inside a generate block.

Consequences of Not Fixing

Not fixing the violation may result in issues in the downstream flow.

How to Debug and Fix

Click the violation message to point to the RTL code where enum is declared and fix it accordingly.

Examples

Consider the following examples:

Example 1

```
parameter USE_CASE = 0;
generate
typedef enum {S_ERROR, S_IDLE, S_EMPTY, S_BUFFER, S_FULL}
    types1;    // violation

if (USE_CASE) begin
typedef enum {S_ERROR, S_IDLE, S_EMPTY, S_BUFFER, S_FULL}
    types2;    //violation only with
    lint_enable_inactive_rtl_check =yes
else begin
typedef enum {S_ERROR, S_IDLE, S_EMPTY, S_BUFFER, S_FULL}
    types3;    //violation
end
endgenerate
```

Example 2

```
parameter ADDER_TYPE = 1;
generate
typedef enum {S_ERROR, S_IDLE, S_EMPTY, S_BUFFER, S_FULL}
    types1;    // violation
```

MoreLint Tags

```

        case (ADDER_TYPE)
0 : typedef enum {S_ERROR, S_IDLE, S_EMPTY, S_BUFFER,
    S_FULL} types2;    //violation only with
    lint_enable_inactive_rtl_check =yes
1 : typedef enum {S_ERROR, S_IDLE, S_EMPTY, S_BUFFER,
    S_FULL} types3;    // violation
endcase
endgenerate

```

Example 3

```

generate
typedef enum {S_ERROR, S_IDLE, S_EMPTY, S_BUFFER, S_FULL}
    types1;    // violation

for (i = 0; i < N; i = i + 1) begin
typedef enum {S_ERROR, S_IDLE, S_EMPTY, S_BUFFER,
    S_FULL}types2;    // violation
end
endgenerate

```

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

EnumOutOfRangeCast-ML

Reports if a cast occurred into an enumerated type with a value which does not lie inside its range.

When to Use

Use this tag to check if a cast occurred into an enumerated type with a value, which does not lie inside its range.

Description

Reports if a cast occurred into an enumerated type with a value, which does not lie inside its range.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] out of range numeric value given by '<numeric value>' cast with enumerated type '<enum type name>' [Hierarchy: <hierarchy>]

Potential Issues

The violation message is reported for a cast in an enumerated type with a value, which does not lie inside its range.

Consequences of Not Fixing

The out-of-range numeric values should not be assigned to enumerated type variables, because such description can lead to indeterminate behavior. As a result, differences between pre-synthesis simulation of the RTL model and synthesis results can occur.

How to Debug and Fix

Click the violation message to highlight the following:

spyglass will point to multiple RTL lines as below.

RTL code for enumeration declaration

RTL code where the out of range numeric value cast is occurred with enumerated type.

Example Code and/or Schematic

Example 1

```
module test(  
    input logic rst, clk  
);  
parameter [2:0] param1 = 3'b101;  
  
reg [2:0] sig1;  
typedef enum logic [2:0]  
  
    {ERR = 3'b001, WRN = 3'b010, INFO }  
msg_states_primary_t;  
msg_states_primary_t stateVar;  
always @(posedge clk, negedge rst)  
begin  
    if (rst == 0)  
begin  
        sig1 = 3'b101;  
        stateVar = msg_states_primary_t'(0); // violation  
        stateVar = msg_states_primary_t'(sig1);  
    end  
    else  
begin  
        stateVar = msg_states_primary_t'(param1); //violation
```

```
stateVar = msg_states_primary_t'(3'o5); //violation
stateVar = msg_states_primary_t'(3'bxxx); //violation
stateVar = msg_states_primary_t'(3'b010);
stateVar = msg_states_primary_t'(3'o2);
end
end
endmodule
```

Example 2

```
module test(
input logic rst, clk
);

//INFO = 0, WRN = 1, ERR = 2, NA = 3
typedef enum{INFO, WRN, ERR, NA} msg_state_set1_t;
msg_state_set1_t msg_state_set1;

always @(posedge clk, negedge rst)
begin
if (rst == 0)
    msg_state_set1 = msg_state_set1_t'(4); //violation
    msg_state_set1 = msg_state_set1_t'(2); //No violation
end
endmodule
```

Example 3

```
module test(
input logic rst, clk
```


MoreLint Tags

```

);

//RMM1 = 2, RMM2 = 7, RMM3 = 8, RMM4 = 9
typedef enum {RMM1=2, RMM2=7, RMM3, RMM4} msg_state_set2_t;
msg_state_set2_t msg_state_set2;

always @(posedge clk, negedge rst)
begin
if (rst == 0)
    msg_state_set2 = msg_state_set2_t'(3); //violation
    msg_state_set2 = msg_state_set2_t'(7); //No violation
end
endmodule

```

Example 4

```

module test(
input logic rst, clk
);
//BLACKBOX0 = 0, BLACKBOX1 = 1
typedef enum {BLACKBOX[2]} msg_state_set3_t;
msg_state_set3_t msg_state_set3;

always @(posedge clk, negedge rst)
begin
if (rst == 0)
    msg_state_set3 = msg_state_set3_t'(2); //violation

```

```
    msg_state_set3 = msg_state_set3_t'(1); //No violation
end
endmodule
```

Example 5

```
module test(
input logic rst, clk
);

//ERR0 = 5, ERR1 = 6, ERR2 = 7
typedef enum {ERR[3]=5} msg_state_set4_t;
msg_state_set4_t msg_state_set4;

always @(posedge clk, negedge rst)
begin
if (rst == 0)
    msg_state_set4 = msg_state_set4_t'(8); //violation

    msg_state_set4 = msg_state_set4_t'(5); //No violation
end
endmodule
```

Example 6

```
module test(
input logic rst, clk
);

//WRN3 = 0, WRN4 = 1, WRN5 = 2
typedef enum {WRN[3:5]} msg_state_set5_t;
```

MoreLint Tags

```
msg_state_set5_t msg_state_set5;

always @(posedge clk, negedge rst)
begin
  if (rst == 0)
    msg_state_set5 = msg_state_set5_t'(4); //violation
    msg_state_set5 = msg_state_set5_t'(2); //No violation
  end
endmodule
```

Reports and Related Files

None

Default Severity Label

Warning

EnumStateDecl-ML

The Finite-state machine's (FSM) enum states parameter declaration should appear before the reg declaration

Language

Verilog

Description

The EnumStateDecl-ML Tag flags enum state parameter declarations that appear after the corresponding reg declaration.

The EnumStateDecl-ML Tag checks that the FSM enum state parameter and reg declarations (identified by the `/*synopsys enum fsm*/ pragma` directive) are in correct order.

Message Details

The following message appears at the location of an FSM's enum state parameter declaration that appears after the corresponding reg declaration:

The param decl (line No. <num>) should appear before the reg decl

Severity

Warning

Examples

The EnumStateDecl-ML Tag flags the following example where parameter declaration `p1` appears after the corresponding reg declaration state:

```
module fsm_example(clk, reset, c, valid);
    input clk, reset, c;
    output valid;

    reg [2:0] state;
    parameter p1 [3:0] ;
    ...
endmodule
```

EventControlInRHS-ML

RHS of assignment statement has a event control

Language

Verilog

Description

The EventControlInRHS-ML Tag flags assignments where the RHS has an event control.

Event control should be defined in the sensitivity list of the always construct.

By default, the tag does not report violation for the blocking assignments. Set the value of the [report_blocking_only](#) parameter to yes to report violation only for the blocking assignments.

Message Details

The following message appears at the location of an assignment statement where the RHS has an event control:

```
The RHS of assignment statement has event control
```

Severity

Warning

ExplicitSignExtend-ML

Reports explicit sign extension

When to Use

Use this tag to detect explicit sign extension.

Description

Verilog can handle automatic sign extension of signed expressions and zero extend for unsigned expressions. Manually extending sign is not recommended as sometimes designer may incorrectly extended the MSB of an unsigned expression or extend sign of signed expression which is not needed.

The ExplicitSignExtend-ML tag reports explicit sign extension.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Explicit sign extension in expression 'expression'.
[Hierarchy: <hierarchy>]

Potential Issues

The violation message is reported when an explicit sign extension is used in an expression.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

Click the violation message to highlight the line where the violation occurred (file and line No. reported in the violation) in the HDL description.

To fix the violation, do not manually extending sign.

Example Code and/or Schematic

Example 1

```
module top();  
  reg [3:0] in;  
  reg [5:0] out;  
  assign out = { {2{in[3]}} , in }; // sign of in is extended  
  improperly.  
endmodule
```

Example 2

```
module top();  
  reg signed [3:0] in;  
  reg signed [5:0] out;  
  assign out = { {2{in[3]}} , in }; // sign of in is extended  
  improperly.  
endmodule
```

Example 3

```
module top();  
  reg [3:0] in;  
  reg [5:0] out;  
  assign out = { {2{in[2]}} , in }; // sign of in is not extended.  
endmodule
```

Example 4

```
module top();  
  reg signed [3:0] in;  
  reg signed [5:0] out;  
  assign out = { {2{in[2]}} , in }; // sign of in is not extended.
```

```
endmodule
```

Example 5

```
module top();  
  reg [3:0] in;  
  reg [5:0] out;  
  reg in2;  
  assign out = { {2{in2}}, in }; // sign of in is not extended  
endmodule
```

Example 6

```
module top();  
  reg [3:0] in;  
  reg [5:0] out;  
  if( out == { {2{in[3]}}, in }) // sign of in is extended  
    improperly  
  ....// some RTL  
endmodule
```

Example 7

```
module top();  
  reg signed [3:0] in;  
  reg signed [5:0] out;  
  if( out == { {2{in[3]}}, in }) // sign of in is extended  
    improperly.  
  ... // some RTL  
endmodule
```

Example 8

```
module top();
```



```

reg signed [3:0] in;
reg signed [5:0] out;
if( out == { {2{in[2]}}, in }) // sign of in is not extended.
... // some RTL
endmodule

```

Example 9

```

module top();
reg [3:0] in;
reg [5:0] out;
if( out == { {2{in[2]}}, in }) // sign of in is not extended.
... // some RTL
endmodule

```

Example 10

```

module top();
reg [3:0] in;
reg [4:0] out;
assign out = { in[3], in }; // sign of in is extended
improperly.
endmodule

```

Example 11

```

module top();
reg [3:0] in;
reg [4:0] out;
assign out = { in[2], in }; // sign of in is not extended.
endmodule

```

Example 12

```

module top();
reg [3:0] in;
reg [7:0] out;
assign out = { in, in }; // sign of in is not extended

```

```
endmodule
```

Example 13

```
module top();  
reg [3:0] in1, in2;  
reg [7:0] out;  
assign out = { in1, in2 };  
endmodule
```

Example 14

```
module top();  
reg [3:0] in;  
reg in2;  
reg [7:0] out;  
assign out = { {4{in2}}, in }; // sign of in is not extended }  
endmodule
```

Example 15

```
module top();  
reg [3:0] in1, in2, in3;  
reg [11:0] out;  
assign out = { in1, in2, in3 };  
endmodule
```

Default Severity Label

Warning

Reports and Related Files

None

ExplicitTwosCompliment-ML

Report explicit two's compliment

When to Use

Use this tag to report explicit two's compliment.

Description

The *ExplicitTwosCompliment-ML* tag reports violations for explicit two's compliment in the form of $\sim xx + C$, instead of $-xx$.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[warning] Explicit two's compliment with terms 'signal' and '<C>' encountered. [Hierarchy: <hierarchy>]

Potential Issues

The violation message is reported when the explicit two's compliment is used.

Consequences of Not Fixing

Not fixing the violation may result in inefficient design implementation.

How to Debug and Fix

Click the violation message to point to the RTL line where the two's compliment is used.

Examples

Consider the following examples:

Example 1

```
module test( a , out);  
input [3:0] a ;  
output [3:0] out ;  
assign out = ~a + 1; // violation  
endmodule
```

Example 2

```
module test( a , out);  
input [3:0] a ;  
output [3:0] out ;  
assign out = -a; // no violation  
endmodule
```

Example 3

```
module test( a ,b out);  
input [3:0] a,b ;  
output [3:0] out ;  
assign out = (~a + 1) + b; // violation  
endmodule
```

Example 4

```
module test( a , out);  
input [3:0] a ;  
output [3:0] out ;  
always @(a) begin  
out = ~a + (1 + b); // violation  
end  
endmodule
```

Example 5

```
module test( a , out);  
input [3:0] a ;
```

MoreLint Tags

```
output [3:0] out ;
assign out = ~a - b + 1; // violation
endmodule
```

Example 6

```
module test( a , out);
input [3:0] a ;
output [3:0] out ;
assign out = (~a + 1) * ( ~b + 1); // 2 violations
endmodule
```

Example 7

```
module test( a , out);
input [3:0] a ;
output [3:0] out ;
always @(a) begin
if(b == (~a + 1)) // violation

end
endmodule
```

Default Severity Label

Warning

FindStringsInComment-ML

Required string present in commented lines

Language

Verilog, VHDL

Description

The FindStringsInComment-ML Tag flags commented lines that contain any of the strings specified through the [comment_string](#) parameter.

Message Details

The following message appears at the location where a commented line contains string `<str>` passed through the `comment_string` parameter:
`comment contains '<str>' string`

Severity

Info

FSMNextStateName-ML

State name prefix or suffix

Description

The *FSMNextStateName-ML* tag reports a violation if the next state of FSM does not have the prefix or suffix (naming convention), which is defined using the [check_next_state](#) parameter. The [check_pattern](#) parameter is used to indicate whether suffix is to be validated or prefix, as a naming convention in the next state of FSM.

Language

Verilog

Parameters

- [check_next_state](#): Default value is "_ns". Set the value of this parameter to a comma separated PERL regular expression patterns to define the prefix or suffix for the next state of FSM.
- [check_pattern](#): Default value is "suffix" and the tag validates the suffix in the next state of FSM. Set the value of this parameter as "prefix" to enable the tag to validate the prefix in the next state of FSM.

Message Details

The following message appears if next state name does not use the specified naming convention:

[WARNING] Next register state <state-name> doesn't match naming convention \$check_next_state

Example and /or Schematic

Example 1

Consider the following code where no parameters are set and the tag uses the default values of the [check_next_state](#) and [check_pattern](#) parameters:

```
always @(state or en)
```

```

begin
curr_state = state;
if (en)
  begin
    case (state)
      ST1 : curr_state = next_state_1; // Violation
      ST2 : curr_state = next_state_2; // Violation
    endcase
  end
end

```

For this example, the tag reports violations because `next_state_1` and `next_state_2` do not have `_ns` as suffix.

Example 2

Consider the following code where the `check_next_state` parameter is set to `ns_` and the `check_pattern` parameter is set as `prefix`:

```

case (state)
ST1 : curr_state = next_state_ns; //Violation
ST2 : curr_state = next_state1_ns; //Violation
endcase

```

For this example, the tag reports violations because `next_state_ns` and `next_state1_ns` do not have `ns_` as prefix.

Example 3

Consider the following code where the `check_next_state` parameter is set to `'_ns'` and `'_nxt'` and the `check_pattern` parameter is set as `suffix`:

```

case (state)
ST1 :
curr_state = next_state_ns; //No violation
ST2 :
curr_state = next_state1_nxt; //No violation
ST3 :
curr_state = next_state1; //Violation

```

MoreLint Tags**endcase**

For this example, the tag reports a violations for `next_state1` because it does not use any of the suffixes specified (`_ns/_nxt`), whereas no violations are reported for `next_state1_ns` and `next_state1_nxt`, because they use the specified suffixes, `_ns` and `_nxt` respectively.

Default Severity Label

Warning

FSMNonConstDefault-ML

Checks whether the next state of FSM is specified as a constant expression in the default branch.

When to Use

Use this Tag detects whether the next state of FSM is specified as a non-constant value.

Description

This Tag reports violations if next state of FSM is specified as a non-constant expression in the default branch of the case block.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Next state "<next state signal>" is specified as a non-constant expression "<non-constant expression>" in the default branch.

Potential Issues

The violation message is reported if the next state of FSM is specified as a non-constant value.

Consequences of Not Fixing

Not fixing the violation may result in unexpected simulation results.

How to Debug and Fix

To debug the violation, click the violation message. This will highlight the RTL line in the HDL viewer where the next state of the FSM is specified as non-constant value.

Example Code and/or Schematic

Example 1

```

module test( Clock, Reset, ThreeOnly, StartFSM1, En_A, En_B,
  En_C );

parameter ST_A=0, ST_B=1, ST_C=2; // FSM1
input Clock, Reset, ThreeOnly, StartFSM1;
output En_A, En_B, En_C;
reg En_A, En_B, En_C;
reg [1:0] CurrStateFSM1, NextStateFSM1;

always @(StartFSM1 or ThreeOnly or CurrStateFSM1)
begin: FSM1_COMB
  En_A = 0;
  En_B = 0;
  En_C = 0;
  case (CurrStateFSM1)
    ST_A : NextStateFSM1 = ST_B;
    ST_B : NextStateFSM1 = ST_C;
    ST_C : NextStateFSM1 = ST_A;
    default : NextStateFSM1 = CurrStateFSM1; // violation
  endcase
end

always @(posedge Clock)
begin: FSM1_SEQ
  if (Reset)
    CurrStateFSM1 = ST_A;
  else
    CurrStateFSM1 = NextStateFSM1;
  end
end

endmodule

```

Example 2

```

module test( Clock, Reset, ThreeOnly, StartFSM1, En_A, En_B,

```

```

En_C);

parameter ST_A=0,ST_B=1,ST_C=2; // FSM1
input Clock, Reset,ThreeOnly,StartFSM1;
output En_A, En_B, En_C;
reg En_A, En_B, En_C;
reg [1:0] CurrStateFSM1, NextStateFSM1;

always @(StartFSM1 or ThreeOnly or CurrStateFSM1)
begin: FSM1_COMB
En_A = 0;
En_B = 0;
En_C = 0;
case (CurrStateFSM1)
ST_A : NextStateFSM1 = ST_B;
ST_B : NextStateFSM1 = ST_C;
ST_C : NextStateFSM1 = ST_A;
default : NextStateFSM1 = ST_A; //no violation
endcase
end

always @(posedge Clock)
begin: FSM1_SEQ
if (Reset)
CurrStateFSM1 = ST_A;
else
CurrStateFSM1 = NextStateFSM1;
end

endmodule

```

Example 3

```

module test( Clock, Reset, ThreeOnly,StartFSM1,En_A, En_B,
En_C);

parameter ST_A=0,ST_B=1,ST_C=2; // FSM1
input Clock, Reset,ThreeOnly,StartFSM1;

```

MoreLint Tags

```
output En_A, En_B, En_C;
reg En_A, En_B, En_C;
reg [1:0] CurrStateFSM1, NextStateFSM1;

always @(StartFSM1 or ThreeOnly or CurrStateFSM1)
begin: FSM1_COMB
  En_A = 0;
  En_B = 0;
  En_C = 0;
  case (CurrStateFSM1)
    ST_A : NextStateFSM1 = ST_B;
    ST_B : NextStateFSM1 = ST_C;
    ST_C : NextStateFSM1 = ST_A;
  endcase
end

always @(posedge Clock)
begin: FSM1_SEQ
  if (Reset)
    CurrStateFSM1 = ST_A;
  else
    CurrStateFSM1 = NextStateFSM1;
  end
endmodule
```

Default Severity Label

Warning

Reports and Related Files

No related reports or files.

FsmUsage-ML

Finite State machine used

When to Use

Use this tag to detect finite state machines (FSMs) in the design.

Description

This tag reports violations for finite state machines (FSMs) in the design.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

Message

```
[warning] finite state machine used for state  
'<currentStateName>' [Hierarchy: '<hierarchy>']
```

NOTE: *Duplicate violations are discarded even if the hierarchies do not match.*

Potential Issues

A violation is reported is when there is a finite state machine in the design.

Consequences of Not Fixing

This is an informative tag about the design. Review each violation.

How to Debug and Fix

Double click the violation message. The HDL viewer window highlights the RTL line where the finite state machine is found.

Example Code and/or Schematic

Example 1

```
reg [1:0] currState, nextState;
always @* begin
    case (currState) // register 'currState' used in a FSM
    [hierarchy: <hierarchy>]
    2'b00 : nextState = 2'b01;
    2'b01 : nextState = 2'b10;
    2'b10 : nextState = 2'b11;
    2'b11 : nextState = 2'b00;
    default : nextState = 2'b00;
    endcase
end
always @(posedge clk or negedge arst) begin
    if (!arst)
        currState <= 2'b00;
    else
        currState <= nextState;
    end
```

Example 2

```
reg [1:0] currState;
always @(posedge clk or negedge arst) begin
    if (!arst)
        currState <= 2'b00;
    else begin
        if (currState == 2'b11) begin currState <= 2'b00; end
        else begin currState <= currState + 2'b01;

```

```
end
```

```
end
```

Example 3

```
reg [1:0] currState, nextState;
```

```
always @* begin
```

```
case (currState) // not a FSM
```

```
2'b00 : nextState <= 2'b01;
```

```
2'b01 : nextState <= 2'b10;
```

```
2'b10 : nextState <= 2'b11;
```

```
2'b11 : nextState <= 2'b00;
```

```
default : nextState <= 2'b00;
```

```
endcase
```

```
currState <= nextState;
```

```
end
```

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

FuncTaskUsedBeforeDefine-ML

Identifies the functions and tasks that have been used before the definition

When to Use

Use this Tag to identify the functions and tasks that have been used before definition.

Description

This Tag identifies the functions and tasks that have been used before definition for cases, for example in a case where there is a requirement where function can not be used before the definition.

Also, the Tag highlights the usage and definition of the function/task in the HDL viewer.

Parameters

None

Messages and Suggested Fix

The following violation message is displayed:

[INFO] [Function|Task] (<function/task name>) defined in file (<function/task definition file>) at line (< function/task definition line number >) is used before declaration

Potential Issues

The violation message is reported when a function or a task is used before the definition.

Consequences of Not Fixing

Not Applicable

How to Debug and Fix

Move the function/task definition before its usage.

Example Code and/or Schematic

Consider the following example:

```
module function_calling(a, f);
    input a ;
    output f;
    wire f;
    assign f = myfunction (a); //Function used before
definition
    function myfunction;
        input a;
        begin
            myfunction = a;
        end
    endfunction
endmodule
```

Default Severity Label

Info

Tag Group

None

Reports and Related Files

None

FuncFileName-ML

FileName should be '<functionname>.v' or '<functionname>.sv' for a file which contains only function definition(s)

Language

Verilog

Description

The FuncFileName-ML Tag flags those source files that contain only function definitions but the source file name is not same as the name of one of the functions described in the source file.

The FuncFileName-ML Tag also checks source files that contain function definitions and descriptions of unused tasks.

For Tag-violating source files, the message reports the name of the first function found in the source file.

Message Details

The following message appears for a Tag-violating source file where the first function is named *<func-name>*:

Use name '<func-name>.<file-extn>' for file containing function definition

Where *<file-extn>* can be *v* or *sv* depending upon the extension of the source file.

Severity

Guideline

FlopFeedbackRace-ML

Reports potential race conditions between flip-flop output and its clock/reset pin

When to Use

Use this tag to report potential race conditions between flip-flop output and its clock/reset pin.

Description

This tag reports potential race conditions between output of a flip-flop and its clock/preset/clear signal. There is a feedback loop, possibly through combinational logic, from a flip-flop output pin to its clock/set/reset pin.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Potential race between flop (output <output-name>) and its [clock|clear|preset] (<signal-name>) detected

Potential Issues

The violation message is reported when there a potential race condition between flip-flop and its clock/preset/clear signal.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

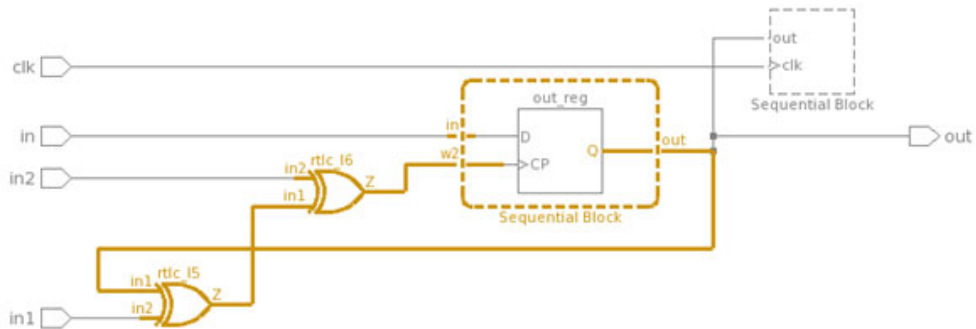
How to Debug and Fix

To fix the violation, ensure that there is no race condition between output of a flip-flop and its clock/clear/present signal.

Example Code and/or Schematic

Consider the following schematic:

MoreLint Tags



For the above schematic, the tag reports the following violation message:
Potential race between flop (output top.out) and its clock (w2) detected

Default Severity Label

Warning

Reports and Related Files

No related reports or files.

GenIndexNonInt-ML

Avoid use of non-integer indexes inside GENERATE constructs

Language

VHDL

Description

The GenIndexNonInt-ML flags use of non-integer indexes of generate constructs as in the following examples:

```
library ieee;
use ieee.std_logic_1164.all

...
type int_enum is ('0','1','2','3');
  for I in int_enum generate
...
type char_range is ('A','B','C','D');
  for I in char_range generate
...
for int i in 0 to 10 generate
...
...
```

NOTE: *The GenIndexNonInt-ML Tag supports generate-for block.*

NOTE: *The GenIndexNonInt-ML Tag is switched off by default.*

Message Details

The following message appears for a non-integer index used in a generate construct:

Avoid non-integer index in GENERATE construct

Severity

Warning

Suggested Fix

Avoid using non-integer indexes if your design methodology does not allow them.

GenvarUsage-ML

Genvar variable has been defined but not used

Language

Verilog

Description

The GenvarUsage-ML Tag flags those genvar variables which are not used inside the generate blocks.

By default, this Tag does not process large arrays. Set the [handle_large_bus](#) parameter to `yes` to process large arrays (greater than 50,000) and report violation if not used correctly.

Message Details

The following message appears at the location where a genvar variable `<var-name>` is declared but not used in the design:

Genvar variable '`<var-name>`' declared but not used. [Hierarchy: '`<hier-path>`']

Severity

Info

Examples

Consider the following example:

```
wire [5:0] a;
wire [5:0] b;
genvar i;
genvar j; // declared but not used
generate
  for (i=0; i<3;i=i+1) begin : gi
    assign a[i] = b[i];
  end
endgenerate
```

In the above example, variable `j` is declared but not used in the design. Hence, the GenvarUsage-ML Tag flags a violation for this variable.

GenvarDeclaration-ML

For-Loop variable defined by using genvar must be declared outside of generate block.

When to Use

Use this Tag to report violations if the for-loop variable is defined by using genvar and declared inside the generate block.

Description

The *GenvarDeclaration-ML* Tag reports violations if for-loop variable is defined using genvar and declared inside the generate block.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Genvar <Signal name> is declared inside a generate block. [Hierarchy: <hierarchy to declaration>]

Potential Issues

The violation message is reported when a genvar signal is declared inside a generate block.

Consequences of Not Fixing

Not fixing the violation may result in unexpected Tag behavior.

How to Debug and Fix

To debug the violation, click the violation message to highlight the line where genvar is declared (file and line No. reported in the violation), in the HDL viewer.

Example Code and/or Schematic

Example 1

```
module test;
```

```

    reg [4:0] data;
    generate
    genvar i; // violation
    for (i=0; i < 4; i=i+1) begin
        assign data[i] = i%2;
    end
    endgenerate
endmodule

```

NOTE: Since the UFE flow does not generate "VE_GENERATE"s, therefore, the Tag will not report violations in the UFE flow. The following illustrates the decompile in default and UFE flows:

Decompile in Default Flow :

```

module test;reg [4:0] data;
    generate genvar i;for(i = 0;(i < 4);i = (i + 1))begin : genblk1assign data[i] = (i % 2);endendgenerateendmodule
    VE_GENERATE
    VE_GENERATE_FOR

```

Decompile in UFE Flow :

```

module test;reg [4:0] data;genvar i;for(i = 0;(i < 4);i = (i + 1))begin : genblk1assign data[i] = (i % 2);endendmodule
    VE_GENERATE_FOR
    * no VE_GENERATE

```

Example 2

```

module tblk (a,z,ck);
    input ck;
    output z;
    input a;
    generate
        genvar i;//Violation
        assign pin[0] = a;
        assign z = pin[3];
        for (i=0; i<3; i=i+1) begin : pblk
            wire conn;
            flop pa (.d(pin[i]), .q(conn), .ck(ck));
            flop pb (.d(conn), .q(pin[i+1]), .ck(ck));
        end
    endgenerate
endmodule

```

MoreLint Tags

```

        end
    endgenerate
endmodule

```

Example 3

```

module top(read,write,data_in,address,data_out);
    input read,write;
    input [31:0] data_in;
    input [3:0] address;
    output reg [31:0] data_out;
    genvar i; //No Violation
        generate
    for (i=0; i < 4; i=i+1) begin : MEM
        memory U (read, write,
            data_in[(i*8)+7:(i*8)],
            address,data_out[(i*8)+7:(i*8)]);
    end
endgenerate
endmodule

```

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

GlobalDataTypeDefined-ML

Data type objects (user-defined type declarations and definitions of identifiers) should not be present or visible in global Scope.

When to Use

Use this Tag to check for the presence of data type objects in global scope.

Description

The *GlobalDataTypeDefined-ML* Tag reports violation when user-defined type declarations (typedef , struct , union , enum) or definitions of identifiers (data type objects, such as, parameter, localparam, function, task, integer, wire, reg, and logic are visible in global scope.

Declaration of any user-defined type or definition of any identifier outside of module/package/interface/program scope is considered as an error, even if they are never used anywhere.

Therefore, this Tag reports violations for following cases.

- User-defined type declarations and definitions of identifiers directly in global scope.
- Including a file, which contains user-defined type declarations and definitions of identifiers in global scope.
- Importing a package, which contains user-defined type declarations and definitions of identifiers in global scope.

Parameters

None

Constraints

None

Messages and Suggested Fix

[ERROR] Data type object '<object-name>' is present in global scope. [File name = '<file name>' Line number = '<line number>']

Potential Issues

The violation message is reported when a data type object is present in the global scope.

Consequences of Not Fixing

Not fixing the violation may result in unexpected SpyGlass behavior.

How to Debug and Fix

Violation message points to the file and the line where the violation occurred.

- If that definition (data type object) is present in one of the input RTL files, the violation message will point out the file and line number where the definition of identifier was found
- If that definition (data type object) is present in one of the include file or imported package, the violation message will point out the file and line number where the include/import command was found.

Example Code and/or Schematic

Example 1

```
Test.v-----

typedef logic Boolean; //Error - Data type object
'typedef' is present in global scope. [File name = 'Test.v'
Line number = '1']

module ( );

endmodule
```

Example 2

```
File.sv-----

typedef logic Boolean;

-----

Test.v -----
```

```
`include "file2.sv" // Error - Data type object 'typedef'
is present in global scope. [File name = 'File.v' Line
number = '1']
module ();
endmodule
-----
```

Example 3

```
File.v -----

package pkg;
typedef logic Boolean;
endpackage
-----

test.v -----

import pkg::Boolean; // Error - Data type object 'typedef'
is present in global scope. [File name = 'File.v' Line
number = '2']

module ();

endmodule
-----
```

Example 4

```
Test.v-----
module ( );
typedef logic Boolean; //No Error
endmodule
-----
```

MoreLint Tags

Example 5

```
File.sv-----  
  
typedef logic Boolean;  
  
-----  
  
Test.v -----  
  
module ();  
`include "file2.sv" // No Error  
endmodule  
  
-----
```

Example 6

```
File.v -----  
package pkg;  
typedef logic Boolean;  
endpackage  
  
-----  
  
test.v -----  
  
module ();  
import pkg::Boolean; // No Error  
endmodule  
  
-----
```

Default Severity Label

Error

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

GroupOFAsgn-ML

Verilog treats differently C-like groups of assignments

Language

Verilog

Description

The GroupOFAsgn-ML Tag flags C-like assignment groups.

While a C-like group of assignments construct is allowed, Verilog interprets the construct differently from C language.

For example, consider the following statement that has C-like group of assignments:

```
b <= c <= d <= 1'b0;
```

Based on C language semantics, you would expect the above statement to be interpreted as three separate assignment statements where three signals (b, c, and d) are assigned the value 1'b0 respectively.

However, Verilog interprets the same statement with the first <= symbol as a nonblocking assignment and other <= symbols as Less Than or Equal To operators:

```
b <= (c <= (d <= 1'b0));
```

Thus, the innermost expression is a Less Than or Equal To comparison between value of signal d and 1'b0. The result of this expression is compared with the value of signal c and the final result is assigned to signal b.

Message Details

The following message appears at the location where a C-like group of assignments is encountered:

Possibly a C-like group of assignment intended here

Severity

Warning

HierarchyReport-ML

Tag will generate a report with hierarchies of usage of global Verilog entities

Description

The *HierarchyReport-ML* tag generates a report with the hierarchies of usage of the following global Verilog entities:

- global interfaces
- global packages
- global functions
- global tasks

The tag reports only one hierarchy for one entity and does not report the hierarchy for the following scenarios:

- if global function/task is not used.
- if interface is not instantiated.
- if no declaration from package is used.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when the tag generates a report with the hierarchies of usage of the global Verilog entities:

A report named `HierarchyReport.rpt` will be generated and all the hierarchy details of used entities will be included.

Report format:

<entity type> name: '<name>', Hierarchy: '<hierarchy>' @

<file>:<line>

Example Code and/or Schematic

Example 1: Interface

Consider the following snippet:

```
interface Bus;  
  logic [7:0] Addr, Data;  
  logic RWn;  
endinterface
```

```
module test;
```

```
  Bus TheBus(); // Instance the interface - hierarchy ':test' will be reported
```

```
  logic[7:0] mem[0:7];
```

```
endmodule
```

Example 2: Package

Consider the following snippet:

```
package my_pkg;  
  parameter P1 = 4;  
endpackage
```

```
import my_pkg::*;
```

```
module test(input in, output out);
```

```
  parameter TMP = P1; // Used package item - hierarchy ':test' will be  
  reported
```

```
  assign out = in;
```

```
endmodule
```

Example 3: Global function/ task

Consider the following snippet:

```
function myfunction;
```

```
input a, b, c, d;
begin
  myfunction = ((a+b) + (c-d));
end
endfunction
```

```
module test(a, b, c, d, e, f);
input a, b, c, d, e ;
output f;
sub sub1(a, b, c, d, e, f);
endmodule
```

```
module sub(a, b, c, d, e, f);
input a, b, c, d, e ;
output f;
wire f;
assign f = (myfunction (a,b,c,d)) ? e :0; // Function call - hierarchy
':test:sub1@sub' will be reported
endmodule
```

Default Severity Label

NA

Tag Group

NA

Reports and Related Files

NA

IfOverlap-ML

The conditions of an if-construct (including unique-if and priority-if) should not overlap (Verilog)
The conditions of an if-construct should not overlap (VHDL)

Language

Verilog, VHDL

Description

Verilog

The IfOverlap-ML Tag flags if constructs where one or more conditions in successive if..else..if statements (including unique-if and priority-if) overlap.

VHDL

The IfOverlap-ML Tag flags if conditions of if ..else..if statement overlap. This Tag checks for both sequential and conditional if blocks and reports violation if the conditions of if..else..if statement evaluates to same conditions.

Message Details

Verilog

The following message appears at the location of if construct condition *<cond1>* (at line *<num1>*) that overlaps condition *<cond2>* (at line *<num2>*) of the same if construct:

If condition '*<cond1>*' at line number '*<num1>*' overlaps condition '*<cond2>*' at line number '*<num2>*'

VHDL

The following message appears at the location of if construct condition *<cond1>* that overlaps condition *<cond2>* of the same if construct at line *<num>*:

If condition '*<cond1>*' overlaps condition '*<cond2>*' at line number '*<num>*'";

Severity

Warning

Examples (Verilog)

In the following example, the *IfOverlap-ML* Tag reports violation because the condition in successive *if..else..if* statements is overlapping:

```
module abc;
wire [1:0] sel;
reg c;
always begin
    if ( sel == 2'h0 )
    begin
        c = 1'b0;
    end
    else if ( sel == 2'h0 ) //Violation
    begin
        c = 1'b1;
    end
    else if ( sel == 2'h2 )
    begin
        c = 1'b0;
    end
    end
endmodule
```

Examples (VHDL)

In the following example, the *IfOverlap-ML* Tag reports violation because the condition in successive *if..else..if* statements is overlapping:

```
library ieee;
use ieee.std_logic_1164.all

...
process(I3,I2,I1,I0,S,P)
begin
    if (S="00") then
        Z <= I0;
    elsif (S="00") then //Violation
        Z <= I1;
    else
```

MoreLint Tags

```
        Z <= I3;  
    end if;  
end process;  
...
```

IfWithoutElse-ML

An "if" clause does not have a corresponding "else" clause and is not preceded by assignment of target signal

Language

Verilog, VHDL

Description

The Tag supports the following data types for target signals:

- logic
- reg
- bit
- byte

For more information, see [Example 2](#).

The IfWithoutElse-ML Tag reports violation for if statements that do not have corresponding else statements and are not preceded by assignment of target signal.

The Tag also reports a violation when an `else` statement corresponding to an `if` statement is empty.

The IfWithoutElse-ML Tag does not flag a variable that is assigned in an `if` construct without an `else` construct if it is already assigned in both `if` and `else` branches of an earlier `if-else` construct.

By default, the *IfWithoutElse-ML* Tag reports the `if` blocks that contain explicit `begin` and `end` statements. Set the [ignore_explicit_if_blocks](#) parameter to `yes` to not report violation for the `if` blocks that contain explicit `begin` and `end` statements.

By default, the IfWithoutElse-ML Tag does not report violation for the signals that are assigned inside both `case` block and `if` condition. Set the value of the [check_conditional_assignment](#) parameter to `yes` to report violation for such signals.

By default, this Tag processes inside-generate blocks. If you set the [ignore_generate_blocks](#) parameter to `yes`, the Tag does not process inside-generate blocks.

By default, the Tag reports latch-inferred `if` blocks. If you set the [ignore_latches](#) parameter to `yes`, the Tag ignores latch-inferred `if` blocks.

By default, the [ignore_loop_index](#) parameter is set to `no`. Set this parameter to `yes` to ignore violations for the `if` blocks where the `if` condition is only dependent on loop variable.

The `else` statement covers conditions that are not met by the preceding `if` (or `elseif` in VHDL) statement. A missing `else` statement may lead to an inferred latch. Thus, to ensure that the resulting circuit is always combinational, the `if` statements must have the corresponding `else` statements or all the target signals must have a default initialization through a blocking assignment statement before the `if` statement.

NOTE: *The IfWithoutElse-ML Tag supports generate-if, generate-for, and generate-case blocks.*

By default, the value of the [ignoreRealLatch](#) parameter is set to `no`. Set the value of the parameter to `yes` to enable the *IfWithoutElse-ML* tag to ignore real latches. SpyGlass considers the following latch (with NBA) as a real latch:

```
always_latch
if (clk) o <= d;
```

By default, the [ignore_top_level_pre_assignment](#) Tag parameter is set to `no` and the Tag report violations for "`if`" blocks without "`else`" block if there are pre-assignments for all the variables modified inside "`if`" block. If you set the [ignore_top_level_pre_assignment](#) parameter to `yes`, the Tag reports violations for "`if`" blocks without "`else`" blocks that are present inside both "`case`" block and "`if`" block and the pre-assignment done outside the parent `if` block.

By default, the value of the [ignore_static_condition](#) parameter is set to `no`. Set the value of the parameter to `yes` to enable the *IfWithoutElse-ML* tag to ignore violations for `if` blocks with static condition.

Message Details

The following message appears at the location of a Tag-violating `if` statement:

'if' statement with condition `<cond-expr>` does not have a corresponding 'else' statement and is not preceded by assignment of target signal [Hierarchy: '`<hier-path>`']

Where,

- `<cond-expr>` is a condition expression
- `<hier-path>` is the instance hierarchy of the Tag violating module

Severity

Warning

Examples (Verilog)

Example 1

The IfWithoutElse-ML Tag flags the following example where the if statement does not have its corresponding else construct and there is no default assignment of the target signal OP:

```
...
always @( SEL or IP )
begin
    if ( SEL )
        OP = IP;
    end
...

```

Example 2

Consider the following example:

```
data_type DE2A;
...
    DE2A          = 1'b0;
    if (e2a & brgr) begin
        DE2A = 1'b1;
    end
end

```

In the above example, when `data_type` is logic, reg, bit, or byte, the Tag will not report violations.

Examples (VHDL)

Example 1

The IfWithoutElse-ML Tag flags the following example where the if statement does not have its corresponding else construct and there is no default assignment of the target signal y:

```
library ieee;
use ieee.std_logic_1164.all

...
process(sel, a, b)
begin
    if ( sel = '0') then
        y <= a;
    end if;
end process;
...
```

Example 2

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

module test (ena,enb,tempA,tempB);
input  ena, enb;
output tempA, tempB;

reg tempA, tempB;

always @(*)
begin
    case(ena)
        00 : tempA = 0 ;
        01 : tempB = 1 ;
    endcase

    if (enb)begin
```

```
        tempA = tempA + 6 ;  
    end  
end  
endmodule
```

In the above example, by default, the *IfWithoutElse-ML* Tag does not report violation for the signal, `tempA`, that is assigned inside both `case` block and `if` condition. Set the value of the [check_conditional_assignment](#) parameter to `yes` to report violation for `tempA`.

ImplicitParamImports-ML

Report re-definition of a localparam implicitly imported from a package

When to Use

Use this tag to check if a parameter that is implicitly imported from a package is also declared locally.

Description

The *ImplicitParamImports-ML* tag reports violations for the definition of a parameter inside a module if the same parameter is implicitly imported from a package within the file scope.

If a parameter is implicitly imported (such as by using wild card), tools use the local definition and not the imported definition of the parameter. This behavior is aligned with LRM.

However, the *ImplicitParamImports-ML* tag reports a violation in this case. Such cases of a local definition by a designer might occur because the designer is unaware of the existence of the parameter in the package. This might lead to an unexpected behavior.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[warning] Declaration '<param_name>' is implicitly imported from package '<package_name>' [Hierarchy: <Hierarchy>]

Potential Issues

The violation message is reported if the same parameter is defined in both the package and local scope.

Consequences of Not Fixing

Not fixing the violation may result in incorrect design modeling.

How to Debug and Fix

Click the violation message to point to the RTL code where the package parameter declaration occurs in the design.

Example Code and/or Schematic**Example 1**

```
test.svp
package test_p;
localparam my_param = 2;
endpackage

Test.v
`include "test.svp"

module test
import test_p::*;
(input wire i, output wire o);

localparam int my_param = 1;//violation
assign o = my_param;
```

Example 2

```
test.svp
package test_p;
localparam my_param = 2;
endpackage
```

MoreLint Tags

```
Test.v
`include "test.svp"
localparam int my_param = 1;//violation
```

```
module test
(input wire i, output wire o);
import test_p::*;
assign o = my_param;
endmodule
```

Example 3

```
test.svp
package test_p;
localparam my_param = 2;
endpackage
```

```
Test.v
`include "test.svp"

task convert;
input [7:0]temp_in;
begin
logic temp_out;
import test_p::*;
localparam int my_param = 1; //violation
temp_out = (9/5) *( temp_in + my_param);
end
endtask
```

```
module test
  ( input [7:0]i, output [7:0]o);
  always @ (i)
  begin
    convert (i);
  end
endmodule
```

Example 4

```
test.svp
package test_p;
localparam my_param = 2;
localparam new_param = 2;
endpackage
```

```
Test.v
`include "test.svp"
```

```
module test
  import test_p::new_param;
  ( input wire i, output wire o);
```

```
localparam int my_param = 1;// No violation
```

```
assign o = my_param;
```

```
endmodule
```


MoreLint Tags

Example 5

```
test.svp
package test_p;
    localparam new_param = 2;
endpackage

Test.v
`include "test.svp"

module test
import test_p::*;
( input wire i, output wire o);

localparam int my_param = 1;// No violation
assign o = my_param;

endmodule
```

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

ImplicitAlways-ML

Reports violations for implicit always blocks (always@*) that will never trigger during simulation

When to Use

Use this Tag to reports violations for implicit always blocks (always@*) that will never trigger during simulation.

Description

This Tag reports implicit always blocks (always@*), that will never trigger during simulation.

The implicit always blocks do not trigger, if no variable signal is read within implicit always block.

Therefore, this Tag report violations, if no variable signal is read within an implicit always block.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Implicit always block will never trigger during simulation. [Hierarchy: '<hierarchy >']

Potential Issues

A violation message is reported for implicit always blocks that will never trigger during simulation.

Consequences of Not Fixing

Not fixing the violation may result in unexpected simulation result.

How to Debug and Fix

To debug the violation, click the violation message. The HDL viewer displays the RTL line where implicit always block is present.

To fix the violation, ensure that a variable signal is read in always block, or

a variable signal is present in sensitivity list of the always block.

Examples and/or Schematic

Example 1

```
module top( out);  
output out;  
reg out;  
  
always @(*) begin //violation  
    out = 1'b0;  
end  
endmodule
```

Example 2

```
module top(in, out);  
input in;  
output out;  
reg out;  
  
always @(in) begin //no violation  
    out = 1'b0;  
end  
endmodule
```

NOTE: *The Tag does not report a violation because always block is not implicit.*

Example 3

```
module top(in, out);  
input in;
```

```
output out;  
reg out;
```

```
always @(*) begin //no violation
```

```
    out = in;  
end  
endmodule
```

NOTE: *The report does not report a violation because a variable signal ('in') is read in implicit always block.*

Example 4

```
module top( out);  
output out;  
reg out;  
parameter param = 1'b0;
```

```
always @(*) begin //violation
```

```
if(param == 1'b0)  
    out = 1'b0;  
else  
    out = 1'b1;  
end  
endmodule
```

NOTE: *The report does not report a violation because no variable signal is read in implicit always block.*

Example 5

```
module top(in, out);  
input in;  
output out;
```

MoreLint Tags

```
reg out;
parameter param = 1'b0;

always @(*) begin //no violation
    if(in == 1'b0)
        out = 1'b0;
    else
        out = 1'b1;
    end
endmodule
```

NOTE: *The report does not report a violation because variable signal ('in') is read in implicit always block.*

Default Severity Label

Warning

Reports and Related Files

None

ImproperRangeIndex-ML

Possible discrepancy in the range index or slice of an array

Description

The *ImproperRangeIndex-ML* Tag reports a violation if the number of bits required to cover index does not match $\log_2 N$. For example, if the size of a signal is ten bits, so a four-bit signal is required to cover its range. Therefore, a violation is reported if:

- any signal is used as bit-select/part-select and is not of four bits
- signal used in a bit-select is less than four bits or greater than four bits.

The tag also reports violation for static index out of range issues.

However, when the indexed part-select is used, the width calculation of the max index is enhanced. For example, consider the following example:

```
A +: B  = A+B-1 : A  -----1
Max index =  A+B-1
width of the max index :- width of (A+B-1)
A -: B  = A : A - B + 1 ----- 2
Max index =A
Width of max index : - width of A
```

The tag evaluates the for loop index max value where the condition and or initial values are non-static.

NOTE:

- 📖 *The Tag checks the index at all possible places in the module*
- 📖 *Tag checking is done for the non-static indexes only*

Language

Verilog

Parameters

- ***concat_size_based_width***: By default, the value of the parameter is `no`. Set the value of the parameter to `yes` to consider the specified width of sized-based numbers in non-static concatenation expressions without considering their value.
- ***check_natural_width_of_div***: Default value is `no`. Set the value of the parameter to `yes` to calculate the natural width of the division operator when it's divisor is static, regardless of the value of the *nocheckoverflow* parameter.
- ***check_natural_width_of_modulo***: Default value is `no`. Set the value of the parameter to `yes` to calculate the natural width of the modulo operator, regardless of the value of the *nocheckoverflow* parameter.
- ***nocheckoverflow***: By default, this parameter is set to `yes` and the width is calculated according to the LRM and natural width is considered for constants. If you set the value of the *nocheckoverflow* parameter to `no` then the width is calculated considering the best fit width of an expression. That is, the width in which maximum value of an expression can be accommodated.
- ***REPORT_LOOP_VAR_EXPR***: Default value is `no`. Set the value of the parameter to `yes`, to reports violations for index expressions with loop variable. When both *ignoreInteger* and *REPORT_LOOP_VAR_EXPR* parameters are set to `yes`, tag will report index expressions with integer loop variables, and doesn't reports index expressions with integer variables, which are not used as loop variables.
- ***report_underflow_expressions***: By default, the value of the parameter is `no`. Set the value of the parameter to `yes` to report violations for underflow scenarios also.
- ***report_only_from_one_hierarchy***: By default, the value of the parameter is `no`, and the tag reports violations for multiple hierarchies. Set the value of the parameter to `yes` to suppress the hierarchy field and report only one violation per assignment.
- ***report_max_val_overflow***: By default, the value of the parameter is `no`. Set the value of the parameter to `yes` to report a violation when the maximum value that can be represented by the index is larger than the maximum value of the range of the signal being indexed.

- **ignoreInteger**: By default, the value of the parameter is `no`. Set the value of the parameter to `yes` to not report a violation for integer index.

Message Details

The following message appears when there is a discrepancy in the range index or slice of an array:

[WARNING] Index '<index-string>' of width '<bit-number>' is <larger/fewer> than the width '<number>' required for the max value '<number>' of the signal '<signal-name>' [Hierarchy : <hier-name>]

Example and /or Schematic

Example 1

Consider the following example where the Tag does not report violation because `b4` covers all the bits of `a4` and width of `b4` is equal to the width of 8:

```
wire [8:0] a4;  
wire [3:0] b4;  
assign c4 = a4[b4]; // No violation
```

In the above example, the Tag does not report violation `[3:0]` is the only possible width to cover the `[8:0]`.

Example 2

Consider the following example:

```
wire [8:0] a4;  
wire [2:0] b4;  
assign c4 = a4[b4]; // Violation
```

In the above example, the Tag reports violation because `[2:0]` is not enough as it has only 0,1,2, and 3.

Example 3

Consider the following example:

MoreLint Tags

```
wire [8:0] a4;
wire [5:0] b4;
assign c4 = a4[b4]; // Violation
```

In the above example, the Tag reports violation because [5:0] has more unused numbers (0,1,2,...31), even though it covers 0...8 indices.

Example 4

Consider the following example where the Tag does not report violation:

```
wire [7:0] a4;
wire [2:0] b4;
assign c4 = a4[b4]; // No violation
```

Example 5

Consider the following example where the Tag does not report violation because the LRM width of b4+b5 is "3" to honor self-determined expressions:

```
wire [7:0] a4;
wire [2:0] b4; wire [2:0] b5;
assign c4 = a4[b4+b5]; // No violation
```

Example 6

Consider the following example where the Tag does not report violation for the first index 'b4', whereas reports a violation for the second index 'b4':

```
wire [7:0] [3:0]a4;
wire [2:0] b4;
assign c4 = a4[b4][b4];
```

Example 7

Consider the following example:

```
assign a [0][1:0]= b [0][c:0];
```

In this case, a syntax gets flashed because non-static c

cannot be present in slice.

Example 8

Consider the following example where the Tag does not report violation for the first index 'b4.' The Tag also does not report violation for the second index 'b4 [b3]' as it is a bit select and 1 bit and parent range of a4 is [0:0], which is also 1 bit. The Tag reports a violation for b3.

```
wire [7:0] [0:0]a4;  
wire [2:0] b4;  
wire [2:0] b3;  
assign c4 = a4[b4][b4[b3]];
```

Example 9

Consider the following example where the Tag does not report violation because b4 covers the a4 bits:

```
wire [50001:50000]a4;  
wire [15:0] b4;  
assign c4 = a4[b4]; // No violation
```

Example 10

Consider the following example where the Tag reports a violation because b4 bits does not cover any of a4 bits:

```
wire [50001:50000]a4;  
wire [1:0] b4;  
assign c4 = a4[b4]; // Violation
```

Example 11

Consider the following example where the Tag reports a violation. Though the width of the range of a4 (the width of -8 is 4) matches the width of b4, but violation is still reported, because b4 doesn't cover all a4 bits:

```
wire [-8: 7]a4;  
wire [3:0] b4;  
assign c4 = a4[b4];
```

Example 12

Consider the following example:

```
wire [-8: 7]a4;  
wire signed [3:0] b4;  
assign c4 = a4[b4]; // No violation
```

Example 13

Consider the following example where the Tag reports a violation because b4 does not cover all the ranges of a4, maximum value of b4 is 7 and it does not cover 8:

```
wire [-1: 8]a4;  
wire signed [3:0] b4;  
assign c4 = a4[b4]; // Violation
```

Example 14

Consider the following example where the Tag reports a violation because b4 does not cover all the ranges of a4, minimum value of b4 is 0 and it does not cover -1:

```
wire [-1: 8]a4;  
wire [3:0] b4;  
assign c4 = a4[b4]; // Violation
```

Example 15

Consider the following example where the Tag does not report violation because the width of 8 is 5 for signed ranges (-16 to +15 has width 5, -8 to +7 has width 4):

```
wire [-1: 8]a4; //signed range as one of the index is -ve  
wire signed [4:0] b4;  
assign c4 = a4[b4]; // No violation
```

Example 16

Consider the following example where the Tag reports a violation because the LRM width of `b4+c4` is 3 only for self determined expressions while width of 8 in `a4`'s range is 4:

```
wire [8: 0]a4;  
wire [2:0] b4;  
wire [2:0] c4;  
assign c4 = a4[b4 + c4]; // Violation
```

Example 17

Consider the following example where the Tag reports a violation because the width of `b4 [2 : 0]` is 3 and it does not match with the width of numeric 8 (width: 4) in the range `a4`:

```
wire [8: 0]a4;  
wire [8:0] b4;  
assign c4 = a4[b4[2:0]]; // Violation
```

Default Severity Label

Warning

InActiveReset-ML

Do not use inactive sets/resets on flip-flops

Language

Verilog+VHDL

Parameter(s)

None

Constraints

■ [set_case_analysis](#)

Message Details

The following message appears when the set/reset pin of flip-flop *<flip-flop-name>* is inactive:

<type> of flop *<flip-flop-name>* is inactive

Where *<type>* can be Set or Reset.

Example

VHDL

```
library ieee;
use ieee.std_logic_1164.all

architecture rtl of testcase_RstCste is
  signal rst_n1, rst_n2 : std_logic;
begin
  rst_n1 <= '1';
  rst_n2 <= rst_n or '1';

  p1 : process(clk, rst_n1)
  begin
```

```
        if rst_n1 = '0' then out1 <= '0';
        elsif clk = '1' and clk'event then
            out1 <= d1; --bad: reg never reseted (see rst_n1)
        end if;
    end process p1;

    p2 : process(clk, rst_n2)
    begin
        if rst_n2 = '0' then out2 <= '0';
        elsif clk = '1' and clk'event then
            out2 <= d2; --bad: reg never reseted (see rst_n2)
        end if;
    end process p2;
end rtl;
```

Tag Severity

Error

IncludeFileForEachModule-ML

Checks if include file is explicitly included in each child module

When to Use

Use this Tag to identify the required include file that is not included in the source file.

Description

This Tag reports a violation when a macro of an include file is used inside a design file but include file is not explicitly included in the design file.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when the include file *<include-file-name>* is not explicitly included in the design file *<design-file-name>* before using the macro *<macro-name>*:

[WARNING] Include file '*<include-file-name>*' is not explicitly included before usage of '*<macro-name>*' in design file '*<design-file-name>*'

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

Double click the violation message. The HDL viewer window highlights the line where macro is used but whose corresponding include file is not included in source file.

Example Code and/or Schematic

Consider the following example:

```
//file test.v
module top(a);
    output a;
    assign a = `macro1;
endmodule
//end of file test.v

//file mac.v
`include "mac2.v"
`define macro1 0
//end of file mac.v
```

In the above example, macro `macro1` is defined in the include file, `mac.v` and used in the source file, `test.v`.

In this case, the Tag reports a violation for `macro1` because `mac.v` is not included in `test.v`.

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

InlineComment-ML

Detect absence of space before/after an inline comment

Language

Verilog, VHDL

Description

The InlineComment-ML Tag flags inline comments without a space before/after the starting comment delimiter.

By default, the InlineComment-ML Tag reports only first 30 messages for each source file. Set the [allviol](#) Tag parameter to *yes* or *<Tag-name>* to report all violation messages.

A comment explains the purpose of the code, increasing understanding for future reuse.

Message Details

The following message appears at the location where an inline comment does not have one space before/after the starting comment delimiter:

Absence of space is detected before/after an Inline Comment

Severity

Guideline

InstNameSameAsMaster-ML

Do not use an instance name identical to involved master name

Language

Verilog+VHDL

Message Details

VHDL

The following message appears for instance that has the same name as its master design unit *<du-name>*:

Do not use an instance name identical to master name '*<du-name>*'

The following message appears for instance that has the same name as its parent entity *<entity-name>*:

Do not use an instance name identical to entity name '*<entity-name>*'

Verilog

The following message appears for an instance that has the same name as its parent module:

Do not use instance name identical to parent module name

The following message appears for an instance that has the same name as its module:

Do not use instance name identical to same module name

Suggested Fix

Modify one or other of the reported names to avoid duplication.

Example

Verilog

```
module testve_nam048(in1, in2, out1, out2);  
    input in1, in2;  
    output out1, out2;
```

MoreLint Tags

```
wire net_and_inst1;  
wire net_and_inst2;  
  
comp_01 testve_nam048(.A(in1), .Z(net_and_inst1)); // bad  
comp_01 comp_01 ( .A(net_and_inst1), .Z(out1)); // bad  
subtest_R28 subtest_R28 (  
    .ins(in2), .outs(net_and_inst2)); // bad  
subtest_R28 test_R28_comp_02(  
    .ins(net_and_inst2), .outs(out2)); // good  
endmodule
```

Tag Severity

Error

InstNameStyle-ML

**Preferably name instances as per
<instance_prefix><component_name><instance_suffix>**

Tag description

The *InstNameStyle-ML* tag flags if an instance name does not follow the recommended naming convention.

For RTL designs, the recommended naming convention for instance names is as follows:

```
<instance_prefix><instantiated_component_name><instance_suffix>
```

Where:

- *<instance_prefix>* is specified through the [instance_prefix](#) parameter
- *<instance_suffix>* is specified through the [instance_suffix](#) parameter.

This guarantees a robust and informative naming convention.

For a netlist design, the [instance_name_netlist](#) parameter is used to specify the naming convention for the instance names.

Language

Verilog+VHDL

Parameter(s)

- [instance_prefix](#): By default, the `instance_prefix` tag parameter is set to the following:

```
"$legal_prefix"
```

Thus, the tag flags those instance names that do not follow this naming convention. You can set the `instance_prefix` tag parameter to any valid Perl regular expression.

- ***instance_name_netlist***: By default, the `instance_name_netlist` tag parameter is set to the following:

```
/ (^[a-zA-Z]\$) | (^[a-zA-Z][a-zA-Z0-9_]*[a-zA-Z0-9]\$) /
```

The default value of this parameter ensures the following:

- ☐ Name should begin with an alphabet.
- ☐ Name should not end with an underscore ('_') and contain only [a-zA-Z0-9_].

You can specify your own naming convention by specifying a different Perl regular expression pattern for instance names in a netlist design.

- ***instance_suffix***: By default, the `instance_suffix` tag parameter is set to the following:

```
"_[0-9]+"
```

Thus, the tag reports the instance names that do not follow this naming convention.

You can set the `instance_suffix` tag parameter to any valid Perl regular expression.

Message Details

The following message appears for instance `<name>` that does not follow the recommended naming convention:

Instance name '`<name>`' should follow the specified convention

Suggested Fix

Name the instances as per the convention.

Example

VHDL

Assuming that the `instance_prefix` tag parameter is set to `u_` and the `instance_number` tag parameter is set to `_[0-9] +`:

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```
entity test_nam108 is
```

```
end;

architecture str of test_nam108 is
  component test_nam108comp is
  end component;

  component test_nam108a is
  end component;
  component test_nam108b is
  end component;

  component TEST_NAM108ab is
  end component;

begin
  u_test : test_nam108comp; --bad: missing component name
  test_nam108comp_0: test_nam108comp; --bad: missing prefix
  utr_test_nam108comp_9z : test_nam108comp; --bad prefix
  u_test_nam108comp : test_nam108comp; --good
  u_test_nam108a : test_nam108a; --good
  u_test_nam108b_1 : test_nam108b; --good
end str;
```

Tag Severity

WarningReuse

InterfaceNameConflicts-ML

Interface Signal Name conflicts with other signal name

Language

Verilog

Description

The InterfaceNameConflicts-ML Tag flags if signals declared in an interface, structure, or union matches with the signals declared in a module.

There can be name conflicts in some tools when an interface name and its internal signals, parameters, etc., are same as signals except for a period (".") and an underscore ("_"). This is because some tools convert period to an underscore. This results in a syntax issue even if the syntax is as per LRM. For example:

```
interface foo_if;
logic sig;
modport source {
output sig;
};

endinterface
module foo (
foo_if somename;
);
logic somename_sig;
endmodule
```

In the above example, somename.sig gets converted to somename_sig and thus conflict with the signal name. This results in shorted nets.

The InterfaceNameConflicts-ML Tag does not dives into structural hierarchy for name conflicts.

Message Details

The following message appears when a signal *<sig-name>* declared in an

interface, structure, or union conflicts with the other signal name *<name>*
in module *<mod-name>*:

```
'<type>' name '<sig-name>' conflict with name '<name>' in  
module '<mod-name>'
```

Where *<type>* can be Interface, Structure, or Union.

Severity

Warning

InterfaceWithoutModport-ML

Interfaces should make use of modport declarations.

Language

SystemVerilog

Description

The InterfaceWithoutModport-ML Tag reports violation for the following cases:

- For interfaces which do not have modport definition, but are used in the design.
- For interfaces connected without modports, even if the interface has modports declared.

If modports are not used with interfaces then signals become bi-directional and can be used as both input and output.

Parameter(s)

- *ignore_sub_modport*: The default value is no. Set the value of the parameter to yes to disable violations reported by the *InterfaceWithoutModport-ML* Tag when the modport is explicitly specified in the sub-module definition.
- *check_top_ports*: The default value is no. Set the value of the parameter to yes to report a violation when the interface typed ports are used without modports in the module definition.

Message Details

Message 1

The following violation message is displayed, if the interface does not have modport declaration, but is used in the design.

Interface <interface_name> declaration does not have a modport definition

Potential Issues

The violation message is reported when the interface does not have modport declaration, but is used in the design.

Consequences of Not Fixing

Not fixing the violation can result in direction mismatches in port connections.

Message 2

The following violation message is displayed, if the interface has a modport declaration, but is used without modport:

Interface connection <interface_instance_name> does not use modport

Potential Issues

The violation message is reported when interface has a modport declaration, but is used without modport.

Consequences of Not Fixing

Not fixing the violation can result in direction mismatches in port connections.

Tag Severity

Warning

Examples**Example 1**

Consider the following example:

```
interface inf (input clk, rst);  
logic [7:0] status1;  
endinterface  
  
module top(input clk, rst);  
wire w1, w2, w3;  
inf bus(.*)//Interface used here  
test tInst(bus, w1, w2, w3);  
endmodule
```

In the above example, the Tag reports a violation at the interface declaration as the interface does not have a modport definition but is used in the design.

Example 2

Consider the following example:

```
module top(input clk, rst, W_R, output addr);  
  wire w1, w2, w3;  
  
  interface inf (input clk, rst, W_R, output addr);  
    logic [7:0] status1;  
    modport P1 (input addr, W_R, output status1);  
    modport P2 (output addr, W_R, input status1);  
  endinterface  
  
  inf bus(.*);  
  test tInst(bus, w1, w2);  
  test tInst2(bus.P1, w1, w2);  
endmodule
```

In the above example, the Tag reports violation for the bus port of tInst as the interface is used without modport. However, the Tag does not report violation for the bus.P1 port of tInst2 as the P1 modport is used for interface connection.

IntRange-ML

It is necessary to specify a range for integer/real signal.

Language

VHDL

Description

The IntRange-ML Tag flags integer and real signals that have been specified without a range.

It is a common convention for synthesizable code that such signals should be specified with range.

The IntRange-ML Tag will not report violations for the following:

- Integer type constants
- Integer type variables (whose class is not constant) when used as:
 - vector index
 - left or right part of range

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

variable INDEX_1 : INTEGER := 1;
    --range not specified, but used as vector index
variable INDEX_2 : INTEGER := 2;
    --range not specified, but used left part of
    range
variable NOT_INDEX : INTEGER := 3;
    --range not specified, but neither used as
    vector index nor as left/right part of range
A(INDEX_1) = '0'; --INDEX_1 used as vector index
signal sig : std_logic_vector(INDEX_2 downto 0);
    --INDEX_2 used as left part of range

for I in 10 downto INDEX_2 loop
```

MoreLint Tags

```
        --INDEX_2 used as right part of range
...
...
end loop;
```

Violation is not reported for `I` in the `for` loop, as it is a `CONSTANT` that has an *implicit* declaration of type `INTEGER`. This is also applicable to index used in `for...generate` schemes.

Message Details

The following message appears at the location where a signal `<sig-name>` of type integer or real has been defined without a range constraint:

Object '`<sig-name>`' has integer type with out range specification

Severity

Guideline

InstPortConnType-ML

Output and inout port connection in instantiation should not be of type, reg.

When to Use

Use this Tag to detect ports that are connected to a reg type signal of the instantiating module.

Description

This Tag reports violations for output port and inout port connections in instantiations, if port is connected to a reg type signal of instantiating module.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Register '<register name>' is illegally connected to output port '<port name>' of module '<module name>', instance '<instance name>'.

Potential Issues

A violation message is reported when a register is connected to an output port of a module instance.

Consequences of Not Fixing

Not fixing the violation may result in unexpected Tag behavior.

How to Debug and Fix

To debug the violation, click the violation message to highlight the violating instance in the HDL viewer.

Example Code and/or Schematic

Example 1

```

module test();
  reg  a;
  reg  b;
  reg  c;

  sub ss( .a(a) ,           //no violation , input port
         .b(b) ,           //violation
         .c(c) );          //violation
  and aa( c , b , a );      //violation

endmodule

module sub( input a, inout b, output c);

  assign c = a & b;

endmodule

```

Example 2

```

module test();
  typedef reg rr;
  wire a, b;

  reg      c1;
  reg [2:0] c2;
  wire     c3;
  logic    c4;
  rr       c5;

  sub ss1( .a(a) , .b(b) , .c(c1) ); //violation
  sub ss2( .a(a) , .b(b) , .c(c2[0]) ); //violation
  sub ss3( .a(a) , .b(b) , .c(c3) ); //no violation
  sub ss4( .a(a) , .b(b) , .c(c4) ); //no violation

```

```
sub ss5( .a(a) , .b(b) , .c(c5) ); //violation

endmodule

module sub( input a, input b, output c);
assign c = a & b;

endmodule
```

Example 3

```
module test();
reg a;
reg b;
reg c1;
wire c2;

sub ss( .a(a), .b(b), .c({c1,c2}) ); //violation

endmodule

module sub( input a, input b, output [1:0]c);

assign c[0] = a & b;
assign c[1] = a & b;

endmodule
```

Example 4

```
interface intf();
reg a;
reg b;
reg c;
modport sport( input a, inout b, output c );

endinterface
```


MoreLint Tags

```

interface myintf( input a , input b , output c);
endinterface

module test();
reg a,b,c;
myintf mi( .a(a) , .b(b) , .c(c) ); //violation

intf ii();
sub ss( .iport(ii) );      //violation
endmodule

module sub( intf.sport iport );
assign iport.c = iport.a & iport.b;
endmodule

```

Example 5

```

module test();
reg [3:0]a;
reg [3:0]b;
reg [3:0]c;

genvar i;
generate
for( i = 0; i < 4 ; i = i + 1) begin :
sub ss( .a(a[i]) ,      //no violation , input port
        .b(b[i]) ,      //violation
        .c(c[i]) );      //violation
end
endgenerate

endmodule

module sub( input a, inout b, output c);

assign c = a & b;
endmodule

```

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

InvalidNetType-ML

Invalid net type detected

Description

The *InvalidNetType-ML* tag reports a violation for all nets except wired.

This tag ensures that Verilog netlist design does not have a net type other than 'wire'.

This tag reports a violation message for non-wire net types, as follows:

- wand, wor
- tri, tri1, tri0, trireg
- triand, trior
- reg

Language

Verilog

Parameters

None

Messages and Suggested Fix

The following message appears when non-wire nets are found:

[ERROR] Invalid Net Type <net-type> declared for Net '<net-name>'

Potential Issues

A violation message is reported when a non-wire net type is declared.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

To debug the violation, click the violation message to highlight the violating (invalid) net-type declared.

Default Severity Label

Error

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

InvalidRepeatCondition-ML

X or Z values are not allowed in repeat condition

Description

The *InvalidRepeatCondition-ML* Tag reports violation if the condition of repeat statement evaluates to X or Z value (for example, 2'b0x, 2'b1z).

Language

Verilog

Message Details

The following message appears when the condition of repeat statement evaluates to X or Z value:

[WARNING] Repeat expression <expression> evaluates to X or Z, causing it to repeat zero or unknown times (Simulation).
[Hierarchy: <hierarchy>]

Example and /or Schematic

Consider the following example:

```
repeat (2'b0z) begin //Violation
...
end
repeat (2'b0x) begin //Violation
...
end
repeat (5) begin //No Violation
...
end
repeat (i+2'b0z) begin //Violation
...
end
```

Default Severity Label

Warning

InvalidTypedefName-ML

Invalid typedef name detected

Description

The *InvalidTypedefName-ML* Tag reports violation for all the typedefs used whose names do not have the suffix '_t'.

Also, if the [typedef_regex](#) is set to a comma separated list of PERL regular expressions specifying the typedef names, the Tag reports all the used typedefs whose names do not match with any of the regular expressions in the specified list.

Language

Verilog

Parameter(s)

- [typedef_regex](#): By default, the parameter is set to '.*_t'. Set the value of this parameter to a comma-separated list of PERL regular expressions specifying accepted typedef names, to enable the *InvalidTypedefName-ML* tag to report all the used typedefs whose names do not match with any of the regular expressions in the specified list.
- [check_unused_typedef](#): By default, the parameter is set to no. Set the value of this parameter to yes to enable the *InvalidTypedefName-ML* tag to report a violation for unused typedef names.

Messages and Suggested Fix

The following message appears if typedefs are used whose name doesn't have the suffix '_t':

[WARNING] Typedef '<typedef-name>' doesn't follow recommended naming conventions

Example Code and/or Schematic

Example 1

Consider the following example:

```
module top(in1, out1);
```

MoreLint Tags

```

    input in1;
    output reg out1;
    typedef integer mytype; // violation
    always_comb begin
        mytype x;
        out1 = in1;
    end
endmodule

```

For the above example, the Tag reports a violation because `mytype` does not have the `'_t'` suffix.

Example 2

Consider the following example where the `typedef_regex` parameter is set to `'.*type, .*_t'`:

```

module top(in1, out1);
    input in1;
    output reg out1;

    typedef integer sometype1;           //Violation
    typedef sometype1 someothertype_t;   //No violation
    typedef someothertype_t mytype;      //No violation
    always_comb begin
        mytype x;
        out1 = in1;
    end
endmodule

```

For the above example, the Tag:

- Reports violation for `sometype1`, because it does not match with any regular expressions in the [typedef_regex](#) parameter.
- Does not report violation for `someothertype_t` because it matches with the `'*_t'` regular expression, specified in the `typedef_regex` parameter.
- Does not report violation for `mytype` because it matches with the `'.*type'` regular expression, specified in the `typedef_regex` parameter.

Default Severity Label

Warning

Reports and Related Files

None

LiteralOverflow-ML

Check if the given bit string literal is of a length longer than specified by the width specification

When to Use

Use this tag to check if the length of the literal value is longer than the specified bit-length.

Description

The *LiteralOverflow-ML* tag reports a violation if the length of the literal value is shorter than the specified bit-length.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[warning] "Length of literal value '%s' is longer than specified bit-length '%d'"

Potential Issues

The violation message is reported when the length of the literal value is longer than the specified bit-length.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tool behavior.

How to Debug and Fix

Click the violation message to point to the RTL line where a particular number based literal is used.

Examples

Consider the following examples:

Example 1

```
module top (in, out);  
  input [7:0]in;  
  output [7:0]out;  
  assign out[7:0] = 8'b0001010000; // violation  
  assign out[7:0] = 8'o1010; // violation  
  assign out[7:0] = 8'h001; // violation  
endmodule
```

Example 2

```
module lit_flo (in, out1, out2, out3);  
  input [7:0]in;  
  output [7:0]out1, out2, out3;  
  assign out1[7:0] = in[7:0] & 8'h2;  
  assign out3[7:0] = 8'b0001_0000;  
  assign out1[7:0] = in[7:0] & 16'0020;  
endmodule
```

Example 3

```
module lit_flo (in, out1, out2, out3);  
  input [7:0]in;  
  output [7:0]out1, out2, out3;  
  //Actual widths are mentioned in the comments below  
  assign out1[7:0] = in[7:0] & 8'd2; // 10  
  assign out3[7:0] = 8'd400; // 1 1001 0000  
  assign out1[7:0] = in[7:0] & 8'd216; // 1101 1000  
  assign out1[7:0] = in[7:0] & 8'd002; // <zeros ?>10  
  //What should be the width to consider when there are preceding  
  zeros?
```

MoreLint Tags

endmodule

Default Severity Label

Warning

InvalidVariedTypedefNameStyle-ML

Reports type names defined by typedefs of struct, union, enum, or other that do not follow a specified or default naming style pattern.

When to Use

Use this tag to detect typedefs that do not follow a specified naming convention.

Description

The tag reports violations for all used typedefs that do not follow the naming convention specified by the default style regex or any of the style regular expressions specified using the parameters listed in the below table:

TABLE 2

Parameter	Typedef Declaration
enum_typedef_regex	regex to match enum typedef names
union_typedef_regex	regex to match union typedef names
struct_typedef_regex	regex to match struct typedef names
other_typedef_regex	regex to match other typedef names

Parameter(s)

- [*enum_typedef_regex*](#): Default value is `".*_e"`. Set the value of the parameter to a comma-separated perl regular expression list of accepted typedef names.
- [*other_typedef_regex*](#): Default value is `".*_t"`. Set the value of the parameter to a comma-separated perl regular expression list of accepted typedef names.
- [*struct_typedef_regex*](#): Default value is `".*_s"`. Set the value of the parameter to a comma-separated perl regular expression list of accepted typedef names.

- [*union_typedef_regex*](#): Default value is `".*_u"`. Set the value of the parameter to a comma-separated perl regular expression list of accepted typedef names.

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Typedef '`<typedef-name>`' of '`<construct-type>`' does not follow recommended naming conventions

Potential Issues

The violation message is reported when the specified typedef does not follow the recommended naming convention.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

Double-click the violation to probe to the HDL description.

On clicking the violation message in console, SpyGlass points to the RTL line of where the particular typedef is declared.

Example Code and/or Schematic

Example 1

Consider the following example where the style patterns are as given by default:

```
module top();
    typedef enum
    { IDLE_STATE, CYPHER_STATE }
    state_e; //no viol

    typedef struct {
        bit [7:0] opcode;
        bit [23:0] addr;
    } instruction_s; //no viol
```

```

typedef union packed {
    bit   r_fmt;
    bit   i_fmt;
    bit   m_fmt;
} content_u; //no viol
typedef logic [31:0] uint32_t; //no viol

typedef enum
{ IDLE_STATE_1, CYPHER_STATE_1 }
state_; //viol

typedef struct {
    bit [7:0] opcode;
    bit [23:0] addr;
} instruction_struct; //viol

typedef union packed {
    bit   r_fmt;
    bit   i_fmt;
    bit   m_fmt;
} content_packed; //viol

typedef logic [31:0] uint32_logic; //viol

endmodule

```

Example 2:

Consider the following example where style patterns are as given by the parameters:

```

configure_lint_tag_parameter -tag
InvalidVariedTypedefNameStyle-ML -parameter
enum_typedef_regex -value { [a-z]([a-z0-9]+)*_enum_suffix,
[a-z]([a-z0-9]+)*_enumobj  }

```

```

configure_lint_tag_parameter -tag
InvalidVariedTypedefNameStyle-ML -parameter
enum_typedef_regex

```

MoreLint Tags

```

configure_lint_tag_parameter -tag
InvalidVariedTypedefNameStyle-ML -parameter
union_typedef_regex -value { [a-z]([a-z0-9]+)*_union }

module top();
    typedef enum
    { IDLE_STATE, CYPHER_STATE }
    state_enum; //viol - the naming pattern not
provided in the list

    typedef struct {
        bit [7:0] opcode;
        bit [23:0] addr;
    } instruction_s; //no viol - default naming pattern still applies

    typedef union packed {
        bit r_fmt;
        bit i_fmt;
        bit m_fmt;
    } content_u; //viol - default naming pattern
overridden

    typedef logic [31:0] uint32_t; //no viol -default naming pattern still
applies

    typedef enum
    { IDLE_STATE_1, CYPHER_STATE_1 }
    state_enumobj; //no viol - naming pattern provided in list

    typedef struct {
        bit [7:0] opcode;
        bit [23:0] addr;
    } instruction_struct; //viol - default naming
pattern still applies

    typedef union packed {
        bit r_fmt;

```

```
        bit i_fmt;  
        bit m_fmt;  
    } content_union; //no viol - naming pattern provided in list  
  
    typedef logic [31:0] uint32_logic; //viol - default  
    naming pattern applied  
  
endmodule
```

Tag Severity

Warning

Reports and Related Files

None

LogicEnumBase-ML

Always declare enum base type explicitly as sized logic type

Language

SystemVerilog

Description

The LogicEnumBase-ML Tag reports violation, if enum is declared without any type, or if the type is something other than sized logic.

Tag will report violation at the line of declaration of the enum.

The LogicEnumBase-ML Tag checks for enum in all the scopes such as within module, within packages, within interfaces, in tasks/functions, inside generate statements and at global scope.

Message Details

Enum <enum_name> declaration with type other than sized logic found

Potential Issues

A violation message is reported

Consequences of Not Fixing

Not fixing the violation may lead to simulation and synthesis mismatch.

Tag Severity

Warning

Examples

Example 1

Consider the following example:

```
typedef enum { FIRST, SECOND, THIRD, FOURTH} st1_type;  
module test();  
    enum bit [0:1] {GREEN, ORANGE, RED} signal;  
endmodule
```

Example 2

Consider the following example:

```
module test();  
    enum reg [0:1] {GREEN, ORANGE, RED} signal;  
endmodule
```

In the above example, the Tag does not report a violation.

LongHierName-ML

Long hierarchical name inferred for the construct

Tag Description

The *LongHierName-ML* tag reports a violation when the length of complete hierarchical name (inferred) of a construct in the netlist or RTL design exceeds the value specified by you through the [hierarchical_name_length](#) parameter.

The types of constructs reported by the tag are as follows:

Construct	Description
net	Wire, reg, logic declarations, and implicitly declared nets
port	Declarations of input, output, and inout ports of a module
instance	Module instantiations
variable	Input, output, inout, reg variable declarations inside task and function, and genvar declarations
parameter	Parameter and local parameter declarations
task	Task declarations and constructs declared inside tasks
function	Function declarations and constructs declared inside functions
real	Real declarations
integer	Integer declarations

By default, the tag reports a violation for all types of constructs mentioned above. To specify the types of constructs to be reported, use the [check_type](#) parameter.

By default, the tag reports a violation for constructs declared in the top-level and higher-level hierarchies. To stop the violations for constructs declared in the top-hierarchy, set the [check_in_top](#) parameter to `no`.

Language

Verilog

Parameter(s)

- ***check_type***: By default, the value of the `check_type` parameter is set to `all` and the *LongHierName-ML* tag reports violations for all types of following constructs:
`net, port, instance, parameter, variable, task, function, integer, real`
Set the value of the `check_type` parameter to a comma-separated or space-separated list of one or more of the above construct types to specify the construct types to be reported by the *LongHierName-ML* tag.
- ***check_in_top***: By default, the value of the `check_in_top` tag parameter is set to `yes` and the *LongHierName-ML* tag reports constructs declared in the top-level and higher-level hierarchies. Set the value of this parameter to `no` to stop the violations for constructs declared in the top-hierarchy. In this case, only the constructs declared at least two levels down the hierarchy are reported by the *LongHierName-ML* tag.
- ***hierarchical_name_length***: By default, the value of the `hierarchical_name_length` tag parameter is set to 1024. You can specify a different positive integer value to specify a different permissible length.

Message Details

The following message appears when the length of the *<hier-name>* hierarchical name (inferred) of the *<construct-name>* construct in the netlist or RTL design exceeds the value specified in the `hierarchical_name_length` parameter:

Long hierarchical name '*<hier-name>*' inferred for '*<construct-name>*'

Examples

Example 1

Consider the following example when the `hierarchical_name_length` parameter is set to 6:

```
module top(s, out1);
```

MoreLint Tags

```

input s; //No violation for port 's', Full hierarchical
        name: 'top.s', length = 5
output [1:0] out1; //Violation for port 'out1', Full
                  hierarchical name: 'top.out1', length = 8
reg [1:0] a1; //No violation for net 'a1', Full hierarchical
             name: 'top.a1', length = 6

always@(posedge s)
begin: L1
    reg [1:0] a2 ; //Violation for net 'a2', Full hierarchical
                  name: 'top.L1.a2', length = 9
end

endmodule

```

The following violations are reported for the above example:

Long hierarchical name 'top.out1' inferred for 'port'

Long hierarchical name 'top.L1.a2' inferred for 'net'

Example 2

Consider the following example when the `hierarchical_name_length` parameter is set to 12 and the `check_in_top` parameter is set to no:

```

module top(input in1, output out1); //No violation inside top module
    wire net2;

    blk inst1(in1, out1);

endmodule

module blk(input in, output o1); //No Violation for ports 'in' and
'out', Full hierarchical names: top.inst1.in and
top.inst1.o1, length = 12

wire net1; //Violation for net 'net1', Full hierarchical
name: top.inst1.net1, length = 14

```

```
reg r; //No Violation for net 'r', Full hierarchical name:  
top.inst1.r, length = 11
```

```
assign abc = in; //Violation for implicitly declared net  
'abc', Full hierarchical name: top.inst1.abc, length = 13
```

```
blk2 inst2(in, o1); //Violation for instance 'inst2', Full  
hierarchical name: top.inst1.inst2. length = 14  
endmodule
```

The following violations are reported for the above example:

Long hierarchical name 'top.inst1.net1' inferred for 'net'

Long hierarchical name 'top.inst1.abc' inferred for 'net'

Long hierarchical name 'top.inst1.inst2' inferred for
'instance'

Tag Severity

Error

MacroWithoutUndef-ML

Macro is not undefined

When to Use

Use this tag to enforce coding guideline to undefine Verilog macros

Description

The *MacroWithoutUndef-ML* tag reports a violation when a macro is not undefined.

The tag reports violation for the following scenarios:

- If a macro is defined within a Verilog file, it must be undefined later within a Verilog file. After processing all Verilog files, the tag detects all macros that are not undefined.
- If a macro is defined within a library Verilog file (that is, file read through -v or -y option), it must be undefined within the same file.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Macro '<macro_name>' is not undefined

Potential Issues

The violation message is reported when a macro is not undefined.

Consequences of Not Fixing

Verilog macros are globally visible. If a Verilog file defines a macro and not undefine it within the same file, all Verilog files subsequently read that macro definition. This can be problematic when integrating IP into larger design. If an IP defines a macro of commonly used name, for example, and does not undefine it, all other Verilog files within the same project are affected by it. There may be other IP in the same project that uses same

name of the macro for other purpose. This type of interaction may take long time to discover and debug.

The library Verilog files (that is, files read through -v or -y option) pose similar risk. These files are read after non-library Verilog files are read. Therefore, a macro defined in library does not affect non-library. But among library files, the effect of macro is hard to predict because order in which these files are processed is unpredictable.

How to Debug and Fix

Click the violation message to highlight the RTL instance where the macro is defined.

To fix the violation message, ensure that the macro is undefined.

Example Code and/or Schematic

For Verilog Files

Example 1

```
RTL1.v      `define AAA // violation; macro not undefined.  
    RTL2.v  
    RTL3.v
```

Example 2

```
RTL1.v      `undef AAA  
    RTL2.v  
    RTL3.v `define AAA // violation; both undef and define  
used but define comes is coming later
```

Example 3

```
RTL1.v      `undef AAA + `define AAA //violation; undef and  
define, but define comes later  
    RTL2.v  
    RTL3.v
```

Example 4:

```
RTL1.v      `define AAA + `undef AAA //no violation, undefined within the
```


MoreLint Tags

same file

```
RTL2.v
RTL3.v    `define AAA + `undef AAA
```

Example 5

```
RTL1.v    `define AAA
RTL2.v
RTL3.v    `undef AAA // no violation, not flag violation
           (undefined in different file)
```

For Library Verilog files**Example1**

```
RTL1.v `define AAA //violation, not undefined
```

Example 2

```
RTL1.v    `undef AAA + `define AAA //violation, undef and
           define, but define comes later
```

Example 3

```
RTL1.v    `define AAA + `undef AAA // no violation, undefined within the
           same file
```

Example 4

```
RTL1.v    `define AAA
RTL2.v    `undef AAA //violation, //define and undef, but
           not in the same Library Verilog file
```

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

MacroFileName-ML

FileName should be '<macroname>.v' or '<macroname>.sv' for a file, which contains only macro definition(s)

Language

Verilog

Description

The MacroFileName-ML Tag flags source files that contain only macro definitions but the source file name is not same as the name of one of the macros described in the source file.

The MacroFileName-ML Tag also checks source files that contain macro definitions and descriptions of unused tasks and functions.

For Tag-violating source files, the message reports the name of the first macro found in the source file.

Message Details

The following message appears for a Tag-violating source file where the first macro is named *<macro-name>*:

Use name '*<macro-name>.<file-extn>*' for file containing macro definition

Where *<file-extn>* can be *v* or *sv* depending upon the extension of the source file.

Severity

Guideline

MissingPortInModport-ML

All the ports of interface should be used in modport declarations

Description

The *MissingPortInModport* tag reports violation if all the ports of the interfaces, irrespective of the direction, are not used in modport declaration.

Parameter(s)

None

Constraint(s)

None

Language

Verilog

Messages and Suggested Fix

Message 1

The following message is reported when all the ports of the interfaces are not used in modport declaration:

[warning] All the ports of interface <interface-name> not used in modport <mod-portname> (Missing ports: <missing_port_list>)

Example Code and/or Schematic

Example 1

Consider the following example:

```
interface inf (input clk, rst, W_R, output addr);  
logic [7:0] status1;  
modport P1 (input addr, output status1); // Not all ports  
are used  
modport P2 (output addr, W_R, input status1); //Not all  
ports are used  
endinterface
```

Example 2

Consider the following example:

```
interface inf (input clk, rst, W_R, output addr);
logic [7:0] status1;
modport P1 (input W_R, clk, rst, output addr); // No violations
as all ports are used
modport P2 (input clk, rst, addr, output W_R); // No violations
as all ports are used, direction doesn't matter
endinterface
```

Example 3

Consider the following example:

```
interface inf (input clk, rst, W_R, output addr);
logic [7:0] status1;
modport P1 (input W_R, clk, rst, output addr, status1); // No
violations as all ports are used, and extra port found in the interface
modport P2 (input clk, rst, addr, status1, output W_R); // No
violations as all ports are used, direction doesn't matter, and extra port
found in the interface
endinterface
```

Example 4

Consider the following example:

```
interface inf (input clk, rst, W_R, output addr);
logic [7:0] status1;
modport P1 (input addr, clk, rst, output status1); // Not
all ports are used
modport P2 (output addr, W_R, clk, input status1); // Not
all ports are used
endinterface
```

Default Severity Label

Warning

MissingPinFunctionCall-ML

Use input pin for all read signals, which are not declared within function scope

Description

The *MissingPinFunctionCall-ML* tag reports function calls in continuous assignments, if it reads signals that are not declared in function scope.

A variable that is outside the scope of a function must not be read inside the function construct as changes in the variable outside the function might result in unknown function behavior, if it is called in a continuous assignment.

The *MissingPinFunctionCall-ML* tag does not report a violation for function calls within `always` constructs.

In addition, the tag does not check parameter/ local parameter usages in functions.

Language

Verilog

Parameters

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Function '<function name>', which calls in continuous assignment, reads non-local signal '<global variable name>'

Potential Issues

A violation is reported for function calls in continuous assignments, if the tag reads signals that are not declared in function scope.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

Click the violation message and VC SpyGlass highlights the RTL line of the function call.

Example Code and/or Schematic**Example 1**

Consider the following example:

```

module test(
    input in1 , in2 , in3 ,
    input clk , rst ,
    input [1:0] sel,

    output reg out1 , out2 , out3 );

    function decode(input c, d, e);
    begin
    case(sel)
    2'b01: decode = d | e;
    2'b10: decode = d & e;
    default: decode = c;
    endcase
    end
    endfunction

    assign out1 = decode( in1 , in2 , in3 ); // violation

    always @(in1, in2, in3, sel ) begin
    out2 = decode( in1 , in2 , in3 ); //no violation, because function call is in
    always block
  
```

```
end

always @(posedge clk) begin
  if(!rst)
    out3 <= 1'b0 ;
  else
    out3 <= decode( in1 , in2 , in3 ); //no violation, because function call is in
    always block
  end
endmodule
```

Example 2

Consider the following example:

```
module test(
  input in1 , in2 , in3 ,
  input clk , rst ,
  input [1:0] sel,

  output reg out1 , out2 , out3 );

function decode(input c, d, e, input [1:0]sel);
begin
  case(sel)
    2'b01: decode = d | e;
    2'b10: decode = d & e;
    default: decode = c;
  endcase
end
endfunction
```


assign out1 = decode(in1 , in2 , in3 , sel); //no violation, as function reads local variables only.

endmodule

Example 3

Consider the following example:

```
module test(
  input in1 , in2 , in3 ,
  input clk , rst ,
  input [1:0] sel,
```

```
  output reg out1 , out2 , out3 );
```

```
  function decode(input c, d, e);
  begin
  case(sel)
  2'b01: decode = d | e;
  2'b10: decode = d & e;
  default: decode = c;
  endcase
  end
  endfunction
```

```
  always @(in1, in2, in3 ) begin
```

```
    out1 = decode( in1 , in2 , in3 ); //no violation, because function call is in
    always block
```

```
  end
```

```
  always @( * ) begin
```

```
    out2 = decode( in1 , in2 , in3 ); //no violation, because function call is in
    always block
```

```
end
always_comb begin
  out3 = decode( in1 , in2 , in3 ); //no violation, because function call is in
  always block
end
endmodule
```

Default Severity Label

Warning

Reports and Related Files

None

MixedResetEdges-ML

Do not use same asynchronous reset with different polarities inside same process block

Language

VHDL

Description

The MixedResetEdges-ML Tag flags process blocks that use both edges of same variable as asynchronous reset.

Message Details

The following message appears when a reset signal *<reset-name>* is used with a different polarity in file *<file-name>* at line *<num>* inside the same process block:

Reset '*<reset-name>*' also used with different polarity (file: '*<file name>*' line: '*<num>*') inside same process block

Severity

Warning

Examples

The MixedResetEdges-ML Tag flags the following example, as the same reset signal `rst` is used with different polarities inside the same process block:

```
process (clk1, clk2, rst)
begin
    if (rst = '0') then    --rst signal used with
                           negative polarity
        q1 <= '0';
    elsif clk1'event and clk1 = '1' then
        q1 <= data;
    end if;

    if (rst = '1') then    --Same rst signal used
```

```

                                with opposite
                                polarity in same
                                block
        q2 <= '0';
    elsif clk2'event and clk2 = '0' then
        q2 <= data;
    end if;
end process;

```

The MixedResetEdges-ML Tag does not flag the following example as different reset signals are used with different polarities inside the same process block:

```

process (clk1, clk2, rst1, rst2)
begin
    if (rst1 = '0') then      --rst1 signal used with
                                negative polarity

        q1 <= '0';
    elsif clk1'event and clk1 = '1' then
        q1 <= data;
    end if;

    if (rst2 = '1') then      --Different signal rst2 is
                                used with positive
                                polarity in same block

        q2 <= '0';
    elsif clk2'event and clk2 = '0' then
        q2 <= data;
    end if;
end process;

```

The MixedResetEdges-ML Tag does not flag the following example as the same reset signal rst is used with different polarities inside different process blocks, P1 and P2:

```

P1: process (clk1, rst)
begin
    if (rst = '0') then      --Reset signal rst used

```

MoreLint Tags

```
                                inside block P1
        q1 <= '0';
    elsif clk1'event and clk1 = '1' then
        q1 <= data;
    end if;
end process;

P2: process (clk2, rst)
begin
    if (rst = '1') then          --Same signal rst used with
                                opposite polarity
                                inside block P2

        q2 <= '0';
    elsif clk2'event and clk2 = '1' then
        q2 <= data;
    end if;
end process;
```

MaxDelay-ML

Delay is more than the threshold value

When to use

Use this tag to check if the delay is more than the threshold value.

Description

The *MaxDelay-ML* tag reports a violation for sequential delay statements where the delay value is greater than the threshold delay (delta delay).

Language

Verilog

Parameters

- *delta_delay_mode*: By default, the value of the parameter is set to `sequential` and the tag reports a violation for only sequential delays. Set the value of the parameter as follows:
 - ☐ `combinational` - To report a violation for combinational delays.
 - ☐ `all` - To report a violation for all delays.
- *global_timescale*: By default, the value of the parameter is set to 0. Set the value of the parameter to any positive integer (in picoseconds) to set timescale for undefined modules.
- *max_delta_delay*: By default, the value of the parameter is set to -1 and the tag does not report any violation. Set the max delay value in picoseconds without `ps` as the value (accepted values are 0 and any positive integer) to configure delta delay.

Message Details

The following message appears when the delay value is greater than the threshold value:

[WARNING] Delay <delay-expression> (<value>) is greater than threshold <value>

Potential Issues

The violation message is reported when the delay value is greater than the

threshold value.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

Click on the violation message and SpyGlass points to the RTL statement line where there is a delay.

Example and /or Schematic

Example 1: Parameter: Default

Consider the following example:

```
`timescale 1ns/10ps
module test (input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // Delta Delay = 1 * 1ns = 1ns;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //Delta delay 2 * 1ns = 2ns
endmodule
```

In the above example, the tag does not report a violation as `max_delta_delay` is not specified.

Example 2: `max_delta_delay` "1000" value 1000 is in picoseconds

Consider the following example:

```
`timescale 1ns/10ps
module test (input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // line 5 Delta Delay = 1 * 1ns = 1ns;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //line 7 Delta delay 2 * 1ns = 2ns
endmodule
```

Example 3: max_delta_delay "2000"

Consider the following example:

```
`timescale 1ns/10ps
module test (input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // Delta Delay = 1 * 1ns = 1ns;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //Delta delay 2 * 1ns = 2ns
endmodule
```

Example 4: max_delta_delay "0"

Consider the following example:

```
`timescale 1ns/10ps
module test (input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; //Delta Delay = 1 * 1ns = 1ns;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; // Delta delay 2 * 1ns = 2ns
endmodule
```

Example 5: max_delta_delay "1000"

Consider the following example:

```
`timescale 1ns/10ps
module test (input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // line 5, Delta Delay = 1 * 1ns = 1ns;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //line 7, Delta delay 2 * 1ns =
2ns;
```



```

wire a,b;
assign a = #2 b; //line 9, Delta delay 2 * 1ns = 2ns;
endmodule

```

Example 6: max_delta_delay "1000" , delta_delay_mode "combinational"

Consider the following example:

```

`timescale 1ns/10ps
module test (input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // line 5 Delta Delay = 1 * 1ns = 1ns;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //line 7 Delta delay 2 * 1ns = 2ns
wire a,b;
assign a = #2 b; //line 9 Delta delay 2 * 1ns = 2ns
endmodule

```

Example 7: max_delta_delay "1000", delta_delay_mode "all"

Consider the following example:

```

`timescale 1ns/10ps
module test (input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // line 5, Delta Delay = 1 * 1ns = 1ns;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //line 7 Delta delay 2 * 1ns = 2ns
wire a,b;
assign a = #2 b; //line 9 Delta delay 2 * 1ns = 2ns
endmodule

```

Example 8: max_delta_delay "1000", delta_delay_mode "all"

Consider the following example:

```
`timescale 1ns/10ps
module test (input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // line 5 Delta Delay = 1 * 1ns = 1ns;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //line 7 Delta delay 2 * 1ns = 2ns
wire a,b;
assign a = #2 b; // line 9 Delta delay 2 * 1ns = 2ns
sub s();
endmodule
sub.v
module sub(input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // line 5 Delta Delay = 1 * 1ns = 1ns;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //line 7 Delta delay 2 * 1ns = 2ns
wire a,b;
assign a = #2 b; // line 9 Delta delay 2 * 1ns = 2ns
endmodule
```

Example 9: max_delta_delay "1000", delta_delay_mode "all"

Consider the following example:

```
`timescale 1ns/10ps
module test (input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // line 5 Delta Delay = 1 * 1ns = 1ns;
```

MoreLint Tags

```

else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //line 7 delta delay 2 * 1ns = 2ns
wire a,b;
assign a = #2 b; // line 9 Delta delay 2 * 1ns = 2ns
sub s();
endmodule

```

```

sub.v
`timescale 10ns/10ps
module sub(input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; //line 5 Delta Delay = 1 * 10ns = 10ns;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //line 7, delta delay 2 * 10ns =
20ns
wire a,b;
assign a = #2 b; // line 9 Delta delay 2 * 1ns = 2ns
endmodule

```

Example 10: max_delta_delay "1000", delta_delay_mode "all"

Consider the following example:

```

`timescale 1ns/10ps
module test (input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; //line 5 Delta Delay = 1 * 1ns = 1ns;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //line 7 delta delay 2 * 1ns = 2ns

```

```
wire a,b;  
assign a = #2 b; // line 9 Delta delay 2 * 1ns = 2ns  
sub s();  
endmodule
```

```
sub.v  
`timescale 100ps/10ps  
module sub(input in, rst, output [7:0] addr);  
always @(posedge clk or negedge rst)  
if (!rst)  
addr <= #1 0; //line 5 Delta Delay = 1 * 100ps = 100ps;  
else if ((addr < 8'hFF))  
addr <= #2 addr + 1'b1; //line 7 delta delay 2 * 100ps = 200ps  
wire a,b;  
assign a = #2 b; // line 9 Delta delay 2 * 100ps = 200ps  
leaf l();  
endmodule
```

```
leaf.v  
module leaf(input in, rst, output [7:0] addr);  
always @(posedge clk or negedge rst)  
if (!rst)  
addr <= #1 0; //line 5 Delta Delay = 1 * 100ps = 100ps;  
else if ((addr < 8'hFF))  
addr <= #2 addr + 1'b1; //line 7 delta delay 2 * 100ps = 200ps  
wire a,b;  
assign a = #2 b; // line 9 Delta delay 2 * 100ps = 200ps  
endmodule
```

Example 11: max_delta_delay "1000", delta_delay_mode "all"

Consider the following example:

```
`timescale 1ns/10ps
module test (input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; //line 5 Delta Delay = 1 * 1ns = 1ns
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //line 7 delta delay 2 * 1ns = 2ns
wire a,b;
assign a = #2 b; // line 9 Delta delay 2 * 1ns = 2ns
sub s();
endmodule
```

```
leaf.v
module leaf(input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // Delta Delay = 1 * 1ns = 1ns;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //delta delay 2 *1ns = 2ns
wire a,b;
assign a = #2 b; // line 9 Delta delay 2 * 1ns = 2ns
endmodule
```

```
sub.v
`timescale 100ps/10ps
module sub(input in, rst, output [7:0] addr);
```

```

always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // Delta Delay = 1 * 100ps = 100ps;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //delta delay 2 * 100ps = 200ps
wire a,b;
assign a = #2 b; // line 9 Delta delay 2 * 100ps = 200ps
leaf l();
endmodule

```

Example 12: max_delta_delay "1000", delta_delay_mode "all"

Consider the following example:

```

leaf.v
module leaf(input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // No delay is applicable at this point
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; // No delay is applicable at this point
wire a,b;
assign a = #2 b; // line 9 Delta delay 2 * 1ns = 2ns
endmodule

```

```

Test.v
`timescale 1ns/10ps
module test (input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // Delta Delay = 1 * 1ns = 1ns;

```

MoreLint Tags

```

else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //line 7 delta delay 2 * 1ns = 2ns
wire a,b;
assign a = #2 b; // line 9 Delta delay 2 * 1ns = 2ns
sub s();
endmodule

```

```

sub.v
`timescale 100ps/10ps
module sub(input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // Delta Delay = 1 * 100ps = 100ps;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //line 7 delta delay 2 * 100ps = 200ps
wire a,b;
assign a = #2 b; // line 9 Delta delay 2 * 100ps = 200ps
leaf l();
endmodule

```

Example 13: max_delta_delay "1000", delta_delay_mode "all", global_timesclae "10000"

Consider the following example:

```

leaf.v
module leaf(input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; //line 5 1 * 10ns = 10 ns

```

```
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //line 7 2 * 10ns= 20 ns
wire a,b;
assign a = #2 b; // line 9 Delta delay 2 * 10ns = 20ns
endmodule

Test.v
`timescale 1ns/10ps
module test (input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // Delta Delay = 1 * 1ns = 1ns;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //delta delay 2 * 1ns = 2ns;
wire a,b;
assign a = #2 b; // line 9 Delta delay 2 * 1ns = 2ns
sub s();
endmodule

sub.v
`timescale 100ps/10ps
module sub(input in, rst, output [7:0] addr);
always @(posedge clk or negedge rst)
if (!rst)
addr <= #1 0; // Delta Delay = 1 * 100ps = 100ps;
else if ((addr < 8'hFF))
addr <= #2 addr + 1'b1; //delta delay 2 * 100ps = 200ps
wire a,b;
assign a = #2 b; // line 9 Delta delay 2 * 100ps = 200ps
```


MoreLint Tags

```
leaf l();  
endmodule
```

Default Severity Label

Warning

MaxFaninLogic-ML

Report a violation on all flops that have drivers greater than ALLOWED_DRIVER_LIMIT

Description

The MaxFaninLogic-ML tag reports a violation if the number of valid drivers for a flip-flop after ignoring the combo cloud exceeds the value specified using the [ALLOWED_DRIVER_LIMIT](#) parameter. The tag starts its traversal from the data pin of the flip-flop. It ignores intermediate ports of sub-modules, clock gating cell, combo cloud, mux select pin, tri-enable pin to find the valid drivers. The stop point for the fan-in traversal is latch, flip-flop and primary port. However, the tag will not traverse through blocked path using set_case_analysis.

Parameter(s)

- [ALLOWED_DRIVER_LIMIT](#): The default value is 1024. Set the value of the parameter to any integer value to specify the allowed driver limit.
- [REPORT_ALL_DRIVERS](#): The default value is no. Set the value of the parameter to yes to report all valid drivers.
- [SKIP_MODULE](#): The default value is NULL. Set the value of the parameter to a comma-separated list of module names to specify the modules for which/ you want to skip checks.
- [IGNORE_LATCH](#): The default value is NULL. Set the value of the parameters to a comma-separated list of latch names that you want to ignore during traversal.

Message Details

Flop '<flip-flop-name>' has high fan-in count ([<fanin_count>]) that exceeds the max fan-in count [max_fanin_Count]

Severity

Warning

ModuleName-ML

Module name does not follow the naming convention

Language

Verilog, VHDL

Description

The ModuleName-ML Tag flags design unit names that do not follow the recommended naming convention.

By default, following is the recommended naming convention for design unit names:

- The design unit name should start with an uppercase alphabet (A-Z).
- The design unit name can contain alphabets, numbers, and underscore character only.
- The design unit name should not contain two successive underscore characters (causes problems for some hardware accelerators and emulators)

Use the [modulename](#) Tag parameter to specify your own design unit naming convention.

NOTE: *The ModuleName-ML Tag is switched off by default.*

Message Details

The following message appears for a Tag-violating design unit name `<du-name>`:

Module '`<du-name>`' does not follow the recommended naming convention

Severity

Guideline

Suggested Fix

Modify all references to the module name to a form that meets the convention.

MultiModuleInCellDefine-ML

**There are more than one module definitions in between a single
'celldefine and 'endcelldefine**

Language

Verilog

Description

The MultiModuleInCellDefine-ML Tag flags 'celldefine/'endcelldefine constructs containing the description of more than one module.

Message Details

The following message appears at the location of a 'celldefine/'endcelldefine construct that contains *<num>* module descriptions:

More than one (<num>) module definitions found between
'celldefine and 'endcelldefine

Severity

Info

MultiOpInModule-ML

Circuit of multiple operators are generated in the same module

Language

Verilog, VHDL

Description

The MultiOpInModule-ML Tag flags multiple operators used in the same design unit.

The MultiOpInModule-ML Tag flags design units when any of the base operators is used along with another base operator or any other operator.

The MultiOpInModule-ML Tag checks for the base operators — addition operator (+), subtraction operator (-), multiplication operator (*), division operator (/), and the modulus operator (%) in Verilog and mod in VHDL).

The MultiOpInModule-ML Tag generates a message when:

- if condition and base operator is present inside a module

```
if (D==0) A+B ;
```

- case and base operator is present inside a module

```
case (A)
when 2'00: A+B ;
```

By default, the MultiOpInModule-ML Tag does not check in sequential blocks inferring flip-flops. To enable checking in sequential blocks inferring flip-flops, set the [check_op_in_flop](#) parameter to yes.

By default, the MultiOpInModule-ML Tag checks for complex operators. To enable checking base operators, set the [check_complex_operators](#) to no.

By default, the MultiOpInModule-ML Tag flags only five messages for each design unit. Set the [allviol](#) Tag parameter to yes or *<Tag-name>* to report all messages for each design unit.

Message Details

The following message appears for a design unit that has multiple operators used:

More than one operator used inside module

Severity

Warning

MultOperVar-ML

Avoid multiple operations on variable in the same expression

Description

The *MultOperVar-ML* Tag checks the presence of the multiple operations on a variable in the same expression.

When there are multiple operations on a variable in a single expression, different results from different simulators are obtained. The evaluation order is undefined when a compound expression modifies the same variable multiple times on the right-hand side of an assignment statement.

System Verilog has increment (++) and decrement (--) operators, which when used multiple times on the same variable in an expression can cause ambiguity, so this Tag reports violation for such expressions.

Language

Verilog

Parameters

None

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following information message is reported when multiple operations are present on a variable in the same expression:

[WARNING] Avoid multiple operations on variable '<var-name>' in the same expression [Hierarchy: '<hier-path>']

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

Avoid using operators that make multiple reads and writes to a variable within the same expression. Using these operators can lead to ambiguous results.

Example Code and/or Schematic

Example 1

Consider the following code snippet:

```
i = 10;
j  = --i + ++i;  //Multiple operations on same variable
```

In this example, the value of *i* is both read and modified multiple times on the right-hand side of the assignment statement. The System Verilog standards do not guarantee the order of evaluation and execution of these multiple read/writes to the same variable in the same expression. Hence the *MultOperVar-ML* Tag reports a violation.

The correct model depends on what order you intended the operations on the variable *i* to occur. One way is:

```
always_comb
begin
    temp = --i;  //pre-decrement i, save result in temp
    j = temp + ++i;  //add temp and i, then increment i
end
```

Example 2

In the following example, the *MultOperVar-ML* Tag reports violations:

```
always @(posedge clk)
begin
    if(i++ + ++i)  //violation for i
        i<=j;
end

always @(posedge clk)
begin
    j = i++ * j++ / ++ i;  //violation for i, not for j
    j = i++ + i-- + j++ + j--; //violation for i and j
```



```
end
```

Example 3

Consider the following example:

```
always @(posedge clk)
begin
    if((i==0) ? i++ + i-- : ++i) //violation for i on the
        LHS of the conditional operator
    j = (i==0) ? i++ + i++ : ++i + i++; //violation for i
        on both, LHS and RHS of the conditional operator
end
```

Example 4

In the following example, the *MultOperVar-ML* Tag reports violations for the variable *i* when :

```
genvar p ;
generate
    for(p=1 ;p<2;p++)
    case(p)
        1 : begin:A
            always @(posedge clk)
            begin
                j = i++ + --i; //violation for i
            end
        end
        2 : begin:B
            always @(posedge clk)
            begin
                j = i++ + i--; //no violation for i, this case not reached
            end
        end
    endcase
endgenerate
```

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

MultipleFilesCellDefine-ML

Scope of ``celldefine` spans across multiple files.

NOTE: *This Tag will be deprecated in a future release as it's functionality is covered by the WRN_1038 Built-In Tag.*

Language

Verilog

Description

The MultipleFilesCellDefine-ML flags 'celldefine/endcelldefine constructs that span multiple source files.

Message Details

The following message appears at the location of a 'endcelldefine statement (in file `<file2-name>` at line `<line-num2>` when the corresponding 'celldefine statement is found in another source file `<file1-name>` at line `<line-num1>`:

```
`celldefine-`endcelldefine spanning across multiple files  
encountered (`celldefine found in file <file1-name> at line  
<line-num1>, `endcelldefine found in file <file2-name> at line  
<line-num2>)
```

Severity

Info

NegativeIndex-ML

Do not use negative or enum array index

When to Use

Use this Tag to detect negative or enumeration values, which are used as an array index.

Description

The NegativeIndex-ML tag detects negative or enumeration value used as an array index. An array index should be within the range because of the following reasons:

- Tag does not report any violation for the while/for loop index variable.
- Tag unrolls the generate-for and checks whether the index present inside the generate-for is negative or not.
- Ensure that index is one of the following parameter type, to be checked whether it's negative or not:
 - ☐ genvar
 - ☐ const int
 - ☐ static const
- Tags does not perform value propagation and does not handles the non-constant indices.

Parameter(s)

- [*ignore_enum_index*](#): Default value is no. Set the value of the parameter to yes to report violations for non-static bit expressions.
- [*ignore_enum_index*](#): Default value is no. Set the value of the parameter to yes to ignore violations for enum indices.
- [*report_declarations*](#): Default value is no. Set the value of the parameter to yes to report violations for declarations with negative indices.

Constaint(s)

None

Messages and Suggested Fix

Message 1

[ERROR] <Negative|Enum> Index '<index-string>' found for array '<array-string>' (dimension no: '<dimension-no>') [Hierarchy: <hierarchy-string>]

Potential Issues

The violation message is reported because there is a negative or enumeration values are detected as an array index.

Consequences of Not Fixing

Not fixing the violation may result in unexpected Tag behavior.

How to Debug and Fix

Click on the violation message to cross-probe to the HDL description. Here, Tag points to the usage of bit-sel/part-sel/array-bit-selects wherever negative or enum index is present.

Message 2

[ERROR]<node-type> '<node-name>' declared with negative index(es) (value: '<value>', dimension no: '<dimension-no>') [Hierarchy: <hierarchy-string>]

Potential Issues

The violation message is reported because a node is declared with negative index.

Consequences of Not Fixing

Not fixing the violation may result in unexpected behavior.

How to Debug and Fix

Click on the violation message to cross-probe to the HDL description. Here, the tag points to the line where negative indexes are declared.

Example Code and/or Schematic

Example 1

Consider the following example:

```

module top( ck, data_in, select, data_out);
    input select, ck;
    input [-1:0] data_in;
    output [0:-1] data_out;
endmodule

```

For the above example, no violation is reported as there is no usage of negative index.

Example 2

Consider the following example:

```

generate for (i=0; i<=2; i=i+1)
begin : block1
for ( k = 0; k <=2; k = k + 1)
if(i >= k)
assign even[i-k] = dgrpipe [i-k];
else
assign even[k-i] = dgrpipe [k-i];
end
endgenerate

```

For the above example, no violation is reported as Tag will unroll the generate loop and will check the indexes.

Example 3

Consider the following example:

```

assign b[-1][-11][-3:-4] = a[-10][-1][-5+:2]; //
Violation

```

For the above example, violation is reported flagged for each dimension and dimension no and signal name is mentioned in the violation message.

Example 4

```

enum int { WHITE, YELLOW, BLUE, GREEN, RED, BLACK } color_1;
wire [3:0] wir_1;
wire [0:-3] wir_2;
assign wir_1[WHITE] = 1'b0; // No violation since the
ignore_enum_index=yes

```

MoreLint Tags

```
assign wir_1[GREEN:BLUE] = 2'b10; // No violation since the
ignore_enum_index=yes
```

```
assign wir_2[-3] = 1'b0; // violation
assign wir_2[0:-2] = 3'b1; // violation
```

For the above example, no violation is reported for wir_1 as the value if the ignore_enum_index parameter is yes. Also, the Tag reports violation for wir_2.

Example 5

Consider the following example:

```
module top( ck, data_in, select, data_out);
    input select, ck;
    input [1:0] data_in;
    output [0:1] data_out;

    C8T28SOI_LL_SDF2SYNCPQX5_P0
    C8T28SOI_LL_SDF2SYNCPQX5_P0_INST( .CP (ck), .D (data_in[-
1]), .TI (select), .TE (select), .Q (data_out[0]) ); //
violation

    C8T28SOIDV_LL_2SDFPQX10_P0
    C8T28SOIDV_LL_2SDFPQX10_P0_INST ( .CP (ck), .D
(data_in[0]), .TI (select), .TE (select), .Q (data_out[-
1]) ); // violation

endmodule
```

For the above example, the Tag reports violations because there is a negative index usage in data_in and data_out.

Default Severity Label

Error

Tag Group

None

Reports and Related Files

None

NestedInterfaceInst-ML

Detect interface instances in interfaces

When to Use

Use this tag to detect interface instances within an interface.

Description

The *NestedInterfaceInst-ML* tag reports a violation for interface instances within an interface.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

```
[WARNING] Interface instance '<instance_name>' of type  
'<parent_interface_name>' detected in interface  
'<interface_name>'
```

Potential Issues

A violation message is reported when an interface instance is detected within an interface.

Consequences of Not Fixing

Not fixing the violation message may result in unexpected synthesis/simulation.

How to Debug and Fix

Click the violation message and VC SpyGlass Lint highlights the RTL line where the interface instance exists.

Example Code and/or Schematic

Example 1

```
interface Intfs_1(Intfs_2 inst, logic [3:0] sum); //
violation

reg p;
endinterface

interface Intfs_2;
reg q;
endinterface

module test();
Intfs_2 a(); // no-violation, not in an interface
endmodule
```

Example 2

```
interface Intfs_1(input in, output out);
reg p;
Intfs inst(); // violation
endinterface

interface Intfs_2;
reg q;
endinterface

module test();
endmodule
```

Example 3

```
interface Intfs_2;
reg q;
endinterface

module test();
interface Intfs_1(input in, output out);
reg p;
Intfs inst(); // violation
endinterface
endmodule
```

Example 4

```
interface I1(I2.m1 i2_inst , logic [3:0] sum ); // no-
violation for 'i2_inst'
modport m2 (output sum);
I2 inf_inst();
endinterface

interface I2;
logic [3:0] address;
modport m1 (input address); // no violation
endinterface

module test (I1 a );
endmodule
```

Default Severity Label

Warning

Reports and Related Files

No related reports or files.

NeverExecutingForLoop-ML

Zero loop count

Description

The *NeverExecutingForLoop-ML* Tag reports a violation if the condition of the `for` loop is not evaluated, which always remains false.

Language

Verilog

Parameter(s)

[disable_rtl_deadcode_extended](#): Default value is no. Set the value of the parameter to yes to not report violations for dead codes.

Message Details

The following message appears if condition of the `for` loop is not evaluated that remains false:

[WARNING] Never executing loop (Index: '<index>' Condition: '<condition>') detected [Hier: <hier-name>]

Example and /or Schematic

Consider the following example where the Tag reports a violation because the `for` loop is not executed as condition is not evaluated:

```
module TOP(a, out);
input a;
output [3:0] out;
reg [3:0] out;
integer i;
always @(a)
    for(i = 4; i <= 2; i = i + 1) //Violation
        out[i] = 0;
endmodule
```

Default Severity Label

Warning

NoArithOp-ML

Arithmetic Operators should be avoided.

Language

Verilog, VHDL

Description

The NoArithOp-ML Tag flags use of arithmetic operators other than addition and subtraction operators.

The NoArithOp-ML Tag flags use of multiplication operator (*), division operator (/), and modulus operator (%) in Verilog and multiplication operator (*), division operator (/), remainder operator (rem), and modulus operator (mod) in VHDL.

The NoArithOp-ML Tag flags the following Verilog expressions as highlighted:

```
...
E = (A+B) / (C+D);
F = (A+B) % (C+D);
G = (A+B) * (C+D);
...
```

The NoArithOp-ML Tag flags the following VHDL expressions as highlighted:

```
library ieee;
use ieee.std_logic_1164.all
```

```
...
x <= i/div;
y <= i mod div;
z <= div rem i;
a <= div * i;
...
```

The NoArithOp-ML Tag ignores use of operators in computing units that is, when the bit-select or part-select is computed using these operators as in the following examples:

```
C[W*2-1:0] = A[W*2-1:0]; //Verilog
```

```
C(W*2-1 downto 0) <= A(W-1 downto 0); --VHDL
```

Message Details

The following messages appear for expression *<expr>* that uses unsupported operator *<opr>*:

Expression '*<expr>*' containing '*<opr>*' operator

Severity

Warning

NoArithShiftInConcat-ML

Tag will report violation for arithmetic operations inside concatenation.

When to Use

Use this Tag to report violation for arithmetic operations inside concatenation

Description

The NoArithShiftInConcat-ML Tag reports violation for arithmetic/shift operator that is used inside a concatenation expression. In addition, the tag also detects unary ops such as +/- (e.g. {1'b0, -b}).

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

Arithmetic/Shift operator '<operation>' used in concatenation '<expression>' [hierarchy '<hierarchy>']

Potential Issues

A violation message is reported when an arithmetic/shift operator is used in a concatenation expression.

Consequences of Not Fixing

Not fixing the violation message may result in unexpected Tag behavior.

How to Debug and Fix

Click the violation message to cross-probe to the HDL description. On clicking the violation message in console line, SpyGlass highlights the concatenation assignment statement in the HDL viewer.

Example Code and/or Schematic

Case 1: concat in always

```
module test;
reg [7:0] wardr;
reg [3:0] win;
always_comb begin
wardr = {win + 2, win}; //violation as operation + used
inside concatenation.
end
endmodule
```

case 4: concat in 'if'

```
module test;
reg [7:0] wardr;
reg [3:0] win;
if (wardr == {win >> 2, win}) //violation as operation >>
used inside concatenation....;
endmodule
```

case 5: interface used

```
module test;
Interface_temp inf();
endmodule
interface Interface_temp();
reg [7:0] wardr;
reg [3:0] win;
always_comb begin
wardr = {win >> 2, win}; //violation as operation >> used
inside concatenation
end
endinterface
```

case 6: interface not used

MoreLint Tags

```

module test;
endmodule
interface Interface_temp();
reg [7:0] wardr;
    reg [3:0] win;
always_comb begin
wardr = {win >> 2, win}; //no violation as interface is not used.
end
endinterface

```

case 7: function used

```

module test;
reg [7:0] wardr;
    reg [3:0] win;
assign wardr = myfunc(win);
endmodule
function myfunc;
    input win;
    begin
        myfunc = {win >> 2, win}; //violation as operation >>
use inside concatenation
    end
endfunction

```

case 8: function not used

```

module test;
reg [7:0] wardr;
    reg [3:0] win;
endmodule
function myfunc;
    input win;
    begin
        myfunc = {win >> 2,win}; //no violation as function is not used
    end
endfunction

```

case 9: with packages

```
package pkg;
    parameter s = 2;
    parameter w = {s >> 2, s}; //violation as operation >>
use inside concatenation
endpackage

`include "pkg.v" // include the file
import pkg::*;   // import the package

module test;
    reg [7:0] wardr;
    reg [3:0] win;
endmodule
```

Case 10: with logical operators

```
module test;
    reg [7:0] a;
    reg [3:0] b;
    m1 inst3 (.in({(a + b) & b}, a)); //violation as
operation + used inside concatenation and NO violation for
&

    m1 inst4 (.in({(a | b) - c}, b)); //violation as
operation - used inside concatenation and NO violation for
|

endmodule

module m1 (input [8:0] in);
endmodule
```

Case 11: replications/multiple concats

```
module test;
    reg [7:0] a;
```

MoreLint Tags

```

reg [3:0] b;

m1 inst3 (.in({c{a + b, a}})); //violation as operation +
used inside concatenation

m1 inst4 (.in({c{a - b}}, b)); //violation as operation
- used inside concatenation

endmodule

module m1 (input [8:0] in);
endmodule

```

Case 12: different bit selects

```

module test (
    input [7:0] in,
    output reg [7:0] out
);
    shortint S [7:0];
    longint I [7:0][3:0];
    logic [3:0] L;
    byte B [7:0];

    parameter C = 0;
    parameter D = 0;

    initial
        out[{B[3:C + 0], D}] = 1'b0; No violation

    always @(in)
    begin
        out[S[0]] = {S[0], S[C + 0]}; // VE_BITSELECT NO violation
        out[I[0][0]] = {I[0][0], I[C + 0][D + 0]}; // VE_ARRAYBITSELECT
NO violation
        out[L[3:1]] = {L[0], L[C + 3 : 1]}; // VE_PARTSELECT NO
violation
    end

```

```

        out[B[0][3:0]] = {B[0], B[C + 0][D + 3: 0]};//
VE_ARRAYPARTSELECT NO violation

    end
endmodule

```

Case 13: with common expr

```

module test;
reg [7:0] wardr;
reg [3:0] win;
    parameter P = 1;
always_comb begin
    wardr = {P{P + 2}}; //violation as operation + used
inside concatenation.
    wardr = {2 + P{1'b0}}; // violation as operation +
used inside concatenation. { (2+P) {1'b0} }
    wardr = {2 + {P{1'b0}}}; // violation as operation
+ used inside concatenation.
    wardr = {{2 + P{1'b0}}, win}; //violation as
operation + used inside concatenation.
    end
endmodule

```

Example 14: streaming concats

```

module test;
initial begin
    byte a = 8'h8C;
    byte b = 8'h00;
    byte c = 8'hA0;
    byte d = 8'hBC;
    reg [31:0] value = {>>{a + b, b, c, d}}; //violation as
operation + used inside concatenation.
end
endmodule

```

MoreLint Tags

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

NoArray-ML

Detects usage of arrays in design

Language

Verilog

Description

The NoArray-ML Tag flags arrays and memories used in design.

By default, the number of messages has been restricted to one per module and 100 per design. Set the [allviol](#) Tag parameter to `yes` or `<Tag-name>` to report all violation messages.

Message Details

The following message appears at the location where an array `<array-name>` is declared:

Array '`<array-name>`' used in the design

Severity

Warning

NoBitArray-ML

Do not use one-pin buses.

Language

Verilog, VHDL

Description

The NoBitArray-ML Tag flags use of single-pin buses.

Single-pin buses cause timing issues and back-end implementation issues.

Message Details

The following message appears at the location where a one-pin bus *<bus-name>* is declared at hierarchy *<hier-path>*:

```
Do not use one-pin bus signal '<bus-name>' [Hierarchy: '<hier-path>']
```

Severity

Warning

Suggested Fix

Avoid using one pin buses.

NoConstSourceInAlways-ML

Checks the use of constant/inferred constants in RHS always block

When to Use

Whenever an implicit always block (always@ *) is used in design.

Description

The Tag reports violation whenever a constant is assigned to a variable inside an implicit always block (always @*).

Default Weight

10

Language

Verilog, System Verilog

Parameter(s)

ignore_non_static_cond: By default, the value of the parameter is set to no. Set the value of the parameter to yes to not report a violation for non-static case/if conditions. When a condition is a combination of static and non-static values, violations are still not reported.

Constraint(s)

None

Messages and Suggested Fix

The following message appears when constant/inferred constants in RHS always block are used:

[WARNING] signal <signal_name> is assigned a constant value <constant_expression> in always block

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

Double click the violation message. The HDL viewer window highlights the line where violation is detected. Fix the violation accordingly by changing the code. Use an `assign` assignment.

Example Code and/or Schematic

In the following example, the *NoConstSourceInAlways-ML* Tag reports a violation because an assignment of constant source `1'b0` is done to `sig1` within an implicit always block (`always@*`):

```
always @ *  
begin  
    sig1 = 1'b0; //Violation for NoConstSourceInAlways-ML  
    sig2 = sig0;  
end
```

The following message is reported for the above example:

```
Signal 'sig1' is assigned a constant value '1'b0' in implicit  
always block
```

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

NoFuncOrProc-ML

Avoid use of functions and procedures

Language

VHDL

Description

The NoFuncOrProc-ML Tag flags use of functions and procedures in a design unit as in the following example:

```
library ieee;
use ieee.std_logic_1164.all

architecture arch of ent is
    ...
    procedure f1 (ARG1 : std_logic) is
    ...
    end procedure;
    ...
end arch;
```

NOTE: *The NoFuncOrProc-ML Tag is switched off by default.*

Message Details

The following message appears at the first line of a function or procedure body:

Avoid use of functions and procedures

Severity

Warning

Suggested Fix

Remove functions and procedures from your design if your design methodology does not allow them.

NoGenericMap-ML

Instantiation must have generic map for a generic declared in the entity/component

Language

VHDL

Description

The NoGenericMap-ML Tag flags component instantiations without generic maps when the corresponding entity/component has one or more generics defined.

Message Details

The following message appears at the location of an instantiation where a generic map is not defined when the corresponding entity/component *<entity-name>* has one or more generics defined:

Generic map does not exist for each generic declared in entity '*<entity-name>*'

Severity

Warning

Examples

The NoGenericMap-ML Tag flags the following example where the instantiation `inst1` of component `test` does not have a generic map:

```
entity top is
  port ( in1 : inout bit );
end entity;

architecture behav of top is
  component test is
    generic ( abc : integer := 1);
    port ( in1: in bit; out1: out bit );
  end component;
  signal sig1, sig2 : bit;
```

```
begin
  inst1 : test port map ( in1 => sig1, out1=>sig2 );
end behav;
```

NoOpen-ML

Flag where open keyword is used

Language

VHDL

Description

The NoOpen-ML Tag flags use of open keyword.

Message Details

The following message appears at the location where the open keyword is used:

open keyword found

Severity

Warning

NoVerilogPrims-ML

Verilog primitives are prohibited

Language

Verilog

Description

The NoVerilogPrims-ML Tag flags all the Verilog primitives used in the design.

Verilog primitives such as `and` as well as `or` must not be used. Following this guideline eases understanding of the HDL.

Given below is the list of Verilog primitives:

<code>and</code>	<code>nand</code>	<code>or</code>	<code>nor</code>	<code>xor</code>	<code>xnor</code>	<code>not</code>
<code>buf</code>	<code>bufif0</code>	<code>bufif1</code>	<code>notif0</code>	<code>notif1</code>	<code>nmos</code>	<code>pmos</code>
<code>rnmos</code>	<code>rpmos</code>	<code>tran</code>	<code>rtran</code>	<code>tranif0</code>	<code>tranif1</code>	<code>rtranif0</code>
<code>rtranif1</code>	<code>cmos</code>	<code>rcmos</code>	<code>pullup</code>	<code>pulldown</code>		

Message Details

The following message appears when a Verilog primitive is encountered:

Don't use Verilog primitives (primitive: '<name>')

Where,

<name> is the name of the primitive used in the design.

Severity

Warning

NonBlockingCounters-ML

Non-blocking counter assignment found

When to Use

Use this tag to detect non-blocking counter assignments.

Description

This tag reports violations for non-blocking counter assignments. Non blocking counter assignments are defined in the following format:

```
a <= a + c;
```

Where, 'c' can be any number.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

Message

[warning] Non blocking counter assignment found for register '<reg_name>' in expression '<counter_statement>' [Hierarchy: '<hierarchy>']

NOTE: Duplicate violations are discarded even if the hierarchies do not match.

Potential Issues

A violation is reported is when there is a non-blocking counter assignment in the design.

Consequences of Not Fixing

This is an informative tag about the design. Review each violation.

How to Debug and Fix

Double click the violation message. The HDL viewer window highlights the RTL line where the nob-blocking counter assignment is found.

Example Code and/or Schematic

Example 1

```
reg [1:0] counter;  
always @(posedge clk or negedge arst) begin  
  if (!arst)  
    counter <= 2'b00;  
  else if (counter != 2'b11)  
    counter <= counter + 2'b01; // register 'counter' is used  
    as a counter assignment [hierarchy: <hierarchy>]  
  else  
    counter <= 2'b00;  
end
```

Example 2

```
wire [1:0] counter;  
assign counter = counter + 1; //not a non blocking counter
```

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

NonConstShift-ML

Checks whether there is any non-constant shift expression (Right operand).

When to Use

Use this Tag to check for the presence of a non-constant shift expression.

Description

This Tag reports a violation when there is a shift operation (both logical and arithmetic shift), whose right operand is not a static constant value.

Parameters

None

Constraints

None

Messages and Suggested Fix

[WARNING] Number of bits to shift ("<variable name>") should be a constant. [Hierarchy: < Hierarchy >]

Potential Issues

The violation message is reported when a non-constant shift expression is present in the design.

Consequences of Not Fixing

Not fixing the violation may result in unexpected behavior.

How to Debug and Fix

Click on the violation message to highlight the RTL line where shift expression is used.

To fix the violation, change RHS of shift operator to a static constant value or expression.

Example Code and/or Schematic

Example 1

```
module test (a, clk, num, b);
  input clk;
  input [7:0] a, num;
  output [7:0] b;
  reg [7:0] b;

  always @(posedge clk)
    b = a << num; //warning on "num", not a static constant
endmodule
```

Example 2

```
module top (a, clk, num, b);
  input clk;
  input [7:0] a, num;
  output [7:0] b;
  reg [7:0] b;

  localparam l_Param = 1;
  parameter param = 2;

  reg [1:0] _reg;
  reg [1:0] _wire;

  always @(posedge clk) begin
    b = a << 1;          //no warning, a static constant
    b = a << l_Param;    //no warning on "l_param", a static constant
    b = a << param;     //no warning on "param", a static constant

    b = a << _reg;       //warning on "_reg", not a
static constant
    b = a << _wire;      //warning on "_wire", not a
static constant
    b = a << num;        //warning on "num", not a static
constant
```

```

    end
endmodule

```

Example 3

```

module top (a, clk, num, b);
input clk;
    input [7:0] a, num;
    output [7:0] b;
    reg [7:0] b;

    always @(posedge clk) begin
        if((a << num ) == 7'b0000000) //warning on "num",
not a static constant
            b = 7'b11111111;
        else
            b = a << num ;    //warning on "num", not a
static
                                constant
    end
endmodule

```

Example 4

```

module top (a);
    input [1:0] a;
    reg[3:0] x;
    reg[3:0]y;

    localparam l_Param = 1;
    parameter  param    = 2;

    reg [1:0] _reg;
    reg [1:0] _wire;

    initial begin
        y = x >> 1;           //no warning, static constant
        y = x >> l_Param;    //no warning on "l_param", a static constant
    end
endmodule

```

```

        y = x >> param;    //no warning on "param", a static constant

        y = x >> _reg;     //no warning , no warning in initial block

        y = x >> _wire;    //no warning , no warning in initial block

        y = x >> a ;       //no warning , no warning in initial block

        y= x <<< 1 ;       //no warning, static constant
        y = x <<< l_Param ; //no warning on "l_param", a static
constant

        y = x <<< param;    //no warning on "param", a static constant

        y = x <<< _reg;     //no warning , no warning in initial block

        y = x <<< _wire ;   //no warning , no warning in initial block

        y = x <<< a ;       //no warning , no warning in initial block

    end
endmodule

```

Example 5

```

module top ();
    reg[3:0] a;
    reg[3:0] b;

    integer s_int = 3;
    integer i = 0;

    initial begin
        a = 4'b1111;
        b = 4'b1111;

        b = a << 1; //no warning, static constant
        b = a << s_int ; //no warning , no warning in initial block
    end
endmodule

```

MoreLint Tags

```

        for( i = 0 ; i < 4 ; i = i + 1) begin
            b = a << i; //no warning , no warning in initial block
        end
    end
endmodule

```

Example 6

```

module top(clk);
input clk;
reg [3:0] a_reg;
reg [3:0] b_reg ;
reg [1:0] c_reg;

localparam val_param = 2'b10;

always @(posedge clk) begin
    task_1(b_reg , a_reg , val_param);
    task_1(b_reg, a_reg, c_reg);
end

    task task_1;
        output [3:0]b;
        input [3:0]a ;
        input [1:0]val_1;
        begin
            b = a << val_1; //warning on "val_1", not a
static constant
        end
    endtask

    task task_2;
        output [3:0]b;
        input [3:0]a ;
        input [1:0]val_2;

        begin
            b = a << val_2; //no warning, task_2 is not used

```

```

end
    endtask

```

```

endmodule

```

Example 7

```

module top(input clk);
    reg [3:0] a_reg;
    reg [3:0] b_reg ;
    reg [3:0] c_reg;

    reg [1:0] val_reg;
    localparam val_param = 2'b10;

    always @(posedge clk) begin
        b_reg = func_1( a_reg , val_param);|
        c_reg = func_1( a_reg , val_reg);
    end

    function [3:0] func_1;
        input [3:0]a ;
        input [1:0]val_1;

        func_1 = a << val_1; //warning on "val_1", not
a static constant
    endfunction
endmodule

```

Example 8

```

interface intf( input [7:0]a , input [7:0]b , output
[7:0]out);
    assign out = a << b; //warning on "b", not a static
constant
endinterface

    module top (a, clk, num, b);
        input clk;

```


MoreLint Tags

```
input [7:0] a, num;
output [7:0] b;
reg [7:0] b;

intf intf_1( a, num , b);
endmodule
```

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

NoFeedThru-ML

Do not use feedthrough signals

When to Use

Use this tag to detect feedthrough signals.

Description

The *NoFeedThru-ML* tag identifies all outputs of a module and analyzes its driver. The tag reports a violation when an output is directly connected to an input (bit of bus signal included).

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] There is a path from input '<port1-name>' to output '<port2-name>' with specified instances [Module Instance list: '<instance-list>']

Potential Issues

The violation message is displayed when there is path from input port to output port.

Consequences of Not Fixing

Not fixing the violation may cause back annotation issues.

How to Debug and Fix

To fix the violation, avoid using feedthrough signals.

Example Code and/or Schematic

Currently Unavailable

MoreLint Tags

Severity

Warning

Reports and Related Files

None

NonLocalLoopIndex-ML

Catches loop indexes that are not locally defined

Description

The *NonLocalLoopIndex-ML* tag checks loop indexes and reports a violation when loop indexes are found to be globally defined.

NOTE: *The tag does not report violation for genvars.*

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Loop index '<index-string>' is not local for current loop

Potential Issues

The violation message is reported when indexes are found to be globally defined.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug And Fix

Click the violation message and VC SpyGlass Lint points to the line where the loop index is not defined locally.

Example Code and/or Schematic

Consider the following code:

```
module top(input [7:0] in, output [7:0] out);
```

MoreLint Tags

```
integer N;  
always@(in)  
for(N=0;N<8;N+1)  
out[N] = in[N];  
endmodule
```

In the above example, the variable of integer type is caught as a violation because it is not defined locally. To void this violation, the for loop should use local variable as loop index. For example:

```
for(integer N=0;N<8;N++)  
out[N] = in[N];
```

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

NonOverridenParams-ML

Tag reports violation for non-overridden parameters

When to Use

Use this parameter to detect parameters whose values are not explicitly overridden in the module instantiation.

Description

The *NonOverridenParams-ML* Tag reports violations for parameters whose values are not explicitly overridden in the module instantiation.

Parameter(s)

- **report_all_params**: By default, the *NonOverridenParams-ML* tag reports only one violation for the module instance. Set the value of the `report_all_params` parameter to `yes` to enable the *NonOverridenParams-ML* Tag to report violations for all the parameters.
- **ignore_libcell_inst**: By default, the parameter is set to `no`. Set the value of the parameter to `yes` to enable the *NonOverridenParams-ML* tag to ignore violations for library cell instances.

Messages and Suggested Fix

[WARNING] Parameter <Parameter Name> is not overridden in instantiation <Instance name> of module <Parent Module Name>.
[Hierarchy: < Hierarchy Path>]

Potential Issues

The violation message is reported when the specified parameter is not explicitly overridden in the module instantiation.

Consequences of Not Fixing

Not fixing the violation may result in unexpected Tag behavior.

How to Debug And Fix

Click the violation message points to the line where the module instantiation happened in the RTL code.

Example Code and/or Schematic

Example 1

```
module test2(input in, output out);
parameter AB=20;
parameter CD=20;
assign out = in;
endmodule

module test();
test2 #(.AB(5)) T1 (); //Violation - CD not overridden
endmodule
```

Example 2

```
module test2(input in, output out);
parameter AB=20;
parameter CD=10/AB;
assign out = in;
endmodule

module test();
test2 #() T1 (); //Violation is reported only for AB
because CD can be calculated from AB
endmodule
```

Example 3

```
module test2(input in, output out);
parameter AB=20; //No Violation
parameter CD=30; //No Violation
parameter TE=40; //Violation
parameter EF=AB/TE; //No Violation
parameter GH=EF; //No Violation
assign out = in;
endmodule
module test();
test2 #((5),(10)) T1 (); //Violation
```

```
endmodule
```

Example 4

```
//inc.v
module inc_mod (
input data,
input clk,
output reg out
);

parameter DATA_WIDTH = 8;
parameter ADDR_WIDTH = 8;
parameter RAM_DEPTH = 1 << ADDR_WIDTH;

assign out = data;

endmodule
//inc.v end

//test.v
`include "inc.v"

module test(input data, input clk, output reg out);

inc_mod #(
.ADDR_WIDTH(8)) mod(data, clk, out); //Violation only for
DATA_WIDTH
Endmodule
```

Example 5

```
//Parameter "report_all_params" should be enabled in order to
get the violation for EF.
module test2(input in, output out);
parameter AB=20;
parameter CD=20; //Violation reports only for CD
parameter EF=30;
assign out = in;
endmodule
```


MoreLint Tags

```
module test();  
test2 #(.AB(5)) T1 ();  
endmodule
```

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

NonSynthRepeat-ML

Repeat statements are non-synthesizable

Description

The *NonSynthRepeat-ML* Tag reports a violation when repeat statements are found, because such statements are not part of the Verilog synthesis level. They may not be supported by synthesis tools.

Language

Verilog

Message Details

The following message is reported when repeat statements are found:

[ERROR] Repeat statements [Condition: '<condition>'] are non-synthesizable

Example and /or Schematic

Consider the following code snippet:

```
module W66_mod2(i1, en, o1);
  input i1, en;
  output o1;
  reg o1;

  always@(en or i1)
    repeat(en)
      o1 = i1;
endmodule
```

For the above example, the Tag reports the following violation:

Repeat statements [Condition: 'en'] are non-synthesizable

Default Severity Label

Error

NonVoidFunction-ML

Checks if the function is explicitly casted to void

Description

The *NonVoidFunction-ML* Tag reports non-void functions that do not return anything in their body/description (implicitly used as void functions).

These functions should explicitly be casted to void.

If the return type is not specified, then the default return type is logic of width 1.

Language

Verilog

Parameters

None

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message is reported when a function (implicitly used as void functions) is not explicitly casted to void:

[WARNING] Function should explicitly be casted to void
[Hierarchy: '<hier-path>']

Potential Issues

None

Consequences of Not Fixing

Not fixing the violation may lead to simulation and synthesis mismatch.

How to Debug and Fix

To avoid this, an explicit return size or type needs to be specified. A good coding guideline would be to always specify the function return type, even when a void return is desired.

This problem can be fixed by:

- Explicitly casting the function to void return type.
- Not using the return value of function call.

Example Code and/or Schematic

Example 1

In the following example, the *NonVoidFunction-ML* Tag reports a violation because function f1 is not explicitly casted to void:

```
function f1 (input [7:0] d_in , output [7:0] d_out);  
    d_out = d_in + 1;  
endfunction
```

```
module test(  
    input [7:0] in,  
    output logic [7:0] out,  
    output logic out_return);  
  
    always @(*)  
    begin  
        out_return = f1(in,out);  
    end  
endmodule
```

Example 2

In the following example, the *NonVoidFunction-ML* Tag does not report violation because the function f2 is explicitly casted to void:

```
function void f2 (input [7:0] d_in , output [7:0] d_out);  
    d_out = d_in + 1;  
endfunction
```

Example 3

In the following example, the *NonVoidFunction-ML* Tag reports a violation for the function f3:

```
function logic [7:0] f3 (input [7:0] d_in , output [7:0]
```

MoreLint Tags

```
d_out);
    d_out = d_in + 1;
endfunction
```

Example 4

In the following example, the *NonVoidFunction-ML* Tag does not report violation because the function f4 has return statement:

```
function logic [7:0] f4 (input [7:0] d_in , output [7:0] d_out);
    d_out = d_in + 1;
    return 1;
endfunction
```

Example 5

In the following example, the *NonVoidFunction-ML* Tag does not report violation because the function f5 contains assignment to function name variable that acts as output:

```
function logic [7:0] f5 (input [7:0] d_in , output [7:0] d_out);
    d_out = d_in + 1;
    f5 = d_in;
endfunction
```

Example 6

In the following example, the *NonVoidFunction-ML* Tag does not report violation because the function contains assignment to function name variable that acts as output:

```
function [7:0] fCeilLog2;
input [255:0] n;
for (fCeilLog2=0; n>0; fCeilLog2=fCeilLog2+1) n = n>>1;
endfunction
```

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

OneLineComm-ML

Use one-line comments

Language

Verilog

Description

The OneLineComm-ML Tag flags multiline comments.

It is recommended to use one line comment (//) as it improves readability and code parsing. The use of multi-line (/ * . . */) comments should be avoided.

Message Details

The following message appears when multiline comment is encountered:
Use only one-line comments.

Severity

Tag

OperShortCircuit-ML

Checks if operator short circuiting occurs across the function call

Description

System Verilog functions are present across Logical OR (||) or Logical AND (&&) operators. If these functions take at least one argument as output or inout direction, then operator evaluation short circuiting may happen as output/inout argument may not always be evaluated in software simulation unlike hardware. In software simulation, the RHS evaluation across logical operator is dependent on return value of LHS while in hardware both RHS and LHS are evaluated for logical operations (|| or &&) that is not the same case with software simulation.

Outputs may be modified inside the function before function returns any value. It may lead to simulation synthesis mismatch as explained in the following example:

```
always @ (posedge clock)
if (f(i1,o1) && f(i2,o2)) // possible side effects
mem[addr] <= data_in;

function f (input [7:0] d_in , output [7:0] d_out);
d_out = d_in + 1;
    if (d_out == 255)
        return 0;
    else
        return 1;
endfunction
```

In simulation, binary operation might be exited while evaluating the expression from left to right.

In the above example, if `f(i1,o1)` returns 1, then `f(i2,o2)` might not get evaluated and thus `o2` would never be updated leading to simulation-synthesis mismatch.

All the binary sub-expressions having logical '&&' / logical '||' (only) will be checked.

Language

Verilog

MoreLint Tags

Parameters

None

Constraint(s)

None

Messages and Suggested Fix

The following message is reported when operator short circuiting can happen:

[WARNING] operator evaluation short circuiting may happen as function '<function-name>' may not be evaluated [Hierarchy: '<hier-path>']

Potential Issues

Not fixing the violation can lead to mismatch in simulation synthesis.

Consequences of Not Fixing

If you do not fix the violation, outputs may be modified inside the function before function returns any value. It may lead to simulation synthesis mismatch.

How to Debug and Fix

To avoid this, it is good to avoid operands with side effects. A side effect occurs when the operand modifies a value when the operand is evaluated. If the operands do not have side effects, then the behavior of short circuiting are the same and simulation will correctly match hardware behavior. To avoid the short circuiting for example above, it must be re-coded to ensure that both calls to the function always execute. For example:

```
always @(posedge clock)
begin
    temp1 = f (i1, o1); // o1 will be updated every time
    temp2 = f (i2, o2); // o2 will be updated every time
    if ( temp1 && temp2 ) // OK! no side effects
        mem[addr] <= data in;
end
```

Example Code and/or Schematic

Example 1

In the following example, the *OperShortCircuit-ML* Tag reports a violation for function f2 because out2 variable may not be evaluated that can lead to simulation synthesis mismatch:

```
function f1 (input d_in , output d_out);
    d_out = d_in + 1;
endfunction

function f2 (input d_in , output d_out);
    d_out = d_in + 1;
endfunction

module top(in1, in2, in3, in4, clk, out1, out2);
    input in1, in2, in3, in4, clk;
    output reg out1;
    output reg out2;

    integer b;

    always@(*)
    begin
        b = (f1(in2, out1) && f2(in1, out2));
    end
endmodule
```

Example 2

In the following example, the *OperShortCircuit-ML* Tag reports a violation for function f2:

```
assign a = f1(in1, out1) || f2(in2, out2);
```

Example 3

In the following example, the *OperShortCircuit-ML* Tag reports a violation for function f8 but as only one violation per logical expression is reported, thus no violation for f6 and f7 is reported:

MoreLint Tags

```
assign a = f5(in1, out1) && f6(in2, out1) && f7(in2, out1) &&
f8(in2, out1);
```

Example 4

In the following example, the *OperShortCircuit-ML* Tag does not report violation because function `f10` does not contain an argument having out/inout direction:

```
assign a = f9(in1, out1) && f10(in2, in3);
```

Example 5

In the following example, the *OperShortCircuit-ML* Tag does not report violation because expression does not contain `||` or `&&` operator:

```
assign a = in1 ? (f11(in2, out1) + f12(in1, out1)): f13(in2, out1);
```

Example 6

In the following example, the *OperShortCircuit-ML* Tag reports a violation for functions `f21` and `f23`:

```
assign a = (f20(in2, out1) && f21(in1, out1)) + (f22(in2,
out1) || f23(in1, out1));
```

Example 7

In the following example, the *OperShortCircuit-ML* Tag does not report violation because expression does not contain `||` or `&&` operator:

```
assign a = f24 (in1, out1) & f25 (in2, out1) | f26 (in2,
out1) & f27 (in2, out1);
```

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

ParameterCheck-ML

Do not use parameters

Description

The *ParameterCheck-ML* tag reports a violation when parameter and local-parameter is declared in the design.

This tag ensures that no parameter declaration and no parameter assignment exist in a Verilog netlist design. The tag reports all type of parameter descriptions made in the design.

Language

Verilog

Parameters

None

Messages and Suggested Fix

Message 1

The following message appears when a parameter declaration is found :

[ERROR] Parameter ('<parameter-name>') Declaration found

Message 2

The following message appears when a parameter is passed at declaration of instance

[ERROR] Parameter passed at declaration of instance
\"<instance-name>\" of module \"<module-name>\"

Examples

Example 1

Consider the following example:

```
module top(a,b);  
input a,b;  
parameter P = 1;
```

MoreLint Tags

```
wire [P:0] c;  
endmodule
```

For the above example, the tag reports the following violation message:
Parameter ('P') declaration found

Example 2

Consider the following example:

```
module top(a,b,c);  
input a,b;  
output c;  
MODA inst_moda #(2,3) (.a(A),.b(B),...);  
endmodule
```

For the above example, the tag reports the following violation message:
Parameter passed at declaration of instance "inst_moda" of
module "MODA"

Default Severity Label

Error

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

ParamValueOverride-ML

Detect overridden parameters in RTL design

When to Use

Use this Tag to detect overridden parameters in RTL design.

Description

This Tag reports information about overridden parameters in RTL design. This Tag generates the report ParamValueOverride_Report Report, which contains default and overridden values of the parameters of each elaborated module specified by [override_module_list](#).

Parameter(s)

[override_module_list](#): Default value is set to "". Set the value of the parameter to comma-separated list of regular expressions to match with the module names.

Constraint(s)

None

Messages and Suggested Fix

Message 1

Following message appears, when there are overridden parameters in elaborated modules are specified using the [override_module_list](#) parameter.

[WARNING] Some overridden parameters were detected. Please check the report 'ParamValueOverride_Report.rpt' for further details

Message 2

Following message appears, when there are no overridden parameters in elaborated modules specified using the [override_module_list](#) parameter:

[INFO] No overridden parameters were detected. The report 'ParamValueOverride_Report.rpt' is generated

MoreLint Tags

Example Code and/or Schematic

Not available

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

ParamValueOverride_Report Report

PortConnectToFixedVal-ML

Port of a module connected to a fixed value.

When to Use

Use this tag to detect port connects to fixed value.

Description

The *PortConnectToFixedVal-ML* tag reports a violation for a port, which:

- Belongs to a module that is specified using the `define_module_list` parameter.
- Is connected to fixed value a through a buffer of other combinational logic gates.
- Connects to a fixed value through a SpyGlass generated buffer.

Parameter(s)

- *define_module_list*: The default value is NULL. Use this parameter to specify the list of modules on which tag need to be applied.

Constraint(s)

None

Messages and Suggested Fix

[ERROR] <direction of port> pin <name of port> of Instance <instance of port> is tied-<supply mode> through supply net <Hierarchy of supply net>

Potential Issues

The violation message is reported when the port of a module is connected to a fixed value.

Consequences of Not Fixing

NA

How to Debug and Fix

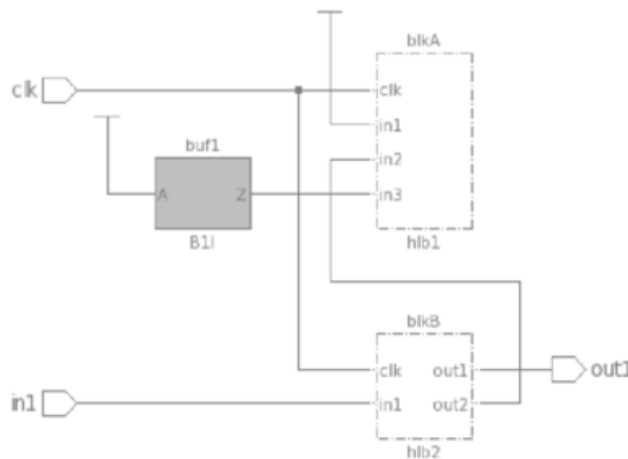
Click the violation message. This highlights the line in the HDL viewer where module is declared in the design.

Example Code and/or Schematic

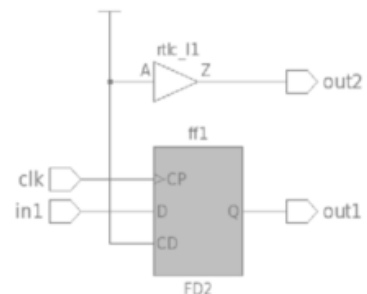
Example 1

Consider the following top module and sub-module:

Top Module



h1b2 Module



In the above example, the following ports of module h1b1 are reported:

- in1, since it is connected to high.
- in2, since it is connected to port out2 of module h1b2, which is connected to high directly.

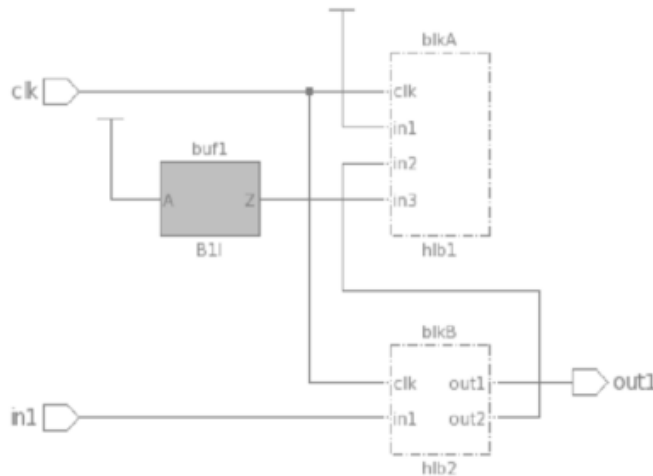
The out2 port of module h1b2 is reported as it is connected to a high.

Also, in3 of module h1b1 and out1 of module h1b2 are not reported as they are connected to a high through buffers.

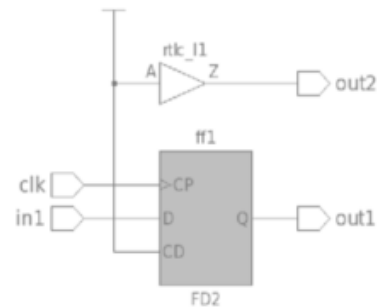
Example 2

Consider the following top module and sub-modules:

Top Module



h1b2 Module



The out ports of module, `b`, is reported as it is connected to a high directly.

The following ports of module, `test`, are reported:

- **a**: It is connected to out port of module `b`
- **reset**: It is connected to high
- **d**: It is an in-out port and connected to a low from outside and high from inside

Default Severity Label

Error

Tag Group

Not Applicable

Reports and Related Files

None

PortNameSameAsModule-ML

Do not use a port name identical to the unit name

Language

Verilog+VHDL

Message Details

The following message is displayed for the `<port-name>` port name, which is same as the unit name:

Do not use a port name '`<port-name>`' identical to the unit name

Suggested Fix

Modify one or other of the reported names to avoid duplication.

Example

VHDL

```
library ieee;
use ieee.std_logic_1164.all

entity test_nam046 is
  port (
    clk_ff  : in std_logic;
    rst_n   : in std_logic;
    test_nam046 : in std_logic_vector(3 downto 0); --bad
    y0_q    : out std_logic_vector(3 downto 0)
  );
end;
```

Tag Severity

Error

PortRange-ML

Top level output port is driven by a constant.

Language

Verilog, VHDL

Description

The PortRange-ML Tag flags top-level output ports that are driven by a constant value.

Message Details

The following message appears at the location where a top-level output port `<port-name>` is driven by a constant value:

Output port `<port-name>` is driven by constant value

Severity

Warning

PortUsedBeforeDefine-ML

Port used before defining

When to Use

Use this tag to detect ports that are used before defining.

Description

This tag reports violations for ports which are used before defining.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

Port '<port name>' is used before defining. [Hierarchy: '<hierarchy>']

Potential Issues

The violation message is reported when a port name is used before defining.

Consequences of Not Fixing

The violation reports bad coding style.

How to Debug and Fix

To fix the violation, ensure that the port name is used before defining.

Example Code and/or Schematic

```
module top ( clk, rst, in, out );  
  input clk;  
  input rst;  
  output reg out;  
  always @ (posedge clk ) begin
```

```
out <= rst ? 1'b0: in; //violation
end
input in;
endmodule
```

Default Severity Label

Warning

Reports and Related Files

None

PortWithoutDirection-ML

Port is defined without direction

When to Use

Use this tag to detect ports that are defined without a direction.

Description

This tag reports violation for ports that are defined without a direction. This tag enforces to specify the direction of a port, when it is defined.

The tag checks port definitions of all used modules and all used interfaces.

Parameter(s)

[*allow_indirect_directions*](#): The default value is no. Set the value of the parameter to yes to report violation for ports that are defined with indirect directions.

Constraint(s)

None

Messages and Suggested Fix

Port '<name>' of <module | interface> '<name>' is defined without a direction [Hierarchy : 'hierarchy']

Potential Issues

The violation message is reported when a port is defined without a direction.

Consequences of Not Fixing

This tag enforces the coding guidelines.

How to Debug and Fix

Clicking the violation message highlights the RTL line of where the port is defined.

Example Code and/or Schematic

Example 1

```
module test([0:1] a, // violation
            input b, // no-violation
            c, // violation by default. if
            'allow_indirect_directions=yes' then no-violation
            logic d, // violation
            output logic e); // no-violation
    assign e = d;
endmodule
```

Example 2

```
interface my_interface([0:1] x, // violation
                       input y, // no-violation
                       logic z); // violation
endinterface
module test(input data, output q); // no violation
    wire [0:1]a;
    wire b;
    my_interface i(a, data, b);
    assign q = data;
endmodule
```

Example 3

```
module test(a, b, c); // violation by default. if
    'allow_indirect_directions=yes' then no-violation
        input a;
        input b;
        output c;
        assign c = a | b;
    endmodule
```


MoreLint Tags

Example 4

```
interface my_interface();  
    logic x;  
endinterface  
module test (input in, // no-violation  
    my_interface data, // no-violation for interface var type ports/  
modports  
    output q); // no-violation  
    assign q = in;  
endmodule
```

Default Severity Label

Error

Tag Group

Not Applicable

Reports and Related Files

None

PortWithoutType-ML

Port is defined without type.

When to Use

Use this Tag to report violation for ports which are defined without a type.

Description

The PortWithoutType-ML Tag reports ports, which are defined without a type. This Tag enforces to specify the type of a port, when it is defined.

Tag checks port definitions for all the modules and all the used interfaces.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

[*allow_indirect_types*](#): Default value is no. Set the value of the parameter to yes to enable the tag to not report a violation for signals that have indirect declared type definition.

Constraint(s)

None

Messages and Suggested Fix

[ERROR] Port '<port-name>' is defined without a type [Hierarchy : '<hierarchy>']

Potential Issues

A violation message is reported when Port '<port-name>' is defined without a type [Hierarchy : '<hierarchy>']

Consequences of Not Fixing

This Tag enforces the coding guidelines.

How to Debug and Fix

To fix the violation, specify intended type of port signal, when it is defined.

Example Code and/or Schematic

Consider the following code snippet:

```
module test(output out1, input logic in1, input in2);  
    logic in2;  
endmodule
```

In the above example, out1 and in2 ports are defined without specifying a type. Therefore, Tag reports violation messages for the out1 and in2 ports.

Default Severity Label

Error

Tag Group

Not Applicable

Reports and Related Files

None

PortIOConnectionSanity-ML

Checks if primary input/output ports are connected to a valid input/output PAD cell pin

Description

This Tag reports primary input and output ports that are topologically connected to either multiple PAD cell pins or no PAD cell pin.

The Tags makes sure that:

- Primary input should be connected to only one input pad cell pin.
- Primary output should be connected to only one output pad cell pin.
- Primary inout ports are not checked by this Tag. Inout ports are checked by InoutPortConnectionSanity-ML Tag.

This Tag does a topological traversal to check PAD pin connectivity. During fan-in/fan-out traversal. Only simple buffers/inverters (SpyGlass generated) are skipped.

Language

Verilog

Parameters

None.

Constraint(s)

None

Messages and Suggested Fix

The following message appears when input/output ports are not connected to a valid input/output PAD cell pin:

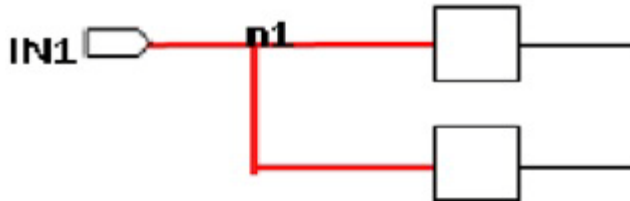
[ERROR] <port-type> port '<port-name>' should be connected to single <port-type> pad cell pin. But connections are:'<hier-pin-names>'

Where, <port-type> is input or output and <hier-pin-names> are the hierarchical pin names.

Example Code and/or Schematic

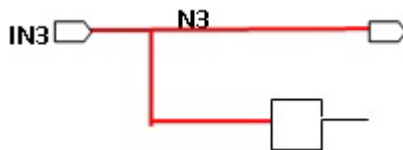
Example 1

In the following schematic, the Tag reports a violation because primary input port is connected to multiple PAD pins of different cells:



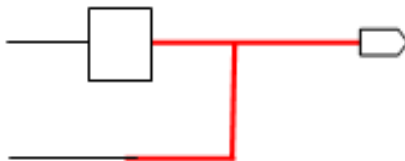
Example 2

In the following schematic, the Tag reports a violation because primary input port is connected to one pad cell pin and it is also connected to output port:



Example 3

In the following schematic, the Tag does not report any violation as primary output is connected to only pad output pin:



Default Severity Label

Error

Tag Group

None

Reports and Related Files

None

ProhibitedDataTypes-ML

Checks for prohibited data types being used directly

When to Use

Use this Tag to check whether any prohibited data types are being used directly.

Description

The ProhibitedDataTypes-ML Tag checks whether any prohibited data types are being used directly.

To identify the prohibited data types, use the `prohibited_data_types` parameter.

Parameters

[*prohibited_data_types*](#): Use this parameter to identify the prohibited data types and the corresponding allowed data types. You can specify a comma-separated list of data types as an input to this parameter.

Constraints

None

Messages and Suggested Fix

[WARNING] Type <prohibited data type> should not be used directly, use <allowed data type> instead. [Hierarchy: <hierarchy>]

Potential Issues

The violation message is reported when a prohibited data type is found in the design.

Consequences of Not Fixing

Not fixing the violation may result in unexpected results.

How to Debug and Fix

Click on violation message to display the RTL code where prohibited data type is declared.

Specify an allowed data type to fix the violation message.

Example Code and/or Schematic

Consider the following examples that explain the Tag behavior assuming the value of the `prohibited_data_types` parameter is set to `my_bit,bit,logic`.

Example 1

```
module top;

    typedef bit my_bit;    //assumed that "my_bit" is the global
                           type to be used

    typedef bit another_bit; //warning :- Type bit should
                           not be used directly, use my_bit instead

    bit a;    //warning :- Type bit should not be used
              directly, use my_bit instead

    logic b; //warning :- Type logic should not be used
              directly, use my_bit instead

endmodule
```

Example 2

```
module top;

    typedef bit another_bit; //warning :- Type bit should
                           not be used directly, use my_bit instead

    bit a;    //warning :- Type bit should not be used
              directly, use my_bit instead

    logic b; //warning :- Type logic should not be used
              directly, use my_bit instead

endmodule
```


Example 3

```

module top;

    typedef bit my_bit;                //Assume that "my_bit" is
    the global type to be used

    test my_test();

endmodule

module test;

    typedef bit another_bit;    //warning :- Type bit
    should not be used directly, use my_bit instead

    bit a;    //warning :- Type bit should not be used
    directly, use my_bit instead

    logic b;    //warning :- Type logic should not be used
    directly, use my_bit instead

endmodule

```

Example 4

```

module top;

    typedef bit my_bit;
    typedef struct{

        bit a;    //warning :- Type bit should not be
        used directly, use my_bit instead

        logic b;    //warning :- Type logic should not be
        used directly, use my_bit instead

    } my_struct;

```

```
struct {  
  
    bit c;        //warning :- Type bit should not be  
used directly, use my_bit instead  
  
    logic d;      //warning :- Type logic should not be  
used directly, use my_bit instead  
  
} my_str ;  
  
endmodule
```

Example 5

```
typedef struct{  
  
    bit a;        //warning :- Type bit should not be  
used directly, use my_bit instead  
  
    logic b;      //warning :- Type logic should not be  
used directly, use my_bit instead  
  
} my_struct;  
  
module top;  
  
    typedef bit my_bit;  
  
endmodule
```

Example 6

```
module top;  
  
    typedef bit [3:0]my_bit;  
  
    bit a;    //warning :- Type bit should not be used  
directly, use my_bit instead
```

```

    bit [3:0]b; //warning :- Type bit should not be used
    directly, use my_bit instead

    bit [4:0]c[3:0]; //warning :- Type bit should not be
    used directly, use my_bit instead

endmodule

```

Example 7

```

module top;

    typedef bit my_bit;

    typedef bit another_bit;      //warning :- Type bit
    should not be used directly, use my_bit instead

    if(0)
        bit a; //No warning

endmodule

```

NOTE: *SpyGlass does not report violation for dead codes.*

Example 8

```

interface intf_1;

    bit a;      //warning :- Type bit should not be used
    directly, use my_bit instead

endinterface

interface intf_2;

    bit a;      //warning :- Type bit should not be used

```

```

    directly, use my_bit instead

endinterface

module top;

    intf_1 intfInst();

endmodule

```

NOTE: *The Tag reports violation irrespective of the interface usage.*

Example 9

```

module top(input clk); // No warning, because default type of 'clk' is wire
    reg [3:0] a_reg;
    reg [3:0] c_reg;
    reg [1:0] val_reg;

    always @(posedge clk) begin
        c_reg = func_1( a_reg , val_reg);
    end

    function [3:0] func_1; // No warning, because default return type is reg

        input [3:0]a ;// No warning, because default type is reg

        input [1:0]val_1;    // No warning, because default type is reg

        bit [3:0] bb; //warning :- Type bit should not be used
        directly, use my_bit instead

        bb = a << val_1
        func_1 = bb;
    endfunction

endmodule

```

Example 10

```

module top(input clk);// No warning, because default type of 'clk' is wire

```

MoreLint Tags

```

reg [3:0] a_reg;
reg [3:0] c_reg;
reg [1:0] val_reg;

function [3:0] func_1; // No warning, because default return type is reg

    input [3:0]a ; // No warning, because default type is reg

    input [1:0]val_1; // No warning, because default type is reg

    bit [3:0] bb; //warning :- Type bit should not be used
    directly, use my_bit instead

    bb = a << val_1;
    func_1 = bb;
endfunction

                                endmodule

```

NOTE: Report violations for prohibited data type declarations, which are within tasks, functions, and interfaces, without considering their usage.

Example 11

Consider the following example and assume that the value of the `prohibited_data_types` parameter is set to `my_bit,bit,logic;my_logic,logic`.

```

module top();
typedef bit my_bit; // No warning

    typedef bit another_bit; //warning:- Type bit should not
    be used directly, use my_bit instead

    logic _logic; //warning:- Type logic should not be used
    directly, use my_bit instead
endmodule

```

NOTE: *If a prohibited data type is mentioned more than once in parameter as above, violation states only first allowed type.*

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

RealValueUsed-ML

Checks if real literals are used

Description

The *RealValueUsed-ML* Tag reports a violation if real literal is rounded to the nearest integer.

The real values are not allowed in synthesis models and hence rounded off to the nearest integer. This is an easy typographical error or IP backport issue. The real-valued object have no physical equivalent thus may not be synthesized by some tools. The real variables should only be used in test benches and simulation models.

The violation is reported:

- at the line where real keyword is found
- for real literals also

However, the tag does not report violation for real values in delay constructs.

Language

Verilog

Parameters

None

Messages and Suggested Fix

The following message appears when a real literal is rounded to the nearest integer:

[ERROR] Real variable/literal <variable-name> is not synthesizable [Hierarchy: <hier-path>]

Examples

Example 1

Consider the following example:

```
function [4:0] Adder;
input [3:0] first;
```

```
input [3:0] second;  
real a, b; // Violations  
begin  
    if(a > b)  
        Adder[3:0] = first + second;  
end  
endfunction
```

For this example, the Tag reports two violations as highlighted in the above example.

Example 2

Consider the following example:

```
module W294_mod(i1, i2, clk, res, o1);  
    input i1, i2;  
    input clk, res;  
    output o1;  
    reg o1;  
    real r1;  
    shortreal r3;  
    struct { real r2; } st1;  
    typedef struct { real r3; } sttype;  
    sttype st2;  
endmodule
```

Example 3

Consider the following example:

```
interface intf1;  
    reg w1;  
    integer w2;  
    byte w3;
```


MoreLint Tags

```
task task_local;
integer a;
real b;
endtask
//real w4;
Endinterface
```

For this example, the Tag reports a violation for the real used.

Example 4

Consider the following example:

```
input i1, i2;
input clk, res;
output o1;
reg o1; real r1;
assign o1 = r1 + 1.0;
endmodule
```

For this example, the Tag reports a violation for the real literal 1.0. The Tag also reports the violation for the declaration of r1.

Example 5

Consider the following example:

```
module W294_mod(i1, i2, clk, res, o1);
input i1, i2;
input clk, res;
output o1;
reg o1;
assign o1 = my_func1(2.0);
endmodule
```

For this example, the Tag reports a violation for the real literal 2.0.

Example 6

Consider the following example:

```
module top (input wire in, output wire out);  
    wire not_cnt,cnt;  
    reg cc;  
    assign # 1.5 out = in*1;//no violation  
    not # 2.5      n0(not_cnt, cnt);  
    always @ (*)  
    begin  
        #1.4 cc = #10.6 cnt*1;//no violation  
        #1.3 cc <= #10.5 cnt*1;//no violation  
    end  
endmodule
```

Default Severity Label

Error

RedundantLogicalOp-ML

Logical operation result is same as one of the operands or is a constant

Language

Verilog, VHDL

Description

The RedundantLogicalOp-ML Tag flags logical operations where one of the operands is a constant.

The RedundantLogicalOp-ML Tag also flags logical operations, where the result is same as one of the operands, or if both operands are the same.

For Verilog, the RedundantLogicalOp-ML Tag flags unary expressions and bit-wise operators only if the value of the [strict](#) Tag parameter is set to `yes` or `RedundantLogicalOp-ML`.

Logical operations with a constant operand or where both operands are same, are certainly a mistake, as such operations are redundant and add overhead for simulation and synthesis operations.

NOTE: *For Verilog, this Tag does not flag messages when an operand in the logical operation is a function with a constant output. The Tag also does not check if the function will always return a constant value.*

The RedundantLogicalOp-ML Tag checks only logical operations (all the operators that are logical operators) for this possibility.

For example:

Verilog

- `(a || 1)` always results in '1'.
- `(a || 0)` always results in 'a', so better write it as '`(a)`' only.
- `((a == b) && (a == b))` always results in '`(a == b)`'.

VHDL

- `(a or 1)` always results in '1'.
- `(a or 0)` always results in 'a', so better write it as '`(a)`' only.
- `((a and b) or (a and b))` always results in '`(a and b)`'.

To avoid excessive violations in VHDL designs, the RedundantLogicalOp-ML Tag reports only 20 messages per design unit and 500 messages per design. Set the [allviol](#) Tag parameter to *yes* or *<Tag-name>* to report all violation messages.

Parameters

[effort_dissolve_brackets](#): Default value is 500. Set the value of this parameter to any integer, as setting higher value for this parameter leads to more accurate results in the expressions involving small brackets and similar operators.

Message Details

Verilog

The following message appears at the location of an expression *<expr>* where a logical operation with a constant as one of the operands, is being performed:

Redundant logical operation '*<expr>*'

VHDL

The following message appears at the location of an expression *<expr>* where a logical operation with a constant as one of the operands, is being performed:

In '*<expr>*' logical operation is being performed on a constant

Severity

Warning

Examples (Verilog)

Example 1

Some of the examples where the RedundantLogicalOp-ML Tag is violated, are as follows:

```
o1[2:0] = i1[2:0] && 3'b101;
z2 = (a==b) && (a==b); -- always results in (a==b)
b = a || 1 -- always results in '1'
```

```
b = a || 0 -- always results in 'a'
```

Example 2

Consider the following example:

```
module top();
  wire a,b,out;
  assign out = (a||b) || (a||b); /* 2 Violations will be
  reported */
endmodule
```

For the above example, the following violations messages are reported:

Redundant logical operation (a||a)

Redundant logical operation (b||b)

Also, in the above example, no violation is reported for the (a||b) expression.

Examples (VHDL)

In the following example, the or logical operation with a constant (true) violates the RedundantLogicalOp-ML Tag:

```
library ieee;
use ieee.std_logic_1164.all

architecture rtl of test is
begin
  process(in1, in2, in3, in4)
  begin
    if((in1=in2) and (true or not(in3<=in4)))
      then out1 <= '1';
    else out1 <='0';
    end if;
  end process;
end rtl;
```

For this example, SpyGlass generates the following message:

In '(true or (not (in3<=in4)))' logical operation is being performed on a constant

RedundantLogicalOpInExpr-ML

Redundant expressions in logical expressions

When to Use

Use this tag to check if redundant expressions occur in logical expressions.

Description

The *RedundantLogicalOpInExpr-ML* tag reports violations when redundant expressions occur in logical expressions.

Language

Verilog+VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[warning] <expression> appears more than once in expression
<parent-expr> [hierarchy: <hierarchy>]

Potential Issues

The violation message is reported for ports where the redundant expressions occur in logical expressions.

Consequences of Not Fixing

Not fixing the violation may result in incorrect design modeling.

How to Debug and Fix

Click the violation message to point to the RTL statement where the duplicate expression is found.

Examples

Consider the following examples for Verilog:

Example 1

```
module test(a, b, c, d, out);
input [1:0] a,b,c,d;
output [1:0] out;
assign out[1] = a[1] & b[1] & c[1] & a[1]; //violation
assign out[0] = a[0] | b[0] | c[0] | d[0];
endmodule
```

Example 2

```
module test(a, b, c, d, out);
input [1:0] a,b,c,d;
output [1:0] out;
assign out[1] = (a[1] & ~b[1]) | (~a[1] & b[1]); // no violation
assign out[0] = a[0] | b[0] | c[0] | d[0];
endmodule
```

Example 3

```
module test(a, b, c, d, out);
input [1:0] a,b,c,d;
output [1:0] out;
assign out[1] = !(a[1] & b[1] & c[1] & a[1]); //violation
assign out[0] = a[0] | b[0] | c[0] | d[0];
endmodule
```

Example 4

```
module test(a, b, c, d, out);
input [1:0] a,b,c,d;
output [1:0] out;
assign out[1] = a[1] & b[1] & c[1] & a[0]; //no violation
assign out[0] = a[0] | b[0] | c[0] | d[0];
endmodule
```

Example 5

```
module test(a, b, c, d, out);
input [1:0] a,b,c,d;
output [1:0] out;
assign out[1] = 1? a[1] & b[1] & c[1] & a[1]:
a[0] | b[0] | c[0] | d[0]; //violation
endmodule
```

Example 6

```
module test(a, b, c, d, out);
input [1:0] a,b,c,d;
output [1:0] out;
assign out[1] = 0? a[1] & b[1] & c[1] & a[1]:
a[0] | b[0] | c[0] | d[0]; //violation
endmodule
```

Example 7

```
module test(a, b, c, d, out);
input [1:0] a,b,c,d;
output [1:0] out;
assign out[1] = condition ? a[1] & b[1] & c[1] & a[1]:
a[0] | b[0] | c[0] | d[0]; //violation
endmodule
```

Example 8

```
module test(a, b, c, d, out);
input [1:0] a,b,c,d;
output [1:0] out;
assign out[1] = condition ? a[1] & b[1] & c[1] & d[0]:
a[0] | b[0] | c[0] | d[0]; // no violation
endmodule
```


Example 9

```

module test(a, b, c, d, out);
input [1:0] a,b,c,d;
output [1:0] out;
assign out[1] = a[1] & b[1] & b[1] & a[1]; //violation
assign out[0] = a[0] | b[0] | c[0] | d[0];
endmodule

```

Example 10

```

module test(a, b, c, d, out);
input [1:0] a,b,c,d;
output [1:0] out;
always @(*) begin
if (a[1] & b[1] & b[1] & a[1]) //violation
assign out[1] = 1;
end
endmodule

```

Example 11

```

module test(a, b, c, d, out);
input [1:0] a,b,c,d;
output [1:0] out;
assign out[1] = a[1] & b[1] & (c[1] + a[1]); //no violation
assign out[0] = a[0] | b[0] | c[0] | d[0];
endmodule

```

Example 12

```

module test(a, b, c, d, out);
input [1:0] a,b,c,d;
output [1:0] out;
assign out[1] = a[1] & b[1] & (Cond ? a[1] : c[1]); //no violation
assign out[0] = a[0] | b[0] | c[0] | d[0];
endmodule

```

Example 13

```
module test(a, b, c, d, out);  
  input [1:0] a,b,c,d;  
  output [1:0] out;  
  assign out[1] = a[1] && b[1] && c[1] && a[1]; //violation  
  assign out[0] = a[0] | b[0] | c[0] | d[0];  
endmodule
```

Default Severity Label

Warning

RepeatedLogic-ML

Report cases when arithmetic or bitwise expressions inside 'conditions in loops' or 'condition bodies' can be placed outside

When to Use

Use this tag to detect arithmetic or bitwise expressions being used in the design.

Description

The *RepeatedLogic-ML* tag detects arithmetic or bitwise expressions being used in the design. The tag only considers the following types of expressions with the following operators.

- Arithmetic operators (+, -, *, /, %)
- Bitwise operators (&, |, ^, ~^)

This tag will only report violations for the following scenarios:

- Expressions found in generate loops
- Expressions found inside for loops

NOTE: *These should not be modified between loop iterations.*

- Expressions found in condition branches that is, inside:
 - ☐ if blocks
 - ☐ case blocks

Expression are to be reported for the following categories:

- ☐ Should be found in every branch
- ☐ No signals/variables at right hand side should have been modified within that branch

This tag helps the designer to get cleaner RTL by detecting such scenarios and notifying the user that they have possible repeated logics. Hence, the designer can extract those logics/expressions.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[Warning] Repeated logic <statement> can be placed outside

Potential Issues

The violation message is reported if repeated logic is found in the design that can be placed outside.

Consequences of Not Fixing

Not fixing the violation may result in incorrect design modeling.

How to Debug and Fix

Click the violation message to point to the RTL line where repeated logic is used.

Example Code and/or Schematic

Example 1

```
module top(a, b, c, in, out);  
parameter DEPTH = 2;  
parameter WIDTH = 2;  
parameter THRESHOLD = 2;  
input logic [5:0] a;  
input logic [5:0] b;  
input logic [5:0] c;  
input logic [31:0][31:0] in;  
output logic [31:0][31:0] out;  
genvar i,j;  
generate
```

MoreLint Tags

```

for (i=0; i<DEPTH/2; i++) begin
for (j=0; j<WIDTH/2; j++) begin
always_comb begin
if ((b+c) > THRESHOLD) // violation, b+c can be calculated
out of for loops
out[i][j] = in[i][j];
else
out[i][j] = 1'b0;
end
end
end
endgenerate
endmodule

```

Example 2

```

module top();
parameter NUM_INTR = 2;
parameter NOF_DIES = 2;
logic intr_section;
logic ext_immediate_intr[256:0];
logic [3:0] num_of_immediate_int;

always_comb begin
for (int i=0;i<128;i++) begin
if ((i+128*intr_section)<NUM_INTR*NOF_DIES) begin
// violation, 128*intr_section can be calculated out of
for loop
num_of_immediate_int = ext_immediate_intr[i+128*intr_section]
+ num_of_immediate_int;

```

```
end
end // for (i=0;i<128;i++)
end
endmodule
```

Example 3

```
module top (output reg a, input b, c);
wire condition;
always_comb begin
if (condition) begin
a = b+c; // violation, a=b+c appears in all branches and
// is not being affected by previous logic in
// branches, can be excluded from "if" construct
end
else begin
a = b+c;
end
end
endmodule
```

Example 4

```
module top (output reg a, input b, c);
wire condition [1:0];
always_comb
casez (condition)
2'd0 :
a = b + c; // violation
2'd1 :
```

MoreLint Tags

```

a = b + c;
2'd2 :
a = b + c;
default:
a = b + c;
endcase
endmodule

```

Example 5

```

module top (output reg a, input reg b, c);
wire condition;
always_comb begin
if (condition) begin
b = 1'b0;
a = b+c; // no violation
// a=b+c appears in all branches, but b is being affected
end
else begin
a = b+c;
end
end
endmodule

```

Example 6

```

module top (output reg a, input b, c);
wire condition;
always_comb begin
if (condition) begin
a = b+c; // no violation as a=b+c doesn't appear in all branches
end

```

```
else begin
a = b+a;
end
end
endmodule
```

Example 7

```
module top (output reg a, input b, c);
wire condition [1:0];
always_comb
casez ( condition )
2'd0 :
a = b + c; // violation even default is not similar (but
don't care space)
2'd1 :
a = b + c;
2'd2 :
a = b + c;
default:
a = 'x;
endcase
endmodule
```

Example 8

```
module top(a, b, c, in, out);
parameter DEPTH = 2;
parameter WIDTH = 2;
parameter THRESHOLD = 2;
input logic [5:0] a;
```


MoreLint Tags

```
input logic [5:0] b;
input logic [5:0] c;
input logic [31:0][31:0] in;
output logic [31:0][31:0] out;
genvar i,j;
generate
for (i=0; i<DEPTH/32; i++) begin
for (j=0; j<WIDTH/32; j++) begin
always_comb begin
out[i][j] = (b+c)&in[i][j]; // violation, b+c can be
calculated out of these loops
end
end
end
endgenerate
endmodule
```

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

ReportCompOperator-ML

Operator `<=` or `>=` used in binary expressions with `std_logic`, `std_ulogic`, `std_logic_vector` or `std_ulogic_vector` type operands

Description

The *ReportCompOperator-ML* tag reports violations for binary expressions whose operator is '`<=`' or '`>=`' and at least one operand is a signal of `STD_LOGIC`, `STD_ULOGIC`, `STD_LOGIC_VECTOR`, or `STD_ULOGIC_VECTOR`.

Rule Algorithm:

The *ReportCompOperator-ML* tag traverses for binary expressions on the given RTL design and reports violations if the binary expressions meet the following conditions.

Binary expression => (operand_1 operator operand_2)

- The operator must be either '`<=`' or '`>=`'.
- At least either one of the operand should be a signal of `STD_LOGIC` type.

Language

VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[Error] operator '`(<= | >=)`' is used in binary expression '`(expr)`'

Potential Issues

The violation message is reported if the `(<= | >=)` operator is used in the

binary expression.

Consequences of Not Fixing

Not fixing the violation may result in incorrect design behavior.

How to Debug and Fix

Click the violation message to point to the RTL line where there is binary expression.

Example Code and/or Schematic

Example 1

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
port (
  clk : in STD_LOGIC;
  reset : in STD_LOGIC;
  done : buffer BOOLEAN
);
end entity test;

architecture behv of test is
begin
  my_proc : process (reset, clk) is
  begin
    if (reset >= '1') then -- violation for (reset >= '1')
    done <= FALSE;
    elsif (clk'event and (clk <= '1')) then -- violation for
      (clk <= '1')
    done <= TRUE;
    end if;
```

```
end process my_proc;  
end architecture behv;
```

Example 2

```
library ieee;  
use ieee.std_logic_1164.all;  
entity test is  
port (  
  clk : in STD_LOGIC;  
  reset : in STD_LOGIC;  
  done : buffer BOOLEAN  
);  
end entity test;
```

```
architecture behv of test is  
begin  
  my_proc : process (reset, clk) is  
  begin  
    if (reset = '1' ) then -- no-violation, operator of the binary expr is neither  
      '<=' nor '>='  
    done <= FALSE;  
  elsif (rising_edge(clk)) then  
    done <= (clk <= '1'); -- violation for (clk <= '1')  
  end if;  
end process my_proc;  
end architecture behv;
```

Example 3

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
port (
  clk : in STD_LOGIC;
  reset : in STD_LOGIC;
  done : buffer BOOLEAN
);
end entity test;

architecture behv of test is
begin
  my_proc : process (clk) is
  begin
    if (rising_edge(clk)) then
      done <= (clk >= reset); -- violation for (clk >= reset)
    end if;
  end process my_proc;
end architecture behv;
```

Example 4

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
port (
  clk : in STD_LOGIC;
  reset : in STD_LOGIC;
```

```
sig : in STD_LOGIC;  
done : buffer BOOLEAN  
);  
end entity test;
```

```
architecture behv of test is  
begin  
my_proc : process (reset, clk) is  
begin  
if ((reset and sig) >= (clk or sig) ) then -- violation,  
both operands for '>=' operator are complex but operands  
sig, clk, reset are STD_LOGIC type signal.  
done <= FALSE;  
end if;  
end process my_proc;  
end architecture behv;
```

Example 5

```
library ieee;  
use ieee.std_logic_1164.all;  
entity test is  
port (  
clk : in STD_LOGIC;  
reset : in STD_LOGIC;  
sig : in STD_LOGIC;  
done : buffer BOOLEAN  
);  
end entity test;
```

MoreLint Tags

```

architecture behv of test is
begin
my_proc : process (reset, clk) is
begin
if (reset >= (clk or sig)) then -- violation for (reset >=
(clk or sig))
done <= FALSE;
end if;
end process my_proc;
end architecture behv;

```

Example 6

```

library ieee;
use ieee.std_logic_1164.all;
entity test is
port (
clk : in STD_LOGIC;
reset : in STD_LOGIC_VECTOR (4 downto 0) := "00000";
done : buffer BOOLEAN
);
end entity test;

```

```

architecture behv of test is
begin
my_proc : process (clk) is
begin
if (reset >= "0000") then -- violation, there is
'STD_LOGIC_VECTOR' signal
done <= FALSE;

```

```
end if;  
end process my_proc;  
end architecture behv;
```

Example 7

```
library ieee;  
use ieee.std_logic_1164.all;  
entity test is  
port (  
  clk : in STD_LOGIC_VECTOR (4 downto 0) := "00000";  
  reset : in STD_LOGIC;  
  done : buffer BOOLEAN  
);  
end entity test;  
  
architecture behv of test is  
begin  
  my_proc : process (reset, clk(0)) is  
  begin  
    if (reset = '1' ) then  
      done <= (clk(0) <= '1'); -- violation, clk is an  
      STD_LOGIC_VECTOR type signal  
    end if;  
  end process my_proc;  
end architecture behv;
```


MoreLint Tags

Default Severity Label

Error

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

RegisterInfo-ML

Generate a report which includes all registers in RTL design

When to Use

Use this tag to generate a report that includes all registers in the RTL design.

Description

The *RegisterInfo-ML* tag generates the [all_register_info Report](#). This report generates information, such as, total counts (Sync Total, Async Total, None Total), Register Name, Type, Reset Type, and flop/latch count for all the registers in the RTL design.

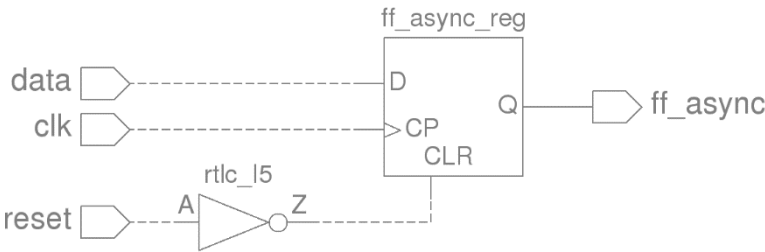
The tag checks the reset type of all bits in the register and reports multiple violations.

For register arrays, the tag reports only register name without bit information.

The following is the syntax of `all_register_info.rpt` report:

Register Name	Type	Reset Type
< hierarchical instance name>	<Flop/Latch>	<Async/Sync/None>

The tag detects asynchronous reset flops, if CLR/PRE pin is driven by valid signal.



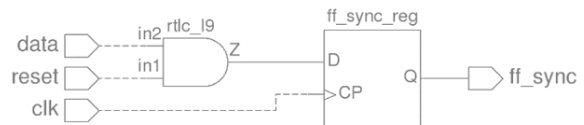
If CLR/PRE pin is not driven by valid signals, then the tag detects synchronous flops using RTL constructors.

MoreLint Tags

```

always @(posedge clk)
if(~reset) begin
    ff_sync <= 1'b0;
end else begin
    ff_sync <= data;
end

```



If the flop does not satisfy both async and sync reset checks, it is considered as non-reset flop.

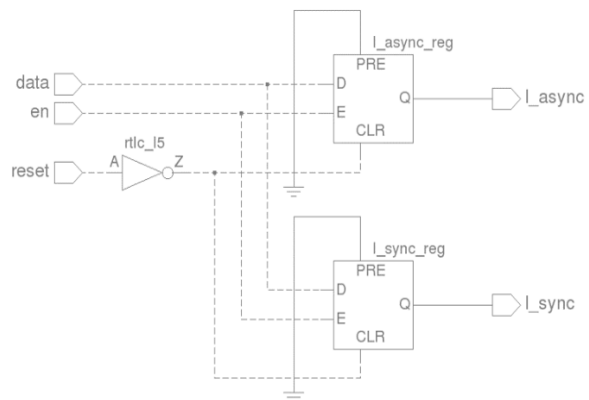
For latches, the tag detects asynchronous latches, when PRE/CLR pin is driven by valid signal.

```

always @(en or reset or data)
if (~reset) begin
    l_async <= 1'b0;
end else if (en) begin
    l_async <= data;
end

always @(en or data)
if (~reset) begin
    l_sync <= 1'b0;
end else if (en) begin
    l_sync <= data;
end

```



If PRE/CLR pins are not connected to valid signal, they are considered as non-reset latches.

Language

Verilog, VHDL

Parameter(s)

flag_reset_type: By default, the `flag_reset_type` parameter is set to "" and the tag does not report violations regarding report data. Set the parameter to `sync`/`async`/`none` to report violations if the combination has the following:

- `sync` - Reports registers on flops/latches with synchronous resets.
- `async` - reports registers on flops/latches with asynchronous resets.
- `none` - reports registers on flops/latches without synchronous or asynchronous resets.

Constraint(s)

None

Messages and Suggested Fix

Message1

[INFO] Please check the report 'all_register_info.rpt' for details on all registers in design.

Potential Issues

A violation is reported when there are register instances in the synthesized design.

Consequences of Not Fixing

Since the violation message is an informational message, it does not have a consequence.

How to Debug and Fix

No fix is required as this is an informational message.

Message 2

[warning] Design contains Async reset type registers. Please check the report 'all_register_info.rpt' for details on all registers in design

Potential Issues

A violation is reported when there are async type register instances in the design.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

Click the violation message to highlight the RTL line that has the async type register instances in the design.

Message 3:

[warning] 'Flop | Latch' type register '<name>' is encountered with reset type 'Sync | Async | None'

Potential Issues

A violation is reported when the flop/ latch comes across sync-reset/ async-reset/ none-reset type of registers in the design.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

Click the violation message and SpyGlass highlights the RTL line where the flop/ latch is defined.

Example Code and/or Schematic**Example 1**

Consider the following example:

```
module test (
  data    ,
  en      ,
  clk     ,
  reset   ,
  ff_async , ff_sync , ff_non ,
  l_async , l_sync , l_non
```

```
);  
input data, en, reset , clk ;  
output reg ff_async , ff_sync , ff_non;  
output reg l_async , l_sync , l_non ;  
always @( posedge clk or negedge reset )  
if(~reset) begin  
ff_async <= 1'b0;  
end else begin  
ff_async <= data;  
end  
always @( posedge clk )  
if(~reset) begin  
ff_sync <= 1'b0;  
end else begin  
ff_sync <= data;  
end  
  
always @( posedge clk )  
ff_non <= data;  
always @ ( en or reset or data)  
if (~reset) begin  
l_async <= 1'b0;  
end else if (en) begin  
l_async <= data;  
end  
  
always @ ( en or data)
```

MoreLint Tags

```

if (~reset) begin
l_sync <= 1'b0;
end else if (en) begin
l_sync <= data;
end
always @ ( en or data)
if (en) begin
l_non <= data;
end
endmodule

```

The all_register_info.rpt is as follows:

Register Name	Type	Reset Type
test/ff_async	Flop	Async
test/ff_sync	Flop	Sync
test/ff_non	Flop	None
test/l_async	Latch	Async
test/l_sync	Latch	Sync
test/l_non	Latch	None

Default Severity Label

Info

Tag Group

Not Applicable

Reports and Related Files

[all_register_info Report](#)

ResetPortConnectedToSupply-ML

Do not have reset connected to static value in instantiation

Tag description

The *ResetPortConnectedToSupply-ML* tag flags if resets are tied to logic 0, logic 1, VDD, and GND.

Resets should not be tied to logic 0, logic 1, VDD, and GND to avoid DFT issues.

NOTE: *The ResetPortConnectedToSupply-ML tag checks only those input ports of module which are connected to reset/set pin of any flip-flop (after ignoring buffers/inverters) in the design.*

Language

Verilog+VHDL

Message Details

The following message appears when the reset input *<port-name>* of module *<mod2-name>* is tied-high or tied-low:

In Module '*<mod1-name>*', reset input '*<port-name>*' of *<mod2-name>* [Instance *<inst-name>*] is 'tied-high | tied-low'

Suggested Fix

Review the reset input that is tied-high or tied-low to make sure that there is no typo.

Example

Consider the following example:

```
module testve_des015_des015_case1_1(  
    clk,  
    reset,  
    in1,
```


MoreLint Tags

```
    out1
  );

  input clk, reset, in1;
  output out1;

  wire x, y, z;
  assign x = 1'b0;
  assign y = ~x;
  assign z = x && in1;
  testve_des015_des015_case1_2 ff1(clk, 1'b0, x, y, z,
  out1);
endmodule

module testve_des015_des015_case1_2(
    clk,
    reset,
    in1,
    in2,
    in3,
    out1
);

  input clk;
  input reset;
  input in1;
  input in2;
  input in3;
  output out1;
  reg out1;

  always @(posedge clk or posedge reset)
  begin
    if(reset)begin
      out1 <= 0;
    end else begin
      out1 <= in1;
    end
  end
```

```
end  
endmodule
```

For the above code, the BB_DES015 tag reports the following violation message:

In Module 'testve_des015_des015_case1_1', reset input 'reset' of testve_des015_des015_case1_2 [Instance testve_des015_des015_case1_1@ff1] is 'tied-low'

Tag Severity

Error

ScopedVarUsedBeforeDefine-ML

Reports scoped variables that are used before defining

Description

The *ScopedVarUsedBeforeDefine-ML* Tag reports violations for scoped variables that are used before defining.

Language

Verilog

Messages and Suggested Fix

The following message is reported for the scoped variables used before defining:

[WARNING] Scoped variable <variable name> is used before defining. [Hier: <hierarchy>]

Example Code and/or Schematic:

Example 1:

Consider the following example where the `ms1.q1` variable is used before defining the `ms1` variable:

```
typedef struct {
    logic q1;
    logic q2;
} s1;

always @(posedge ck) begin
    ms1.q1 <= d1; // violation
end
s1 ms1;
```

Example 2:

Consider the following example where the `i.x` variable is used before defining the `i` variable:

```
module mod_a(input i1,input i2,input ck);
integer a;
```

```
always @(posedge ck) begin
a <= i.x; //violation
end
inf i();
endmodule

interface inf();
int x = 0;
endinterface
```

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

SepTFMacro-ML

Task/functions/macros should be defined in separate file

Language

Verilog

Description

The SepTFMacro-ML Tag flags source files that have a mix of task, function, macro, and other definitions.

The SepTFMacro-ML Tag requires that each task, function, or macro should be described in a separate source file. Also, these files should not have any other definitions.

Message Details

The following message appears for a source file that contains a mix of task, function, macro, and other definitions:

Define task/functions/macros in a separate file

Severity

Guideline

SimSynthMismatchInVarInit-ML

Simulation - synthesis mismatch in variable initialization

When to Use

Use this tag to check if there is a simulation - synthesis mismatch in variable initialization.

Description

The SimSynthMismatchInVarInit-ML tag reports a violation for simulation - synthesis mismatch when there is a variable initialization in an initial block or for `reg` and `logic` in the declaration.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Initialization of variable '<variable_name>' is ignored. This may cause simulation - synthesis mismatch
[Hierarchy: '<hierarchy>']

Potential Issues

The violation message is reported when there is a simulation - synthesis mismatch in variable initialization.

Consequences of Not Fixing

Not fixing the violation may result in simulation - synthesis mismatch in variable initialization.

How to Debug and Fix

Click the violation message to highlight the RTL line where the variable is assigned.

Example Code and/or Schematic

Example 1

Consider the following example:

```
Initial begin
c = 1'b0; // violation
end
```

Example 2

Consider the following example:

```
module test;
wire sig1=1'b0; // no violation
reg sig2=1'b1; // violation
endmodule
```

Example 3

Consider the following example:

```
module test;
reg c[0:3];
logic w[0:3];
integer i;
int j=0;
initial begin
for (i=0;i<=3;i=i+1) begin
c[i] = 1'b0; // violation
if(0) begin
for(j=0;j<=3;j++) begin
w[j] = 1'b0; // violation
end
end
end
```

```
end  
end  
endmodule
```

Example 4

Consider the following example:

```
module test;  
  wire sig1 = 1'b0; // no-violation  
  reg    sig2 = 1'b1; // violation  
  byte  sig_bit = 1'b1; // no-violation  
  logic sig3 = 1'b0; // violation  
  initial begin  
    sig_bit = 1'b0; // violation  
  end  
endmodule
```

Default Severity Label

Warning

Reports and Related Files

None

SigAsgnDelay-ML

Delays should not be used in Signal assignment

Language

VHDL

Description

The SigAsgnDelay-ML Tag flags delays used in signal assignment statements.

While delays are allowed in RTL description, they are ignored during synthesis. Hence, you should ensure that your design does not depend on delays to achieve correct functionality since this behavior will not necessarily map into post-synthesis behavior.

By default, the SigAsgnDelay-ML Tag flags delays used in sequential signal assignments only. Set the value of the [strict](#) Tag parameter to `yes` or `SigAsgnDelay-ML` to flag delays in concurrent assignments also.

Message Details

The following message appears at the location where a delay is specified in a signal assignment statement:

```
signal assignment uses delay
```

Severity

Warning

Examples

Some of the examples where the SigAsgnDelay-ML Tag flags a message are as follows:

```
case sel is
  when 1 => sig <= '0'; sig1 <= '0' after 12 ns;
  when others => sig<='X';
end case;
L1: for I in 1 to 10 generate
  signal sig3, sig4, sig5 : std_logic;
begin
```

```
sig3 <= sig4 after 1ns;  
sig4 <= '1' after 1ns, '0' after 2ns, 'X' after 3ns;  
sig5 <= '0';  
end generate L1;
```

SigAssignX-ML

Signal driven by X.

Language

Verilog, VHDL

Description

The SigAssignX-ML Tag flags signals driven by signals that are explicitly driven by X value.

NOTE: By default, the SigAssignX-ML Tag is switched off.

Message Details

The following message appears at the location where the signal *<sig-name>* is assigned X value:

Signal *<sig-name>* is assigned X value[Hierarchy: '*<hier-path>*']

Severity

Info

Examples (Verilog)

Some of the examples in Verilog where the SigAssignX-ML Tag is violated are as follows:

Example 1:

```
assign a = 1'bx;
assign b=a;           //violation for b
```

Example 2:

```
assign a = 1'bx;
assign a = 1'b0;
assign b=a           // violation for b as above two
                      statements are concurrent
```

Example 3:

```
assign b = a;         // violation for b
assign a = 1'bx;
```

```
assign a = 1'b0;
```

Example 4:

```
assign a = 1 ? 1'b0 : 1'bx;
assign b = a;           // violation for b
```

If any signal is assigned x value in an always block, no violation is reported as shown below:

```
always @(clk)
  a <= 1'bx;

assign b = a;           // no violation for b
```

The SigAssignX-ML Tag also supports module hierarchy but does not traverse inside instantiated module as shown below:

```
module HA (a,b,s,c);
  input a,b;
  output s,c;
endmodule

module top (x,y,s,c);
  inout x,y;
  output s,c;
  assign x = 1'bx;
  assign y = 1'bx;
  HA ha1(x,y,s,c);           // violation for a and b
                              // (input ports in module HA)

endmodule
```

Examples (VHDL)

Some of the examples in VHDL where the SigAssignX-ML Tag is violated are as follows:

Example 1:

```
library ieee;
```

MoreLint Tags

```
use ieee.std_logic_1164.all
```

```
i <= 'X';
j <= i;           --violation for j
```

Example 2:

```
k <= '0' when a = '0' else 'X' when a = '1';
j <= k           --violation for j as in a
                  particular condition k will
                  have 'X' value
```

Example 3:

```
signal i : std_logic_vector(0 to 2)
j <= i(2);       -- violation for j
i <= ('0','0','X');
```

Example 4:

```
i<= 'X';
i<= '0';
j<= i;           - violation for j as above two
                  statements are concurrent
```

Example 5:

If a signal is assigned X in process, no violation is reported if that signal is further assigned to any other signal as shown below:

```
process
begin
  i <= 'X';
end process

j<=i;           -- no violation for j
```

Example 6:

```
architecture abc of def is
  component INV
    port(yin:in std_ulogic;yout: out std_ulogic);
  end component;
  signal i : std_ulogic;
```

```
begin
  i <= 'X';
  inst1:INV port map(i,a);  -- violation for yin
                             input signal in
                             component as i
                             port maps to yin
end abc;
```

SigAssignZ-ML

Signal driven by Z.

Language

Verilog, VHDL

Description

The SigAssignZ-ML Tag flags signals driven by signals that are explicitly driven by Z value.

NOTE: By default, the SigAssignZ-ML Tag is switched off.

Message Details

The following message appears at the location where signal *<sig-name>* is assigned Z value:

signal <sig-name> is assigned Z value[Hierarchy: '<hier-path>']

Severity

Info

Examples (Verilog)

Some of the examples in Verilog where the SigAssignZ-ML Tag is violated are as follows:

Example 1:

```
assign a = 1'bz;
assign b=a;           //violation for b
```

Example 2:

```
assign a = 1'bz;
assign a = 1'b0;
assign b=a;           //violation for b as above two
                      statements are concurrent
```

Example 3:

```
assign b = a;         //violation for b
assign a = 1'bz;
```

```
assign a = 1'b0;
```

Example 4:

```
assign a = 1 ? 1'b0 : 1'bz;
assign b = a;           // violation for b
```

If any signal is assigned z value in always block, no violation is reported as shown below:

```
always @(clk)
  a <= 1'bz;

assign b = a;           // no violation for b
```

The SigAssignZ-ML Tag also supports module hierarchy but does not traverse inside instantiated module as shown below:

```
module HA (a,b,s,c);
  input a,b;
  output s,c;
endmodule

module top (x,y,s,c);
  inout x,y;
  output s,c;
  assign x = 1'bz;
  assign y = 1'bz;
  HA ha1(x,y,s,c);           // violation for a and b
                              // (input ports in module HA)

endmodule
```

Examples (VHDL)

Some of the examples in VHDL where the SigAssignZ-ML Tag is violated are as follows:

Example 1:

MoreLint Tags

```
i <= 'Z';
j <= i;           --violation for j
```

Example 2:

```
library ieee;
use ieee.std_logic_1164.all

k <= '0' when a = '0' else 'Z' when a = '1';
j <= k           --violation for j as in a
                  particular condition k will have
                  'Z' value
```

Example 3:

```
library ieee;
use ieee.std_logic_1164.all

signal i : std_logic_vector(0 to 2)
j <= i(2);       -- violation for j
i <= ('0','0','Z');
```

Example 4:

```
library ieee;
use ieee.std_logic_1164.all

i<= 'Z';
i<= '0';
j<= i;           -- violation for j as above two
                  statements are concurrent
```

Example 5:

If a signal is assigned Z in process, no violation is reported if that signal is further assigned to any other signal as shown below:

```
library ieee;
use ieee.std_logic_1164.all

process
begin
  i <= 'Z';
end process
```

```
j<=i;           -- no violation for j
```

Example 6:

```
library ieee;
use ieee.std_logic_1164.all

architecture abc of def is
  component INV
    port(yin:in std_ulogic;yout: out std_ulogic);
  end component;
  signal i : std_ulogic;
  begin
    i <= 'Z';
    inst1:INV port map(i,a);  -- violation for yin
                              input signal in
                              component as i
                              port maps to yin
  end abc;
```

SignalModifiedAfterRead-ML

Report, if signal is set after it is read in the always_comb block.

When to Use

Use this tag to detect a signal that is set after it is read in the `always_comb` block.

Description

The *SignalModifiedAfterRead-ML* tag reports violation when signal is set after it is read in the `always_comb` block. The tag checks both blocking and non-blocking assignment in the `always_comb` blocks.

When a signal is updated within the `always` block, it needs to update all other dependent variables according to the updated value.

For '`always @(*)`' blocks, the read and set order is not important, as it will trigger again based on sensitivity list. In the `always_comb` blocks, it gets evaluated once. Therefore, order is important.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

Message 1

Signal '<signal_name>' is set after it's read at line '<line number>' [Hierarchy: <hierarchy>]

Potential Issues

The violation message is reported when the signal is set after it is read

Consequences of Not Fixing

When a signal is set after it is read, simulation tools can have bad results.

How to Debug and Fix

Click the violation message in Console to highlight the following RTL lines:

- The line of where variable is set.
- The line of where variable is read.

Message 2

Signal '<signal_name>' is set after it's read at line '<line number>' in task/function call at line number '<line number>' [file:<file name>][Hierarchy: ': <hierarchy> ']

Potential Issues

The violation message is reported when the signal is set after it is read in a task/function call.

Consequences of Not Fixing

When a signal is set after it is read, simulation tools can have bad results.

How to Debug and Fix

Click the violation message in Console to highlight the following RTL lines:

- The line of where variable is set.
- The line of where variable is read.

Example Code and/or Schematic

Example 1

```
module test(in1 , in2 , read , out);
input in1 , in2 , read ;
output logic out;
logic cntr;
always_comb begin
    cntr = in1 ;
    out  = cntr & in2;
    if(read)
        cntr = in1 + 1; //violation
end
endmodule
```

Example 2

```
module test(in1 , in2 , read , out);
input in1 , in2 , read ;
output logic out;
logic cntr;
always_comb begin
    cntr = in1;
    if(read)
        cntr = in1 + 1; //no violation
    out  = cntr & in2;
end
endmodule
```

Example 3

```
module test(in1 , in2 , read , out);
input in1 , in2 , read ;
output logic out;
logic cntr;
always_comb begin
    out  = cntr & in2;
    if(read)
        cntr = in1 + 1; //violation
end
endmodule
```

Example 4

```
module test(in1 , in2 , read , out);
input in1 , in2 , read ;
output logic out;
logic cntr;
always @(*) begin
    cntr = in1 ;
    out  = cntr & in2;
    if(read)
        cntr = in1 + 1; //no violation
end
endmodule
```

Example 5

```
module test(in1 , in2 , read , out);
input in1 , in2 , read ;
output logic out;
logic cntr;
always_comb begin
    cntr = in1;
    if(read)
        out = cntr & in2;
    else
        cntr = in1 + 1; //no violation; read and set statements are in
        different branches.
    end
end
endmodule
```

Example 6

```
module test(in1 , in2 , read , out);
input in1 , in2 , read ;
output logic out;
logic cntr;
always_comb begin
    cntr = in1 ;
    out  = cntr & in2;
    if(read)
        cntr = in1 + 1; //violation; all set statements
        after read statement
    else
        cntr = in2 + 1; //violation; all set statements
        after read statement
end

endmodule
```

Example 7

```
module test(in1 , in2 , read , out);
input in1 , in2 , read ;
output logic out;
logic cntr;
always_comb begin
    cntr = in1 ;
    if(cntr)
        cntr = in1 + 1; //violation; rule considers
        conditional expressions as read statements.
```

```
end  
endmodule
```

Example 8

```
module test(in1 , in2 , read , out);  
input in1 , in2 , read ;  
output logic out;  
logic [3:0]cntr;  
always_comb begin  
    cntr = 4'b0000 ;  
    out  = cntr & in2;  
    if(read)  
        cntr[0] = in1 + 1; //violation  
end  
endmodule
```

Example 9

```
module test(in1 , in2 , read , out);  
input in1 , in2 , read ;  
output logic out;  
logic [3:0]cntr;  
always_comb begin  
    cntr = 4'b0000 ;  
  
    out  = cntr[0] & in2;  
    if(read)  
        cntr = in1 + 1; //violation  
end
```



```
endmodule
```

Example 10

```
module test(in1 , in2 , read , out);  
input in1 , in2 , read ;  
output logic out;  
logic [3:0]cntr;  
always_comb begin  
    cntr = 4'b0000 ;  
    out  = cntr[1] & in2;  
    if(read)  
        cntr[0] = in1 + 1; //no violation  
end  
endmodule
```

Example 11

```
module test(in1 , in2 , read , out);  
input in1 , in2 , read ;  
output logic out;  
typedef struct packed{  
    logic a;  
    logic b;  
}val;  
val cntr;  
always_comb begin  
    cntr = 2'b00 ;  
    out  = cntr.a & in2;  
    if(read)
```

```
        cntr.b = in1 + 1; //no violation
    else
        cntr.a = in1 + 1; //violation
    end
endmodule
```

Example 12

```
module test(in1 , in2 , read , out);
    input in1 , in2 , read ;
    output logic out;

    function eval;
        input in1 , in2 , read ;
        logic cntr;
        begin
            cntr = in1 ;
            eval  = cntr & in2;
            if(read)
                cntr = in1 + 1; //violation
            end
        endfunction

    always_comb begin
        out = eval(in1 , in2, read);
    end
endmodule
```

MoreLint Tags

Tag Severity

Warning

Reports and Related Files

None

SignedPackedMDA-ML

Reports if a signed packed multi-dimensional array is used

Language

Verilog

Description

The *SignedPackedMDA-ML* Tag reports violation if a signed packed multi-dimensional array is used.

NOTE: *The SignedPackedMDA-ML Tag ignores all unused arrays inside the design and reports only the used signed packed multidimensional arrays.*

Message Details

The following message appears when a signed packed multi-dimensional array is used:

Signed Packed Multi-Dimensional array '<array-name>' used in module '<module-name>'

Severity

Warning

Examples

Example 1

Consider the following example:

```
module top(in1,in2,in3,out1,out2,out3);  
input signed [1:0][1:0] in1; //Violation  
input logic in2 [1:0][1:0] ;  
input logic signed [1:0][2:0] in3 [1:0]; //Violation  
output out1 [1:0][1:0];  
output bit [1:0][1:0] out2;  
output bit signed [1:0][1:0] out3 [4:0][1:0]; //Violation
```

MoreLint Tags

```

logic w1[1:0][1:0];
logic signed [1:0][1:0]w2; //Violation
wire [1:0][1:0]w3;
wire w4[1:0][1:0];
wire signed [1:0][1:0]w5 [1:0][2:0]; //Violation
endmodule

```

Example 2

Consider the following example:

```

typedef struct {
byte a;
reg b;
shortint unsigned c;
bit signed [2:0][1:0] w2 [1:0]; //Violation
reg [2:0][1:0] w3 [1:0];
} myStruct;
module top(input in1);
myStruct in1 [1:0][2:0];
endmodule

```

Example 3

Consider the following example:

```

typedef struct packed {
byte a;
reg b;
shortint unsigned c;
bit signed [2:0][1:0] w2; //Violation
reg [2:0][1:0] w3;
} myStruct;

```

```
module top(input in1, in2);  
  myStruct in1 [1:0][2:0];  
  myStruct signed [1:0][2:0] in2 [1:0][2:0]; //Violation  
endmodule
```

Example 4

Consider the following example:

```
task my_task;  
  input logic w1[1:0][1:0];  
  output logic signed [1:0][1:0]w2[1:0][2:0]; //Violation  
  logic signed [1:0][3:0]w3[1:0]; //Violation  
begin  
end  
endtask
```

Example 5

Consider the following example:

```
interface mem_if (input wire clk);  
  wire reset;  
  wire we;  
  wire ce;  
  wire [7:0] datai;  
  logic [7:0] datao;  
  wire [7:0] addr;  
  logic signed [1:0][2:0] multiWire1; //Violation  
  wire [1:0][2:0] multiWire2;  
  bit signed [2:0][1:0] multiWire3 [1:0]; //Violation  
endinterface
```

Example 6

Consider the following example:

```
function signed [1:0][1:0] my_func ; //Violation
input logic [1:0][1:0]w1;
logic signed [3:0][1:0]w3[1:0]; //Violation
begin
end
endfunction
```

Example 7

Consider the following example:

```
package myPack;
logic w1[1:0][1:0];
logic signed [1:0][2:0] w2[1:0][1:0]; //Violation
endpackage
```

Example 8

Consider the following example:

```
always@(*)
begin
logic signed [1:0][1:0] logic_wire1; //Violation
logic logic_wire2[1:0][1:0];
w2 <= w1;
end
```

Example 9

Consider the following example:

```
generate
for (i=0; i < 4; i=i+1) begin : MEM
logic signed [1:0][1:0] logic_wire1; //Violation
wire logic_wire2[1:0][1:0]; //No Violation
```

```
...  
end  
endgenerate
```

Example 10

Consider the following example:

```
typedef logic signed [1:0] foo_t;  
foo_t [1:0][7:0] foo4; //No Violation
```

Example 11

Consider the following example:

```
parameter logic signed [3:0][1:0]P = 8'b0; //Violation  
localparam logic signed [3:0][1:0]P1 = 8'b0; //Violation  
parameter logic [3:0][1:0]P2 = 8'b0; //No Violation
```

Example 12

Consider the following example:

```
typedef enum bit signed [3:0] {m1, m2, m3} enum1_t;  
enum1_t [3:0][1:0]enum_array1; //No Violation  
enum1_t [3:0]enum_array1; //No Violation
```

Example 12

Consider the following example:

```
input signed [7:0] in1[1:0][1:0]; //No Violation
```


SimSynthMismatchInVarInit-ML

Simulation - synthesis mismatch in variable initialization

When to Use

Use this tag to check if there is a simulation - synthesis mismatch in the variable initialization.

Description

The *SimSynthMismatchInVarInit-ML* tag reports a violation for simulation - synthesis mismatch when there is a variable initialization in an initial block or for reg and logic in the declaration.

Language

Verilog + SystemVerilog

Parameter(s)

- *check_var_decl_init*: By default, the value of the parameter is set to `no`. Set the value of the parameter to `yes` to enable the tag to perform the following:
 - ☐ Avoid checking for initialization inside initial block of any data type such as bit, byte, int, reg and logic data type.
 - ☐ Report a violation for initialization of reg, logic, bit, byte, short real, real, int, and other data type except the wire data types declared in the module scope.
 - ☐ Report a violation in any named block where a reg, logic, bit, byte, integer, and int signal is initialized except the wire data type.
 - ☐ Report a violation if there is any initialization done on all kind of data type such as reg, logic, integer, int or real data type inside generate block. The tag checks all types of generate blocks as follows:
 - ◆ generate for
 - ◆ generate case
 - ◆ generate if

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Initialization of variable '<var_name>' inside [<generate-for/generate-if/generate-case/module>] is ignored. This may cause simulation - synthesis mismatch
[Hierarchy:'Hier_path'];"

Potential Issues

The violation message is reported when the initialization of the variable inside the generate-for/ generate-if/ generate-case/ module is ignored leading to a simulation-synthesis mismatch.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

Click the violation message to highlight the RTL line where the particular initialization of integer variable has occurred.

Example Code and/or Schematic

Consider the following examples when the check_var_decl_init parameter is set to yes:

Example 1

```
genvar i;
generate
for (i=0; i<=K*2; i=i+1) begin
integer base_l0 = R_MAX + (i>>1); // violation
real base_r0 = R_MAX + K - (i>>1); // violation
end
endgenerate
```

Example 2

```
generate
if (USE_CASE)
integer base_l0 = R_MAX + 1; // violation
```

MoreLint Tags

```

else
real base_r0 = R_MAX + 2; // violation
endgenerate

```

Example 3

```

generate
case(ADDER_TYPE)
0 : integer base_l0 = R_MAX + 1; // violation
1 : shortreal base_r0 = R_MAX + 2; // violation
endcase
endgenerate

```

Example 4

```

generate
case(ADDER_TYPE)
0 : reg base_l0 = R_MAX + 1; // violation
1 : real base_r0 = R_MAX + 2; // violation
endcase
endgenerate

```

Example 5

```

module test;
wire sig1=1'b0; // no violation
reg sig2=1'b1; // violation
logic sig3=1'b1; // violation
byte c = "S"; // violation
bit [10:0] a = "\x41"; // violation
bit [1:4][7:0] h = "hello" ; // violation
endmodule

```

Example 6

```
module test;
wire sel, in1;
wire sig1=1'b0; // no violation
reg out;
always @ (sel or in1)
begin
integer base_l0 = R_MAX + (i>>1); // violation
real base_r0 = R_MAX + K - (i>>1); // violation
out = sel ? in1 : base_l0 & base_r0;
end

endmodule
```

Example 7

```
for (genvar i=0; i<=K*2; i=i+1) begin
integer base_l0 = R_MAX + (i>>1); // violation
integer base_r0 = R_MAX + K - (i>>1); // violation
```

Example 8

```
genvar i;
generate
for (i=0; i<=K*2; i=i+1) begin
always @ (sel or in1)
begin
integer base_l0 = R_MAX + (i>>1); // violation
real base_r0 = R_MAX + K - (i>>1); // violation
out = sel ? in1 : base_l0 & base_r0;
end
```

```
end
endgenerate
```

Example 9

```
module test;
wire sig1=1'b0; // no violation
reg sig2=1'b1; // violation
logic sig3=1'b1; // violation
generate
for (genvar i=0; i<=K*2; i=i+1) begin
integer base_l0 = R_MAX + (i>>1); // violation
integer base_r0 = R_MAX + K - (i>>1); // violation
end
endgenerate
```

```
endmodule
```

Consider the following examples when the `check_var_decl_init` parameter is set to no:

Example 10

```
module test;
wire sig1=1'b0; // no violation
reg sig2=1'b1; // violation
logic sig3=1'b1; // violation
generate
for (genvar i=0; i<=K*2; i=i+1) begin
integer base_l0 = R_MAX + (i>>1); // no violation
integer base_r0 = R_MAX + K - (i>>1); // no violation
end
endgenerate
```

```
endmodule
```

Example 11

```
module test;  
  wire sel, in1;  
  wire sig1=1'b0; // no violation  
  reg out;  
  always @ (sel or in1)  
  begin  
    integer base_l0 = R_MAX + (i>>1); // no violation  
    real base_r0 = R_MAX + K - (i>>1); // no violation  
    out = sel ? in1 : base_l0 & base_r0;  
  end  
end
```

```
endmodule
```

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

SingleEntInFile-ML

There should be only one entity or one entity and its architecture in a file

Language

VHDL

Description

The SingleEntInFile-ML Tag flags the following cases:

- More than one entity is described in a source file.
- An entity and a package are described in a source file.
- An entity and a package body are described in a source file.
- An entity and a configuration are described in a source file.
- An entity and an architecture of a different entity are described in a source file.
- Architectures of different entities are described in a source file.

The SingleEntInFile-ML Tag does not flag when an entity and its architecture are described in the same file.

Message Details

The following message appears when two entities `<entity1-name>` and `<entity2-name>` are described in the same source file `<file-name>`:

Entity '`<entity1-name>`' and Entity '`<entity2-name>`' present in the same file '`<file-name>`'

The following message appears when entity `<entity1-name>` and architecture `<arch-name>` of entity `<entity2-name>` are described in the same source file `<file-name>`:

Architecture '`<entity2-name>.<arch-name>`' and Entity '`<entity1-name>`' present in the same file '`<file-name>`'

The following message appears when architecture `<arch1-name>` of entity `<entity1-name>` and architecture `<arch2-name>` of entity `<entity2-name>` are described in the same source file `<file-name>`:

Architecture '<entity2-name>.<arch2-name>' and Architecture '<entity1-name>.<arch1-name>' present in the same file '<file-name>'

The following message appears when entity *<entity1-name>* and package *<pkg-name>* are described in the same source file *<file-name>*:

Package '<pkg-name>' and Entity '<entity1-name>' present in the same file '<file-name>'

The following message appears when entity *<entity1-name>* and package body *<pkg-name>* are described in the same source file *<file-name>*:

Package body '<pkg-name>' and Entity '<entity1-name>' present in the same file '<file-name>'

The following message appears when entity *<entity1-name>* and configuration *<config-name>* of entity *<entity2-name>* are described in the same source file *<file-name>*:

Architecture '<entity2-name>.<arch-name>' and Entity '<entity1-name>' present in the same file '<file-name>'

Severity

Warning

Suggested Fix

Make sure only an entity or an entity and its architecture is present in a file.

SpareCaseExprBit-ML

Detect redundant bits in case expression

Language

Verilog

Tag Description

The *SpareCaseExprBit-ML* tag reports a violation when the number of bits in a case expression does not match with the number of case items.

$n < 2^{(N-1)} + 1$

N – number of bits in case expression

n – number of case items (ignore default case) .

Parameter(s)

ignore_one_hot: By default, the *ignore_one_hot* parameter is set to no. Set the parameter to yes to enable the tag to not report violations for case blocks with one-hot case items.

Constraints

None

Message Details

Message

[warning]Case expression '<expression>' of width '<bit-number>' is not matched with item count '<count>' [Hierarchy: <hier-name>]

Potential Issues

The violation message is reported when the number of bits in a case expression does not match with the number of case items.

Consequences of Not Fixing

Not fixing the violation may result add unnecessary resources in the synthesized netlist.

How to Debug and Fix

To debug the violation, click the violation message. VC SpyGlass highlights the RTL line with the case block.

Examples

Example 1

Consider the following example:

```
module test(i1, i2, i3, i4, i5, i6, i7, i8, sel, outt);
input  [3:0] i1, i2, i3, i4, i5, i6, i7, i8;
input  [3:0] sel;
output [3:0] outt;
reg [3:0] outt;
always @(*) begin
    case(sel[3:0]) // violation - Redundant case_expression
        4'd1 : outt = i1;
        4'd2 : outt = i2;
        4'd3 : outt = i3;
        4'd4 : outt = i4;
        4'd5 : outt = i5;
        4'd6 : outt = i6;
        4'd7 : outt = i7;
        4'd8 : outt = i8;
        default : outt = 4'h0;
    endcase
end
endmodule
```

Example 2

Consider the following example:

```
module test(i1, i2, i3, i4, i5, i6, i7, i8, sel, outt);
```

MoreLint Tags

```
input  [3:0] i1, i2, i3, i4, i5, i6, i7, i8;
input  [3:0] sel;
output [3:0] outt;
reg [3:0] outt1;
always @(*) begin
  case(sel[2:0]) // No violation 3-bit case_expression is sufficient
    3'd0 : outt1 = i1;
    3'd1 : outt1 = i2;
    3'd2 : outt1 = i3;
    3'd3 : outt1 = i4;
    3'd4 : outt1 = i5;
    3'd5 : outt1 = i6;
    3'd6 : outt1 = i7;
    3'd7 : outt1 = i8;
  endcase
end
assign outt = sel[3] ? 4'h0 : outt1;
endmodule
```

Default Severity Label

Warning

Tag Group

morelint

Reports and Related Files

No related reports or files.

SpecifyBlockUsed-ML

Specify blocks are not supported

When to Use

Use this tag to detect specify blocks used in the design.

Description

Specify blocks are used for describing paths between a source and a destination and for assigning delays to these paths. The Specify blocks are not supported by synthesis.

Parameter(s)

None

Messages

SPECIFY construct is not supported

Example Code and/or Schematic

```
module DCVER_276(o1,i1,i2,opcode);  
input [7:0] i1,i2;  
input [2:1] opcode;  
output [7:0] o1;  
specify //Not supported by synthesis  
    if (opcode == 2'b00) (i1,i2 *> o1) = (25.0,25.0);//  
violation  
    if (opcode == 2'b01) (i1=>o1) = (5.6,8.0);  
endspecify  
endmodule
```

Reports and Related Files

None

Default Severity Label

Warning

SyncRstFirstUse-ML

Synchronous reset should be used in first if condition of always block

Description

The *SyncRstFirstUse-ML* Tag reports violation if the reset signal is not used in the first `if` statement in the `always` block.

Set of signals to be treated as reset signals should be provided as a comma-separated list of signal names using the [user_defined_resets](#) parameter.

Language

Verilog

Messages and Suggested Fix

The following message appears if the reset signal is not used in the first `if` statement in the `always` block:

[WARNING] Sync reset '<sync-reset pattern>' not used in first if condition of sequential element '<seq-element-name>'

Example Code and/or Schematic

Example 1

Consider the following example where the [user_defined_resets](#) parameter is set to 'rst_n':

```
always_ff @(posedge clk)
begin
    if (blah) foo <= 1'b0; //Violation
    else if (rst_n) foo <= 1'b0;
    else foo <= in;
end
```

For the above example, the Tag reports a violation because synchronous

reset 'rst_n' is not used in first `if` condition.

Example 2

Consider the following example where the [user_defined_resets](#) parameter is set to 'rst_n':

```
always_ff @(posedge clk)
begin
    if (!rst_n) //No violation
        foo <= 1'b0;
    else if (blah) foo <= 1'b0;
    else
        foo <= in;
end
```

For the above example, the Tag does not report violation because synchronous reset is used in first `if` condition.

Default Severity Label

Warning

Reports and Related Files

None

SynchValueUsed-ML

Synchronized value should not be used asynchronously

Language

VHDL

Description

The SynchValueUsed-ML Tag flags expressions where synchronized signals are being read in an asynchronous manner.

Such constructs may result in simulation errors and also may not be synthesizable.

Message Details

The following message appears at the location of an expression where a synchronized signal *<name>* is being read in an asynchronous manner:

synchronized value *<name>* should not be used

Severity

Warning

Examples

Consider the following example where TMP signal is first synchronized to the positive edge of CLK clock signal and is then read asynchronously:

```
process ( CLK )
  variable TMP : std_logic;
  begin
    if ( CLK'event and clk='1') then
      TMP := D;
    end if;

    Q <= TMP;
  end process;
```

For this example, SpyGlass generates the following message:
synchronized value TMP should not be used

SynthElabDuName-ML

Synthesized module name is longer than required limit.

Language

Verilog, VHDL

Description

The SynthElabDuName-ML Tag flags if the length of a synthesized module name exceeds the specified limit.

By default, the maximum length allowed for the name of a synthesized module is 40. Use the [synth_name_length](#) Tag parameter to change the default value.

Message Details

The following message appears at the location of a module whose length *<mod-length>* exceeds the limit *<mod-limit>* specified by the `synth_name_length` Tag parameter:

Module name length *<mod-length>* exceeds the limit *<mod-limit>* in the instance [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the instance hierarchy of the Tag violating module.

Severity

Warning

SVConstruct-ML

SystemVerilog construct used

Language

Verilog

Description

The SVConstruct-ML Tag flags use of SystemVerilog constructs.

Parameter(s)

REPORT_ONLY_SVCONSTRUCT: By default, the value of the parameter is null. Set the value of the parameter to comma-separated system Verilog constructs for which you want to report violation.

Message Details

The following message appears at the location where SystemVerilog construct `<sv-constr>` is used:

SystemVerilog construct (`<sv-constr>`) found

Potential Issues

The violation message is reported at the location where SystemVerilog construct is used. This is an informational message.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix information is required.

Severity

Info

TwoStateData-ML

Avoid using two state data type signals in RTL

Language

SystemVerilog

Description

The TwoStateData-ML Tag reports a violation, if two state data type is used in the design. Following are the two-state data types in SystemVerilog, for which Tag will report violation:

`bit, byte, shortint, int, longint`

The Tag does not flag violation for `int` data type, if it is used as a for loop index.

The Tag reports a violation at the line of declaration of the two-state signal.

The Tag does not report violation for following four-state data types:

`logic/reg, integer, wire, wand, wor`

The TwoStateData-ML Tag considers the following cases for declaration of two state data type:

- Globally or in a module
- In a structure or a union definition
- In a function or a task
- In a package
- In a for-generate construct
- In interfaces
- Using a typedef

Also, the Tag checks for ENUM types defined using two state data types. If a type is not specified in the ENUM declaration, the Tag considers it as type `int` according to LRM.

Parameter(s)

- [ignore_parameter_type](#): Default value is no. Set the value of the parameter to yes to avoid reporting violations on two state data types used as parameter types.
- [check_enum_usage](#): By default, the parameter is set to no. Set the parameter to yes to enable the tag to report a violation for 2-state enums where they are used as datatypes. The tag does not report a violation on enum datatype usages for parameters or localparameters. The `check_enum_usage` parameter is ineffective if the `check_four_state_assign` parameter is set to yes.

Message Details

The following violation message is displayed for the TwoStateData-ML Tag:
Two state data type "<data-type>" declaration found

Potential Issues

The violation message is reported when the two state data type declaration is found in the design.

Consequences of Not Fixing

Not Fixing the violation can result in mismatch in simulation and synthesis.

Tag Severity

Warning

Examples

Example 1

Consider the following example:

```
int g_sig;
module test(input longint sig0);
    bit [5:3][3:5] sig1;
    byte sig2, sig3;
    int sig4 [3:0];
    shortint sig5;
    int sig6;
endmodule
```

The above example shows declaration of two state data type globally or in a module. The Tag reports violation for all declarations, except sig6, which will violate only when it is not a for loop index.

Example 2

Consider the following example:

```
typedef struct {  
    byte e11;  
    int e12;  
    bit [3:0] e13;  
} str_type;
```

The above example shows declaration of two state data type in a structure/union definition. The Tag reports violation for all the declarations.

Example 3

Consider the following example:

```
typedef bit b_type;  
module test();  
    b_type sig1;  
    b_type sig2, sig3;  
endmodule
```

The above example shows a declaration of two state data type using a typedef. In the above example, the Tag reports violation on typedef declaration.

TypedefNameConflict-ML

Ensures that variable name are not same as that of the System Verilog type-defined name

When to Use

Whenever there is a System Verilog design using Typedef constructs.

Description

This Tag reports a violation for the signals whose name is same as that of any type-defined name in the System Verilog design. No violation is reported for two signals having same name.

Default Weight

10

Language

Verilog, System Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is reported for the signals `<signal-name>` whose name in the module in module `<hier-module-name>` is the same as that of any type-defined name at the `<type-def-line>` line:

[FATAL] Signal name `<signal-name>` collides with the name of type defined in module `<hier-module-name>` at line no. `<type-def-line>`

Potential Issues

None

Consequences of Not Fixing

There is a possibility of ambiguity.

How to Debug and Fix

Double click the violation message. The HDL viewer window highlights the line where violation is detected. Fix the violation by appropriately changing the variable or type-defined name.

Example Code and/or Schematic

Consider the following example:

```
module top();  
    typedef int tx;    //Type name is tx  
    tx x;  
    logic y;    // No violation  
    logic tx;    //TypedefNameConflict-ML violation (signal  
                name is also tx)  
endmodule
```

In the above example, the *TypedefNameConflict-ML* Tag reports violation as the signal name tx is same as type-defined name tx.

Default Severity Label

Fatal

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

UncoveredEnumCase-ML

All enum items should be defined in case statement

When to Use

Use this tag to check if all enum items are defined in a case statement.

Description

The *UncoveredEnumCase-ML* tag reports a violation if the selector of a case statement is an enum type and every declared enum item is not defined within the case as a case item.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Enum item(s) '<item_list>' not covered in case statement under selector '<name>' of type '<type>' [Hierarchy: 'hierarchy']

Potential Issues

A violation is reported when the declared enum item is not defined within the case as a case item.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

Click the violation message, SpyGlass points to the RTL line where the case statement is defined.

Example Code and/or Schematic

Example 1

Consider the following example:

```
module test(input in, output reg out);
typedef enum {A, B, C, D} SELECT; // A=0 , B=1 , C=2 , D=3
SELECT sel;

always@(posedge in) begin /*sequential blocks*/
case (sel) // violation - enum name 'D' is not defined
A: out = 2'b00;
B: out = 2'b01;
C: out = 2'b10;
default: out = 2'b00;
endcase
end
endmodule
```

Example 2

Consider the following example:

```
module test(input in, output reg out);

typedef enum {A, B, C, D} SELECT; // A=0 , B=1 , C=2 , D=3
SELECT sel;

always_comb begin /*combinational blocks*/
case (sel) // violation - enum name 'D' is not defined
A: out = 2'b00;
B: out = 2'b01;
```


MoreLint Tags

```
C: out = 2'b10;  
default: out = 2'b00;  
endcase  
end  
endmodule
```

Example 3

Consider the following example:

```
module test(input in, output reg out);  
  
typedef enum {A, B, C, D} SELECT; // A=0 , B=1 , C=2 , D=3  
SELECT sel;  
  
always@(posedge in) begin  
    unique case (sel) /*unique | priority cases*/ //  
        violation - enum name 'D' is not defined  
    A: out = 2'b00;  
    B: out = 2'b01;  
    C: out = 2'b10;  
    default: out = 2'b00;  
    endcase  
end  
endmodule
```

Example 4

Consider the following example:

```
module test(input in, output reg out);  
  
typedef enum {A, B, C, D} SELECT; // A=0 , B=1 , C=2 , D=3
```

```
SELECT sel;

always@(posedge in) begin
  case (sel) // no-violation
    0: out = 2'b00;
    1: out = 2'b01;
    2: out = 2'b10;
    3: out = 2'b11;
    default: out = 2'b00;
  endcase
end
endmodule
```

Example 5

Consider the following example:

```
module test(input in, output reg out);

  typedef enum {A, B, C, D} SELECT;
  SELECT sel;

  always@(posedge in) begin
    case (sel) // no-violation
      A: out = 2'b00;
      B: out = 2'b01;
      C: out = 2'b10;
      D: out = 2'b11;
      default: out = 2'b00;
    endcase
  end
```

```
endmodule
```

Example 6

Consider the following example:

```
module test(input in, output reg out);
```

```
typedef enum {A, B, C, D} SELECT;
```

```
SELECT sel;
```

```
always@(posedge in) begin
```

```
case (sel) // no-violation
```

```
A: out = 2'b00;
```

```
B: out = 2'b01;
```

```
B+1: out = 2'b10;
```

```
D: out = 2'b11;
```

```
default: out = 2'b00;
```

```
endcase
```

```
end
```

```
endmodule
```

Example 7

Consider the following example:

```
module test(input in, output reg out);
```

```
typedef enum {A, B, C, D} SELECT;
```

```
SELECT sel;
```

```
always@(posedge in) begin
```

```
case (sel) inside // no-violation
```

```
[A:D]: out = 2'b01; // [A:D] -> A,B,C,D
```

```
default: out = 2'b00;  
endcase  
end  
endmodule
```

Example 8

Consider the following example:

```
module test(input in, output reg out);  
  
typedef enum {A, B, C, D} SELECT;  
SELECT sel;  
  
always@(posedge in) begin  
    case (sel) // no-violation  
        A, D: out = 2'b00;  
        B: out = 2'b01;  
        C: out = 2'b10;  
        default: out = 2'b00;  
    endcase  
end  
endmodule
```

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files

UniquelfMissingCond-ML

Reports violation if priority/unique type if/case statement is used but all the conditions are not covered

Language

Verilog

Description

The *UniqueIfMissingCond-ML* tag reports a violation if priority/unique type if/case statement is used but all the conditions are not covered.

The tag identifies the following keywords:

Unique: The unique keyword instructs all the software tools that each selection item in a series of decisions is unique from any other selection item in that series, and all the legal cases must be listed.

Priority: The priority keyword instructs all the software tools that each selection item in a series of decisions must be evaluated in the order in which they are listed, and all the legal cases must be listed.

Simulator: The unique/priority evaluates if all the conditions in parallel and simulator issues are run time error/warning when no condition is true and if condition doesn't have a corresponding else.

Synthesis: The synthesis tool optimizes the logic assuming that all other unlisted conditions are don't care keywords.

Violation Message

Used Unique if statement but all the conditions are not covered

Example Code and/or Schematic

Consider the following example:

```
module top ();  
  
    wire [1:0] SEL1,SEL2;  
    wire IN;  
    reg OP;
```

```
always @(SEL1 or SEL2)
begin
    unique /*or priority*/
    if(SEL1== 1'b1)
    begin
        OP <= 1'b0;
    end
    else if (SEL2 == 1'b0)
    begin
        OP <= IN;
    end
end
endmodule
```

The *UniqueIfMissingCond-ML* tag reports a violation for the following if condition:

```
if(SEL1== 1'b1);
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this tag:

```
Rule: UniqueIfMissingCond-ML | file: top.v | line: 10 |
MODULE: top | HIERARCHY: top
```

```
STATUS: REPORTED
```

```
Reason: Violation is reported because of the following
"Related FLPs" status.
```

```
Related FLPs:
```

```
top.else_missing_0: FALSIFIED
```

In the above example, all the conditions corresponding to if statement in the above statement are not covered. Therefore, the *UniqueIfMissingCond-ML* reports a violation for the above mentioned statement.

UnInitParam-ML

A parameterized module/entity has been instantiated but parameters/generics are not explicitly specified.

Language

Verilog, VHDL

Description

The UnInitParam-ML Tag flags parameterized module/entity instantiations that do not have parameters/generics specified.

Message Details

The following message appears at the location where parameterized module/entity instance `<inst-name>` is instantiated but parameters/generics are not specified:

Instance `<inst-name>` instantiated but parameter not specified

Severity

Warning

UnInitTopDuParam-ML

Top level module/entity has parameters/generics but the parameters/generics are not explicitly specified on the command line.

Language

Verilog, VHDL

Description

The UnInitTopDuParam-ML Tag flags parameters of top-level module/entity whose values are explicitly not passed through command line.

Message Details

The following message appears at the location of the parameter/generic *<param-name>* of top-level module/entity, whose value is explicitly not passed through the command line:

Parameter *<param-name>* is not passed through command line

Severity

Warning

UnifiedNetList-ML

Preferably have "UNIQUEIFIED" netlist

Tag Description

The *UnifiedNetList-ML* tag reports violation if the design contains multiple instances of the same module.

This tag highlights multiple instantiations in the RTL statements.

This tag does not report violation for multiple instantiation of black boxes and lib cells.

Language

Verilog+VHDL

Parameter(s)

viol_wrapper_module: By default, the value of this parameter is set to `no`, and the *UnifiedNetList-ML* tag does not report violation for wrapper modules. Set the value of this parameter to `yes` to report such cases.

Message Details

The following message appears if the design `<design-name>` contains multiple instances of module `<module-name>` :

```
Netlist design '<design-name>' contains multiple instantiation
of '<module-name>'
```

Example Code and/or Schematic

```
module top();

    slave inst2();
    slave inst3(); // Violation

    AN2 inst4 ();
```

```
    AN2 inst5 (); // No Violation for lib cell instances

endmodule

module slave();
    wrapper inst3();
    wrapper inst4(); // No Violation default. But give violation if
    "viol_wrapper_module" parameter set to "yes"

endmodule

module wrapper(); // This is a wrapper module, since there are only one
instance of BB
    wire a,b,c;
    BB inst(a,b,c);
endmodule
```

Tag Severity

Error

UninitializedReset-ML

Ensure that there are no uninitialized signals in reset block of a flip-flop.

When to Use

Use this Tag to check for uninitialized signals in the reset block of a flip-flop.

Description

The *UninitializedReset-ML* Tag checks whether there are any uninitialized signals in reset block of a flip-flop.

Language

Verilog

Parameter

ignore_sync_reset: The default value of the parameter is no. Set the value of the parameter to yes to report violations only for the flip-flops that have an asynchronous reset.

Constraints

None

Messages and Suggested Fix

[WARNING] Signal '<Signal name>' is not initialized in reset block.

Potential Issues

The violation message is reported when there is an uninitialized signal in reset block of a flip-flop.

Consequences of Not Fixing

Not fixing the violation may result in unexpected design behavior.

How to Debug and Fix

To debug the violation, cross-probe to the HDL description. Click the violation message to highlight the assignment in flip-flop in the design.

Example Code and/or Schematic

Example 1: ignore_sync_reset = no

```
always_ff @(posedge clk) begin: _ff_1
    if (rst == 1'b1) begin
        a <= 1'b0;
    end
    else begin
        a <= in1;

        if (a == 1'b1) begin
            b <= in2;//violation since b has not
            initialized in if block
        end
    end
end: _ff_1
```

Example 2: ignore_sync_reset = no

```
always_ff @(posedge clk) begin
    if (rst == 1'b1) begin
        other_flop2 <= 0;
    end
    else begin
        my_flop2 <= in_case2;//violation since my_flop2 has not
        initialized in if block
    end
end
```

Example 3: ignore_sync_reset = yes

```
always_ff @(posedge clk or posedge rst) begin
    if (rst == 1'b1) begin
        other_flop2 <= 0;
    end
    else begin
        my_flop2 <= in_case2;//violation since my_flop2 has not
        initialized in if block
    end
end
```

MoreLint Tags

```
always_ff @(posedge clk) begin
  if (rst == 1'b1) begin
    other_flop <= 0;
  end
  else begin
    my_flop <= in_case;//no violation since this is a
    synchronous flip-flop
  end
end
```

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

UniqueCase-ML

Unique, Unique0, or priority case constructs used with full_case or parallel_case or both.

Language

SystemVerilog

Description

The UniqueCase-ML Tag flags if unique/Unique0/priority case statements are used with full_case or parallel_case pragmas.

Verilog case statements (including casez and casex) have the following coding issues that often result in design problems:

- Not all possible branches need to be specified (incomplete decisions)
- Redundant (duplicate) decision branches can be specified
- Software simulation evaluate decisions in the order listed (priority decoding), but the decision might be able to be evaluated in any order in hardware (parallel decoding).

To overcome the above case problems, Verilog synthesis compilers use synthesis full_case and parallel_case pragmas. However, it is dangerous to use these pragmas as they may introduce further problems than they can solve.

SystemVerilog adds two decision modifiers, unique and priority, to eliminate all the problems due to incomplete and redundant decision statements and to prevent the common problems caused by synthesis full_case and parallel_case pragmas.

This Tag reports a violation if full_case or parallel_case pragmas are used with unique/priority case statements, as shown in the following examples:

Example 1

```
unique case (sel) // synopsys full_case
    ...
endcase
```

Example 2

```
unique case (sel) //synopsys parallel_case
```

```
...
endcase
```

Example 3

```
priority case (sel) // synopsys full_case
...
endcase
```

Example 4

```
priority case (sel)
//synopsys parallsel_case full_case
...
endcase
```

Example 5

```
unique0 case (sel) // synopsys full_case
...
endcase
```

Example 6

```
unique0 case (sel) //synopsys parallel_case
...
Endcase
```

Message Details

The following message appears at the location where `full_case/parallel_case` pragma is used along with the `unique/priority case` statements:

'full_case | parallel_case' used along with 'unique | unique0 | priority' case statements

Potential Issues

The violation message is reported at the location where `full_case/parallel_case` pragma is used along with the `unique/priority case` statements.

Consequences of Not Fixing

Using the `full_case` or `parallel_case` pragmas with `unique`/`priority` case statements can affect synthesis behavior.

Severity

Warning

Suggested Fix

Do not use `full_case` or `parallel_case` along with `unique`/`priority` statement.

UniquePriorityMisuse-ML

Detects if unique, unique0, and priority keywords are misused in case constructs

Description

The *UniquePriorityMisuse-ML* Tag reports a violations when unique, unique0, and priority keywords are used before case and if. If unique0 is present unique and priority keywords, the Tag reports a violation for unique0 only.

The tag also considers case-inside statement for tag checking.

Language

Verilog

Messages and Suggested Fix

The following message appears when funique, unique0,and priority keywords are used before case and if:

[2-MEDIUM] Use of <unique | priority | unique0> construct in <if/case> block

Example Code and/or Schematic

Example 1

```
module top();
    reg dff;
    wire clk,din,reset;
    always @ (posedge clk)
        unique if (reset) begin //Violation
            dff <= 0;
        end else begin
            dff <= din;
        end
endmodule
```

In the above example, the Tag reports a violation because unique is used

before if.

Example 2

Consider the following example:

```
module mux (a,b,c,d,sel,y);  
input a, b, c, d;  
input [1:0] sel;  
output y;  
reg y;  
always @ (a or b or c or d or sel)  
unique case (sel) //Violation  
    0 : y = a;  
    1 : y = b;  
    2 : y = c;  
    3 : y = d;  
    default : $display("Error in SEL");  
endcase  
endmodule
```

In the above example, the Tag reports a violation because unique is used before case.

Default Severity Label

2-Medium

Reports and Related Files

None

UnsuppCompDir-ML

``ifdef`, ``resetall`, ``default_nettype` and ``unconnected_drive` are not supported by all tools

Language

Verilog

Description

The UnsuppCompDir-ML Tag flags unsupported compiler directives.

The UnsuppCompDir-ML Tag flags the use of `'ifdef`, `'resetall`, `'default_nettype`, and `'unconnected_drive` compiler directives.

NOTE: *The UnsuppCompDir-ML Tag does not flag when the `'default_nettype` compiler directive is set to `wire`.*

These compiler directives are not supported by all synthesis tools.

Parameter(s)

[check_directive_type](#): By default, the parameter set to `""`. Set the value of the `check_directive_type` parameter to one or a comma/ space-separated list of compiler directives to enable the tag to report a violation for the use of ``ifdef`, ``resetall`, ``default_nettype` and ``unconnected_drive` compiler directives.

Message Details

The following message appears at the location of an unrecognized synthesis directive of type `<pragma>`:

compiler directive `<pragma>` is not supported by all synthesis tools

Severity

Warning

Suggested Fix

Avoid using these directives in the synthesizable logic.

UnusedFunctionInput-ML

Function input has been declared but it is not read

Language

Verilog

Description

The *UnusedFunctionInput-ML* Tag reports a violation if a function input has been declared but not read.

This probably indicates an error. Modules using this function might be connecting something to that input and expecting it to be used in some way. It is most likely a result of a typographical error.

By default, the Tag reports a violation if any bit of a function input is declared but not read in the current scope of a module. If you set the [checkfullbus](#) parameter to `yes`, the Tag reports a violation only when all bits of a function input are completely unread in the current scope of a module.

By default, this Tag does not process large arrays. Set the [handle_large_bus](#) parameter to `yes` to process large arrays (greater than 50,000) and report violation if not used correctly.

By default, the *UnusedFunctionInput-ML* Tag does not check the input ports of tasks. Set the [checktaskarg](#) parameter to `yes` to enable this Tag to check the input ports of tasks.

Message Details

Message 1

The following message appears at the location of a function input `<input>` that has been declared but not read:

Function Input '<input>' declared but not read.[Hierarchy: '<hier-path>']

Message 2

The following message appears at the location of a task input `<input>` that has been declared but not read, when the [checktaskarg](#) parameter is set as `yes`:

MoreLint Tags

Task Input '<input>' declared but not read.[Hierarchy: '<hier-path>']

Examples

Consider the following example:

```
module function_calling(a, b, c, d, e, f);

    input a, b, c, e;
    input [1:0]d ;
    output f;
    wire f;
function myfunction;
    input a, b, c;
    input [2:0] d;
    reg out;
    begin
        myfunction = ((a+b) + (c-d[2]));
    end
endfunction

    assign f = (myfunction (a,b,c,d)) ? e :0;

endmodule
```

In this example, the *UnusedFunctionInput-ML* Tag does not report a violation when you set the *checkfullbus* parameter to yes.

Severity

Warning

UnpackedStructUsed-ML

Do not declare unpacked structure outside the package

Language

SystemVerilog

Description

The UnpackedStructUsed-ML reports violation for unpacked structures and unpacked `typedef structs` declared outside the package declaration.

The Tag checks at all scopes where structures are defined, like interfaces, functions, tasks, and named blocks.

The Tag reports violation for the following cases:

- unpacked struct objects
- declaration of unpacked struct when typedef is used
- unpacked struct defined at ports of a module

However, the Tag does not report violation for unpacked structures, declared inside packages.

By default [report_all_unpacked](#) parameter is set to no. Set the value of the parameter to yes to enable the Tag to report violations for all the declarations that include an unpacked dimension.

Message Details

Message 1

[WARNING] Unpacked <struct/array> "<type_name>" is defined within <module/interface> "<scope name>"

Potential Issues

The violation message is reported at unpacked array and struct definitions.

Consequences of Not Fixing

This tag is used to follow better coding practice.

How to Debug and Fix

To fix the violation, ensure that an unpacked structure is not defined within a module or interface.

Message 2

[WARNING] Unpacked <struct/array> variable "<variable_name>" is declared within <package/module/interface/global/function/task> "<scope_name>"

Potential Issues

The violation message is reported at unpacked array and struct declarations.

Consequences of Not Fixing

This tag is used to follow better coding practice.

How to Debug and Fix

To fix the violation, ensure that an unpacked structure is not declared outside the package.

Message 3

Unpacked <array/struct> "<name>", member of unpacked struct "<type_name>" is defined within <package/module/interface/global/function/task> "<scope name>"

Potential Issues

The violation message is reported for unpacked array members of structs.

Consequences of Not Fixing

This tag is used to follow better coding practice.

How to Debug and Fix

To fix the violation, ensure that an unpacked structure is not declared outside the package.

NOTE: *The above violation messages are reported for a package as well when the [report_all_unpacked](#) parameter is set to yes.*

Tag Severity

Warning

Examples

Example 1

Consider the following example:

```
module test();  
  struct {  
    integer a;  
    logic [5:0] b;  
  } st1;  
  
  struct packed { //No violation, as packed structure is used.  
    integer a;  
    logic [5:0] b;  
  } st2;  
  
endmodule
```

In the above example, the Tag reports violation for the first `struct` object as it is an unpacked struct object. However, the Tag does not report violation for the second `struct` object as it is a packed structure.

Example 2

Consider the following example:

```
typedef struct {  
  integer a;  
  logic [5:0] b;  
} st_type;  
  
module test();  
  st_type d1;  
  st_type d2;  
endmodule
```

In the above example, the Tag reports a violation at the declaration of `struct`, since `typedef` is used.

Example 3

Consider the following example:

```
package p1;
```



```
typedef struct {    // No violation
    logic [7:0] newPack [3:0];
    logic [31:0] Addr;
} my_new_type;
endpackage

module receiver(input [7:0] p1, input [31:0] newAd);
import p1::my_new_type;

my_new_type collect;
endmodule
```

In the above example, the Tag does not report a violation as unpacked structures are declared inside package.

Example 4

Consider the following example:

```
module test (input struct {logic l1 [3:0];} st1, input
clock);
endmodule
```

In the above example, the Tag reports a violation as struct is defined at ports of a module.

UseBusWidth-ML

Bits of a bus should be specified when a bus is used as a whole

Language

Verilog, VHDL

Description

The UseBusWidth-ML Tag flags bus signal usages where the bit-width is not indicated every time the bus is read or set as a whole.

By default, the UseBusWidth-ML Tag checks all assignment statements. Set the value of the [strict](#) Tag parameter to `yes` or `UseBusWidth-ML` to check function calls, task calls, and actual arguments in any instantiation.

NOTE: *Presently, the UseBusWidth-ML Tag in Verilog mode does not check for bus signal usages in conditional expressions, if statement conditional expressions, case construct expressions, and always construct conditional expressions, irrespective of the strict Tag parameter setting.*

To avoid excessive messages, the number of messages has been restricted to 20 per module and 500 per design by default (in both Verilog and VHDL). Set the [allviol](#) Tag parameter to `yes` or `<Tag-name>` to report all violation messages.

NOTE: *For VHDL, this Tag does not flag messages for formal arguments in case of port mapping (instantiations).*

Message Details

The following message appears at the location where a bus signal `<name>` is read or set without the bit-width indicated:

Bus-width not specified for the bus '`<name>`'

Severity

Guideline

Examples (Verilog)

Consider the following example flagged by the UseBusWidth-ML Tag:

```
module test(in1, in2, out1);
```

MoreLint Tags

```

input [2:0] in1, in2;
output [2:0] out1;

reg [2:0] out1;
wire en1;

always @(en1)
  if(en1)
    out1 = in1 & in2;
endmodule

```

The bus signals out1, in1, and in2 have been used in the always construct without bit-width indicated. Hence, SpyGlass generates the following messages:

Bus-width not specified for the bus 'in1'

Bus-width not specified for the bus 'in2'

Bus-width not specified for the bus 'out1'

Examples (VHDL)

Consider the following example where bus signals in1 and out1 are used without specifying the bits:

```

library ieee;
use ieee.std_logic_1164.all

entity top is port(
  in1: std_logic_vector( 7 downto 0);
  out1: out std_logic_vector ( 7 downto 0));
end top;

architecture arc of top is
begin
  process(in1)
  begin
    out1 <= in1;
  end process;
end arc;

```

For this example, SpyGlass generates the following messages:

Bus-width not specified for the bus 'out1'

Bus-width not specified for the bus 'in1'

To fix this message, modify the assignment statement as follows:

```
out1(7 downto 0) <= in1(7 downto 0);
```

UseImplicitAlways-ML

Sensitivity list must be implicit in combinational always blocks

When to Use

Use this Tag to report combinational always block that contains any signals in sensitivity list.

Description

If always block contains only combinational logic, the sensitivity list should be implicit ("*").

This Tag reports violations, if combinational always block contains any signals in sensitivity list.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Combinational always block is not allowed with non-implicit sensitivity list '<sensitivity list>'. [Hierarchy: <hierarchy>]

Potential Issues

The violation message is reported when a combinational always block contains any signals with the sensitivity list.

Consequences of Not Fixing

Not fixing the violation may result in unexpected Tag behavior.

How to Debug and Fix

Click on the violation message to highlight the violating always block in the sensitivity list.

Example Code and/or Schematic

Example 1

```
module test( a , b , out);  
    input a , b ;  
    output reg out ;  
    always @(a or b) begin //violation  
        out = a & b;  
    end  
endmodule
```

Example 2

```
module test( a , b , out);  
    input a , b ;  
    output reg out ;  
    always @(*) begin //No violation  
        out = a & b;  
    end  
endmodule
```

Example 3

```
module test(rst , clk , in , out);  
    input rst , clk , in ;  
    output reg out ;  
    always @(posedge clk) begin // No violation ( sequential always block )  
        if(!rst)  
            out <= 0;  
        else  
            out <= in;  
        end  
    end  
endmodule
```

Example 4

```

module test( en , in , out);
    input en , in ;
    output reg out ;
    always @(en , in ) begin //violation
        if(en)
            out = in;
        end
    endmodule

```

Example 5

```

module test(rst , clk , in , out);
    input rst , clk , in ;
    output reg out ;
    always_ff @(posedge clk) begin // No violation ( sequential always
block )
        if(!rst)
            out <= 0;
        else
            out <= in;
        end
    endmodule

```

Example 6

```

module test( en , in , out);
    input en , in ;
    output reg out ;
    always_latch begin // No violation
        if(en)
            out <= in;
        end
    endmodule

```

```
endmodule
```

Example 7

```
module test( a , b , out);  
    input a , b ;  
    output reg out ;  
    always_comb begin // No violation  
        out = a & b;  
    end  
endmodule
```

Example 8

```
module test( a , b , out);  
    parameter w = 5;  
    input [w-1 : 0 ]a , b ;  
    output reg [w-1 : 0 ] out ;  
    genvar i ;  
    for ( i = 0; i < w ; i = i +1 ) begin  
        always @(a[i] or b[i]) begin // violation  
            out = a[i] & b[i];  
        end  
    end  
endmodule
```

Default Severity Label

Warning

Tag Group

Not applicable

[MoreLint Tags](#)

Reports and Related Files

No related reports or files.

UseLogic-ML

Checks for reg and wire usage in the design

When to Use

Whenever there is a declaration of a signal in the System Verilog design.

Description

As Verilog and System Verilog both are loosely type languages, it allows designers to write construct that may lead to some serious design bugs. Ports declared as `wire` or `reg` are converted into inout ports, which is a language feature. This conversion allows you to write output port declared as `wire` into input port. To avoid this, you can use logic. Output port declared as logic cannot be written into inout port.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when a signal is declared as `wire` in System Verilog design:

```
[WARNING] Signal <signal-name> is declared as 'wire/reg'  
[Hierarchy: 'Hier-Name']
```

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

Double click the violation message. The HDL viewer window highlights the line where violation is detected. Fix the violation by using logic instead of `wire` or `reg` construct.

Example Code and/or Schematic

In the following example, the *UseLogic-ML* Tag reports a violation for signal 'in' because it is declared as 'wire' in System Verilog design:

```
module TOP(input in, output out);  
    wire in;    // UseLogic-ML report violation  
    logic out; // No violation for UseLogic-ML as logic is used  
    assign out = in;  
endmodule
```

The following message is reported for the above example:

Signal 'in' is declared as 'wire' [Hierarchy: 'TOP']

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

UseSVCasting-ML

Checks if System Verilog casting constructs are used for assigning a packed bit vector to a packed structure

When to Use

When assigning a packed bit vector to a packed structure.

Description

This Tag checks for type casting and size casting when a packed bit vector is assigned to a packed structure. The Tag reports a violation when System Verilog casting constructs are not used for assigning a packed bit vector to a packed structure.

Default Weight

5

Language

System Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears when type casting is not used while assigning a packed bit vector to a packed structure:

[WARNING] Use SV type casting before assigning a packed bit vector <RHS_Expr> to a packed structure. [Hierarchy: 'Hier_name']

Potential Issues

A packed bit vector is assigned to a packed structure without type casting.

Consequences of Not Fixing

None

How to Debug and Fix

Double click the violation message. The HDL viewer window highlights the line where violation is detected. Fix the violation by using type casting.

Message 2

The following message appears when a packed bit vector is assigned to a packed structure of different size without using size casting:

[WARNING] Use SV size cast of appropriate size before assigning packed bit vector <RHS_Expr> to a packed structure. [Hierarchy: 'Hier_name']

Potential Issues

A packed bit vector is assigned to a packed structure of different size without using size casting.

Consequences of Not Fixing

None

How to Debug and Fix

Double click the violation message. The HDL viewer window highlights the line where violation is detected. Fix the violation by using size casting.

Example Code and/or Schematic

Example 1

In the following example, a packed vector `flat_w1` is assigned to a packed structure `obj2` without type casting:

```
typedef struct packed {
    logic [7:0] [15:0]w1;
    logic [7:0] [15:0]w2;
} mystruct1;

module test();
    wire [15:0] [15:0]flat_w1;
    mystruct1 obj1;
    mystruct1 obj2;
    assign flat_w1 = obj1;    // No viol
```

```
    assign obj1 = mystruct1'(flat_w1); // No viol
    assign obj2 = flat_w1; // Violation
endmodule
```

The *UseSVCasting-ML* Tag reports the following message for the above example:

Use SV type casting before assigning a packed bit vector 'flat_w1' to a packed structure

Example 2

In the following example, a packed vector flat_w1 is assigned to a packed structure obj2 after type casting, but there is a size mismatch.

```
typedef struct packed {
    logic [7:0] [15:0]w1;
    logic [7:0] [15:0]w2;
} mystruct1;

module test();
    wire [14:0] [14:0]flat_w1;
    mystruct1 obj1;
    mystruct1 obj2;
    assign obj2 = mystruct1'(flat_w1);
endmodule
```

The *UseSVCasting-ML* Tag reports the following message for the above example:

Use SV size cast of appropriate size before assigning packed bit vector 'mystruct1'(flat_w1)' to a packed structure

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

VarInCaseInside-ML

Variable or function call in case-inside item

When to Use

Use this tag to detect variable or function call in case-inside item.

Description

This tag reports violations for a variable or function calls used as case item in case-inside constructor. Variables/function calls inside the case item causes potential mismatches between simulation and synthesis. In simulation, value 'x' matches with both '0' and '1'. However, in synthesis, value 'x' only matches with '0'.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

Message

[ERROR] variable/function call '<variable/function_call>' is used as a case item in case inside [Hierarchy: '<hierarchy>']

Potential Issues

A violation is reported is when a variable/ function call is used as a case item in case-inside constructor.

Consequences of Not Fixing

Not fixing a violation may result in unexpected tag behavior.

How to Debug and Fix

Double click the violation message. The HDL viewer window highlights the RTL line where the variable is used as case item inside case inside.

Example Code and/or Schematic

Example 1

```
module test(output O);  
  reg [1:0] R;  
  always_comb begin  
    case (2'b01) inside  
      R : O = 1'b1;  
    default : O = 1'b0;  
  endcase  
end  
endmodule
```

Example 2

```
module test(output O);  
  reg [1:0] R;  
  always_comb begin  
    R = 2'b01;  
    case (2'b01) inside  
      R : O = 1'b1;  
    default : O = 1'b0;  
  endcase  
end  
endmodule
```

Example 3

```
module test(output O);  
  reg [1:0] R;  
  always_comb begin  
    R = 2'b11;
```


MoreLint Tags

```
case (2'b01) inside
  R : O = 1'b1;
default : O = 1'b0;
endcase
end
endmodule
```

Example 4

```
module test(output O);
  reg [1:0] Q, R;
  always_comb begin
    case (2'b01) inside
      Q | R : O = 1'b1;
    default : O = 1'b0;
    endcase
  end
endmodule
```

Example 5

```
module test(output O);
  reg [1:0] Q, R;
  function somefunction (input [1:0] Q, R);
    somefunction = Q + R;
  endfunction
  always_comb begin
    case (2'b01) inside
      somefunction(Q,R) : O = 1'b1;
    default : O = 1'b0;
    endcase
  end
endmodule
```

```
end  
endmodule
```

Example 6

```
module test(output O);  
  reg [1:0] R;  
  always_comb begin  
    case (R) inside  
      2'b00 : O = 1'b1; // No errors  
    default : O = 1'b0;  
    endcase  
  end  
endmodule
```

Example 7

```
module test(output O);  
  reg [1:0] Q, R;  
  always_comb begin  
    case (2'b01) inside  
      R : O = 1'b1;  
      Q : O = 1'b0;  
    default : O = 1'b0;  
    endcase  
  end  
endmodule
```

Example 8

```
module test  
  #(parameter logic [1:0] R = 2'b01)  
  (output logic O);  
  always_comb begin
```

MoreLint Tags

```
case (2'b01) inside
R : O = 1'b1; // No errors
default : O = 1'b0;
endcase
end
endmodule
```

Default Severity Label

Error

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

V2KConstruct-ML

Verilog 2001 construct used

Language

Verilog

Description

The V2KConstruct-ML Tag flags Verilog 2001 constructs in the design.

The V2KConstruct-ML Tag does not flag sized parameters as they are supported in SpyGlass by default.

NOTE: *By default, SpyGlass supports Verilog 2000 constructs. Therefore, the V2KConstruct-ML Tag has been switched off and you need to run this Tag using the `configure_lint_tag -enable -tag <V2KConstruct-ML>` option.*

Message Details

The following message appears at the location where a Verilog 2001 construct `<const>` is used:

verilog 2001 construct (`<const>`) found

Severity

Info

ValueSizeOverFlow-ML

Input value should not be greater than size specified in IEEE conversion functions

Language

VHDL

Description

The ValueSizeOverFlow-ML Tag flags mismatched inputs to function calls.

The ValueSizeOverFlow-ML Tag flags cases where the size of input value is more than the size specified in the corresponding IEEE conversion function call.

By default, the *ValueSizeOverFlow-ML* Tag checks assignments related to the following IEEE conversion functions:

```
To_StdUlogicVector (val : integer; size : integer ) return
std_ulogic_vector;

To_StdlogicVector   (val : integer; size : integer ) return
std_logic_vector;

CONV_UNSIGNED(ARG: INTEGER; SIZE: INTEGER) return UNSIGNED;
CONV_UNSIGNED(ARG: UNSIGNED; SIZE: INTEGER) return UNSIGNED;
CONV_UNSIGNED(ARG: SIGNED; SIZE: INTEGER) return UNSIGNED;
CONV_UNSIGNED(ARG: STD_ULOGIC; SIZE: INTEGER) return
UNSIGNED;

CONV_SIGNED(ARG: INTEGER; SIZE: INTEGER) return SIGNED;
CONV_SIGNED(ARG: UNSIGNED; SIZE: INTEGER) return SIGNED;
CONV_SIGNED(ARG: SIGNED; SIZE: INTEGER) return SIGNED;
CONV_SIGNED(ARG: STD_ULOGIC; SIZE: INTEGER) return SIGNED;
CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER) return
STD_LOGIC_VECTOR;
CONV_STD_LOGIC_VECTOR(ARG: UNSIGNED; SIZE: INTEGER) return
STD_LOGIC_VECTOR;
CONV_STD_LOGIC_VECTOR(ARG: SIGNED; SIZE: INTEGER) return
```

```

STD_LOGIC_VECTOR;

CONV_STD_LOGIC_VECTOR(ARG: STD_ULOGIC; SIZE: INTEGER) return
STD_LOGIC_VECTOR;

EXT(ARG: STD_LOGIC_VECTOR; SIZE: INTEGER) return
STD_LOGIC_VECTOR;

SXT(ARG: STD_LOGIC_VECTOR; SIZE: INTEGER) return
STD_LOGIC_VECTOR;

```

Such mismatches cause overflow and the function cannot convert the input value correctly.

To enable the Tag to also check for assignments related to TO_SIGNED and TO_UNSIGNED IEEE conversion functions, set the value of the parameter [check_conversion_function](#) to a comma-separated list of these IEEE conversion functions.

The following describes the syntax of TO_SIGNED and TO_UNSIGNED conversion functions:

```

TO_SIGNED (ARG: INTEGER, SIZE: NATURAL) return SIGNED VECTOR;

TO_UNSIGNED (ARG: UNSIGNED INTEGER, SIZE: NATURAL) return
UNSIGNED VECTOR;

```

By default, the [report_underflow](#) parameter is set to no. Set this parameter to yes to also report the underflow under certain conditions. Refer to the [report_underflow](#) section for details.

By default, the tag does not report violation for cases for cases where to_integer function is parsed with an argument whose bit width is larger than 32. Set the value of the [report_to_integer](#) parameter to yes to report violation for such cases.

Message Details

The following message appears at the location where the size of the input expression *<expr>* of a function call *<func-name>* is more than the size *<num>* of the function:

Size of expression '*<expr>*' ('*<expr-size>*' bits) is more than the size of '*<expr>*' ('*<expr-size>*' bits) specified in function '*<func-name>*'

If the `report_underflow` parameter is set to yes under the conditions

MoreLint Tags

mentioned in the [report_underflow](#) section, the following message is also reported:

Size of expression '<expr>' mismatches with the size '<num>' specified in function '<func-name>'

Severity

Warning

WrapInstance-ML

Black box cell should not be contained on its own in a wrapper module

Description

The WrapInstance-ML tag flags unwanted black box instances in a wrapper design unit.

The WrapInstance-ML Tag requires you to specify the names of the allowed cells in a design unit using the [wrapperDuCellList](#) Tag parameter as in the following example:

```
configure_lint_tag_parameter -tag WrapInstance-ML -parameter  
wrapperDuCellList -value "DU1 c1 c2 c3, DU2 c2 c5"
```

Here, instances of cells c1, c2, and c3 are allowed in the design unit DU1 where instances of cells c2 and c5 are allowed in the design unit DU2.

Then, the WrapInstance-ML Tag flags all other black box instances (instances of cells other than the respective allowed cells or instances of design units whose RTL description is not available) in design units DU1 and DU2.

For mapping FPGA design to an ASIC implementation, it is necessary that there should not be any black box cell in a wrapper design unit. For performing this, the WrapInstance-ML Tag checks that all instantiated cells are from a specified set only.

NOTE: *The WrapInstance-ML Tag supports generate-for block.*

Message Details

The following message appears at the location where a Tag-violating black box cell `<bb-name>` is instantiated on inside a wrapper design unit `<du-name>`:

Instantiation of blackbox '`<bb-name>`' in the design unit '`<du-name>`' is not correct

Severity

Warning

ZeroReplicationMultiplier-ML

Reports replications with a zero replication multiplier in Verilog

Description

The ZeroReplicationMultiplier-ML Tag reports replications with a zero replication multiplier. Although, Verilog allows this usage, such replications may cause simulation/synthesis mismatch.

The following replication scenarios are considered by this Tag:

- Inside modules
- Inside used interfaces
- Inside used functions and tasks
- Inside generate blocks

Language

Verilog

Message Details

The following message appears a when a replication with a zero replication multiplier is found:

[ERROR] zero Replication multiplier used in concat '<concat-expr>' [Hierarchy : <hierarchy-string>]

Examples

Consider the following code snippet:

```
assign out = {in1, {0{in2}}};
```

In this example, `{0{in2}}` has 0 as a replication multiplier, and the Tag reports a violation for this.

Default Severity Label

Error

MemConflict-ML

Possible memory conflict (overwrite) detected

Language

Verilog

Description

The MemConflict-ML Tag flags memory conflicts.

There should not be memory conflicts during assignments to memory elements in different always constructs in the same module. Such conflicts can lead to unknown values in a memory element.

The MemConflict-ML Tag considers the always block present between the two branches of the generate block as mutually exclusive block.

NOTE: *The MemConflict-ML Tag supports generate-if, generate-for, and generate-case blocks.*

Message Details

The following message appears at the location (line <num2>) where a memory element <mem-name> is being assigned again when it was earlier assigned (at line <num1>) in a different always construct in the same module:

<mem-name> is being written from more than one always blocks at <num1> and <num2>

Severity

Warning

Examples

Consider the following example where the memory element mem is being assigned in two always constructs in the same module:

```
module test(in, addr2, addr1);  
    input [31:0] in;  
    input addr1, addr2;
```

MoreLint Tags

```
reg [31 : 0] mem[12 : 0];

always @(in or addr1)
    mem[addr1] = in;

always @(in or addr2)
    mem[addr2] = ~in;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

mem is being written from more than one always blocks at 8 and 11

NOTE: *The index values addr1 and addr2 are variables and their actual values are not known at the time of simulation. As these values can be same, VC SpyGlass Lint flags the above example.*

MultiAssign-ML

Signal or Variable may be multiply assigned (beside initialization) in the same process scope.

Language

VHDL

Description

The MultiAssign-ML Tag flags signals/variables that are assigned more than once in the same process scope.

The MultiAssign-ML Tag flags the following cases:

- An assignment that overrides a previous assignment (other than initialization) to the same signal/variable within the same process
- A signal/variable is assigned in two parallel conditional blocks and these conditions can be TRUE simultaneously.

Some of the cases flagged by the MultiAssign-ML Tag are as follows:

```
library ieee;  
use ieee.std_logic_1164.all
```

```
...  
if(cond1) then  
    a <= b;  
end if;  
if(cond2) then  
    a <= c;  
end if;  
...
```

```
...  
if(cond1) then  
    a <= b;  
end if;  
a <= c  
...
```

```
...  
a<=b;  
a<=d;  
...
```

The MultiAssign-ML Tag does not flag the following example as it is a common style of RTL modeling:

```
...
a <= b;
if (C1) then
  a <= b1;
  if (C12) then
    a <= b12;
  end if;
end if;
...
```

Message Details

The following message appears at the location where signal/variable *<name>* is being assigned again when it was earlier assigned (at line *<num>*) in the same process scope:

<obj-type> '*<name>*' is already assigned previously at line *<num>*

Where *<obj-type>* can be Signal or Variable.

Severity

Warning

Suggested Fix

Aside from overwriting an initial value, multiple overwrites of a value within a block is at least confusing. In some cases, it may indicate an error. Thus, it is recommended to rewrite your code to remove multiple overwrites.

NestedCaseStmt-ML

Nesting depth of case construct exceeds specified depth.

Language

Verilog, VHDL

Description

The NestedCaseStmt-ML Tag flags deeply nested case constructs.

By default, the NestedCaseStmt-ML Tag flags nested case constructs with more than one level of nesting as in the following example:

```
...
case :           //depth is one here
    ...
    case :       //depth is two here.
        ...
    endcase
endcase
...
```

Use the [casedepth](#) Tag parameter to specify a different nesting depth.

Message Details

The following message appears at the location of a nested case construct of nesting depth more than \$casedepth:

Nested case construct found. Depth exceeding specified limit \$casedepth

Severity

Info

NestedCellDefine-ML

There is a ``celldefine` in the source before a ``endcelldefine` to close the previous ``celldefine`

Language

Verilog

Description

The NestedCellDefine-ML flags nested 'celldefine/endcelldefine constructs.

The NestedCellDefine-ML flags 'celldefine statement that follows another 'celldefine statement instead of the expected 'endcelldefine statement.

Message Details

The following message appears at the location of a 'celldefine statement that follows another 'celldefine statement instead of the expected 'endcelldefine statement:

Nested `celldefine directive found

Severity

Info

NoAssignX-ML

Ensure RHS of the assignment does not contains 'X'

When to Use

Use this tag to identify assignments that have static value containing X on RHS.

Description

The NoAssignX-ML tag reports violation for assignments that have static value containing X on right-hand side of the assignment.

By default, the NoAssignX-ML tag does not report any violation for the default clause of case statement. Set the value of the [check_xassign_casedefault](#) parameter to `yes` to report violation for such cases.

Default Weight

5

Language

Verilog

Parameter(s)

- [check_enum_decl](#): By default, the parameter is set to `no` and the *NoAssignX-ML* tag does not report violations for enum declarations where x values are assigned to enum elements. Set the value of this parameter to `yes` to report violations for enum declarations where x values are assigned to enum elements.
- [ignore_fullcase_default](#): By default, the parameter is set to `no`. Set this parameter to `yes` to not report a violation for default clauses of a fully covered case when the `check_xassign_casedefault` parameter is set to `yes`.

Constraint(s)

None

Messages and Suggested Fix

The following message at the at the location of Tag violating assignment

[WARNING] RHS of the assignment contains 'X' <reason>

Where,

<reason> can have one of the following values:

- Reason: Initialization to X
- Reason: Default clause assignment in incompletely specified case
- Reason: Default clause assignment in fully specified case

Potential Issues

Violation may arise when the right-hand side of an assignment has static value containing x.

Consequences of Not Fixing

Simulators treat X as literal X whereas synthesis tools replace it with 0 or 1 to optimize the circuit. Thus, designs with such assignments may have mismatches in pre- and post-synthesis simulation results.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the assignment statement where the RHS of the assignment contains X value. The value, X, is evaluated from one of the following sources:

- Direct hard-coded X assignment
- Parameter, local parameter, macros, or constants
- Function call, which returns X
- Expression which evaluates to X

To fix the violation, remove x from the right-hand side of the assignment statement.

Example Code and/or Schematic

Example 1

Consider the following example:

```
always@( sell) begin
```

```

out1 <= 4'bxxxx;
case (sel1)
  2'b00 : out1 <= in1 && in2;
  2'b01 : out1 <= in1 || in2;
  2'b10 : out1 <= in1 & in2;
  2'b11 : out1 <= in1 | in2;
  default :
    out1 <= 4'bxxxx;
endcase
case (sel1)
  2'b00 : out2 <= in1 && in2;
  default :
    out3 <= 4'bxxxx;
endcase
end

```

The NoAssignX-ML Tag generates the following message for the above example:

violation 1: RHS of the assignment contains 'X'(Reason : Initialization to 'x')

Also, the Tag reports the following violation messages for the above example when the value of the [check_xassign_casedefault](#) parameter is set to yes:

violation 2: RHS of the assignment contains 'X'(Reason : Default clause assignment in fully specified case)

violation 3: RHS of the assignment contains 'X'(Reason : Default clause assignment in incompletely specified case)

Example 2

Some examples of assignments flagged by the NoAssignX-ML Tag where out and wire1 are assigned values containing x:

```

module test(out,wire1);
output out,wire1;
wire wire1;
assign out = 1'bx;
assign wire1 = 2'b1x;
`define supply0 2'bxx

```

MoreLint Tags

```
wire [1:0] wire2 = `supply0;  
assign wire3 = `supply0;  
endmodule
```

The NoAssignX-ML Tag generates the following violation message at 'x' assignment for the above examples:

RHS of the assignment contains 'x'

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

NoDisableInFunc-ML

Disable statements in functions may not be synthesizable

Language

Verilog

Description

The NoDisableInFunc-ML Tag flags disable statements used in the function descriptions.

Use of disable statements within function is generally not supported by synthesis tools.

Message Details

The following message appears at the location where a disable statement is encountered in a function description:

'disable' statement used in function

Severity

Warning

NoDisableInTask-ML

Disable statements in tasks may not be synthesizable

Language

Verilog

Description

The NoDisableInTask-ML Tag flags disable statements used in task descriptions.

Use of disable statements within task descriptions is generally not supported by synthesis tools.

Message Details

The following message appears at the location where a disable statement is encountered in a task description:

'disable' statement used in task

Severity

Warning

NoExprInPort-ML

Port connections in instances should not contain expressions

Language

Verilog

Description

The NoExprInPort-ML Tag flags instance port connections using expressions.

The NoExprInPort-ML Tag does not flag port connections using concatenation operation.

Such port connections are not recommended as they result in generation of an intermediate net, which is the result of the port expression. The intermediate net is then effectively connected to the port of the instance.

By default, the `lint_check_top_du` application variable is set to `no` and the *Tag* reports violation for logic expressions which are present in port connections in the instances inside every module. If you set the parameter value to `yes` or `NoExprInPort-ML`, then *NoExprInPort-ML* Tag reports violation for logic expressions, which are present in port connections of the instances inside the top module only.

By default, the *checkGateInst* parameter is set to `no` and the Tag reports violations for logic expressions that are present as port connections of user module instances. However, when instance is a Verilog primitive, the Tag skips reporting violation for logic expressions that are present in port connections. Set this parameter `yes` to include instances of Verilog primitives also for reporting violations.

Message Details

The following message appears at the location of a module instance where an expression is used in port connection:

Expressions used in port connection in module instantiation

Severity

Warning

Examples

Example 1:

Some of the examples flagged by the NoExprInPort-ML Tag are as follows:

```
m1 inst1 (.in(a + b), ...);
m1 inst2 (.in(a & b), ...);
m1 inst3 (.in(a >> 1), ...);
```

Some of the examples not flagged by the NoExprInPort-ML Tag are as follows:

```
m1 inst1 (.in({a,b}), ...);
m1 inst2 (.in({a,c,a}), ...);
m1 inst3 (.in({a + b, a}), ...);
m1 inst4 (.in({a & b, b}), ...);
```

Example 2:

Consider the following example where the Tag does not report violation on "b1+b2" because master of inst is a Verilog primitive (and) and the [checkGateInst](#) parameter is not set:

```
module test;
  wire a,b1,b2;
  wire c;
  and inst(c, a, b1+b2); //No warning
endmodule
```

Example 3:

Consider the following example where the Tag reports violation on "b1+b2" as the [checkGateInst](#) parameter is not set to yes:

```
module test;
  wire a,b1,b2;
  wire c;
  and inst(c, a, b1+b2); //warning
endmodule
```

NoGenLabel-ML

Unnamed generate block or duplicate generate block names used in module

Language

Verilog

Description

The *NoGenLabel-ML* Tag reports violations, if the block label is missing or duplicate labels are used for generate blocks. If you do not specify designated labels, instance names become dependent on the interpretation of specific tools, which can cause discrepancy issues. Naming a generate block simplifies equivalence checking and constraints application because it makes names used in the synthesized netlist predictable. Therefore, it is recommended to use unique labels for each generate block.

Use the following parameters to customize the Tag behavior:

- ***check_dup_label***: By default, the Tag does not report any duplicate generate labels. Set this parameter to *yes* to report the generate blocks with duplicate labels.
- ***checkInstName***: Default value is *" "*. Set the value of the parameter to a regular expression to enable the NoGenLabel-ML tag to report violation for generate block whose name matches the value specified using the parameter.
- ***check_nested_decl***: By default, the parameter is set to *no*. Set this parameter to *yes* to consider instances and declarations in nested generate blocks as part of parent generate blocks and report violations if the parent generate blocks do not have a name.
- ***genlabeltype***: By default, the Tag checks for *generate-if*, *generate-for*, and *generate-case* types of generate blocks. Use this parameter to check for specific types of generate blocks.
- ***strict***: By default, the Tag reports only those unnamed generate blocks that contain declaration or instantiation. Set this parameter to *yes* to report all unnamed generate blocks.
- ***checkInstName***: By default,

Tag Exceptions

The *NoGenLabel-ML* Tag does not report violations for:

- duplicate labels of same `if-else` branch
- duplicate labels of `case` items within same `case` statements

Refer to [Example 4](#) for an illustration of these exceptions.

Message Details

The following message appears at the location of the generate block with a duplicate block label:

Duplicate label <block_label> assigned within the same module scope in module <module_name>

In addition, the following message appears at the location of the generate block with a missing block label:

Block label missing for <gen_type> block in module <module_name>

Where, <gen_type> can assume one of the following values:

- generate
- generate-if
- generate-for
- generate-case

Severity

Warning

Examples

Example 1

The following example demonstrates the behavior of the *NoGenLabel-ML* Tag when the [genlabeltype](#) parameter is set to `all` and the [check_dup_label](#) parameter is set to `no`:

```
module top(input a_in, b_in, output data_out);
    generate
        begin                                //violation by strict is set
```

```

    genvar g5;
    if (1)
    begin
        //violation by default
        for (g5 =0; g5 < 3; g5=g5+1)
        begin
            //violation when strict is set
            end
            a inst0 (.a(a_in),.b(b_in),.z(data_out));
        end
    end
endgenerate
endmodule

```

In the above example, by default, the Tag reports a violation only for the generate-if block because instantiation or declaration is present inside this block.

Example 2

The following example demonstrates the behavior of the *NoGenLabel-ML* Tag when the *genlabeltype* parameter is set to all and the *check_dup_label* parameter is set to yes:

```

module top(input a_in, b_in, output data_out);
generate
    begin: label_1
    genvar g5;
    if (3 > 0)
        begin: label_1 //violation when strict is set
            for (g5 =0; g5 < 3; g5=g5+1)
            begin: label_1 //violation by default
                a inst0 (.a(a_in),.b(b_in),.z(data_out));
            end
        end
    end
endgenerate
endmodule

```

In the above example, by default, the Tag reports a violation for duplicate label only for the generate-for block because instantiation or declaration is present inside this block.

Example 3

The following example demonstrates the behavior of the *NoGenLabel-ML* Tag when the *genlabeltype* parameter is set to *generate-if,generate-for*; the *strict* parameter is set to *yes*; and the *check_dup_label* parameter is set to *yes*:

```
module top(input a_in, b_in, output data_out);
  generate
    begin: label_1
      genvar g5;
      if (3 > 0)
        begin: label_1 //no violation for duplicate label
          for (g5 =0; g5 < 3; g5=g5+1)
            begin: label_1 //violation for duplicate label
          end
          a inst0 (.a(a_in),.b(b_in),.z(data_out));
        end
      end
    endgenerate
  endmodule
```

Example 4

The following example demonstrates the behavior of the *NoGenLabel-ML* Tag when the *genlabeltype* parameter is set to *all*, the *strict* parameter is set to *yes*, and the *check_dup_label* parameter is set to *yes*:

```
module example (in1, out1);
  parameter SEL = 2'b0;
  input [3:0] in1;
  output out1;

  generate
    begin: LBL
      if (SEL == 2'b1)
        begin : SELECT
          assign out1 = in1[1];
        end
      else
        begin : SELECT
```

```
                //no violation for duplicate label SELECT
case (SEL)
    2'd0 : begin : OUT assign out1 = in1[0]; end
    2'd1 : begin : OUT assign out1 = in1[1]; end
            //no violation for duplicate label OUT
    2'd2 : begin : OUT assign out1 = in1[2]; end
            //no violation for duplicate label OUT

    2'd3 : begin : OUT assign out1 = in1[3]; end
            //no violation for duplicate label OUT
endcase
end
end
endgenerate
endmodule
```

NoInoutPort-ML

Do not use bi-directional ports

Language

Verilog, VHDL

Description

The NoInoutPort-ML Tag reports inout ports used in the design.

By default, the NoInoutPort-ML Tag flags inout ports at different levels of hierarchy based on the [checkinalldu](#) and [inoutlevel](#) Tag parameters as follows:

checkinalldu	inoutlevel	Checking scope
set	set or not set	All design units
not set	not set	Only top-level design units
not set	set	top-level and design units at specified hierarchy level

NOTE: *The NoInoutPort-ML Tag is switched off by default.*

Inout ports have to be assigned Z's when you are not writing to them. Otherwise, you would not be able to read anything put into them from the other side.

Perl regular expressions can be specified using the [ignore_inoutport_expr](#) Tag parameter to ignore the matching IO ports.

When the ignore_inoutport_expr Tag parameter is set, it checks for ports at all levels of hierarchy and the values specified using the inoutlevel and checkinalldu Tag parameters are ignored.

Message Details

Verilog

The following message appears at the location where an inout port `<port-name>` is declared in module `<module-name>`:

Inout port '`<port-name>`' exists in module '`<module-name>`'

VHDL

The following message appears at the location where an inout port `<port-name>` is declared in entity `<entity-name>`:

Inout port '`<port-name>`' exists in entity '`<entity-name>`'

Severity

Warning

NonConstReset-ML

Signals must be reset to constant values.

Language

Verilog, VHDL

Description

The NonConstReset-ML Tag flags signals that are not assigned to constant values under set/reset condition.

Consider the following example where signal `q` is being set to a non-constant value (`a`) under the reset condition:

```
...
process( clk, reset, a )
begin
    if ( reset='1' ) then
        q <= a;
    elsif( clk='1' and clk'event ) then
        q <= d;
    end if;
end process;
...
```

Verilog

Signals must be reset to constant values to avoid undefined states during reset or set mode.

The NonConstReset-ML Tag performs checking based on the first reset/set condition.

The NonConstReset-ML Tag reports violation on:

- The first reset/set condition (asynchronous or synchronous) if the RHS of the assignment is non-static.
- The next asynchronous reset/set condition if and only if the first set/reset condition is also asynchronous.

If the first reset/set is asynchronous reset/set and second `else` condition is synchronous (that is, not present in the sensitivity list), then this condition is ignored. Also, no further Tag checking is performed on the block.

However, if the first set/reset condition is synchronous then further set/reset conditions will be ignored.

When the *ignore_sync_reset* parameter is set, the NonConstReset-ML Tag reports violation for signals under asynchronous set/resets condition only and no Tag checking is done for synchronous set/resets condition.

Also, when the *report_non_const_read* parameter is set, the *NonConstReset-ML* Tag reports non-constant variable read in if/case conditions inside the reset block.

Consider the following (Verilog) examples:

Example 1:

```
always @(posedge clk or posedge a)
begin
  if(b)
    q<=data1;
  else if(a == 1'b0)
    q<=data2; // this is asynchronous condition and
               the first condition is synchronous
  else
    q<=data3;
end
```

Example 2:

```
always @(posedge clk or posedge a)
begin
  if(a)
    q<=data1;
  else if(b == 1'b0)
    q<=data2; // this is synchronous condition and
               the first condition is asynchronous
  else
    q<=data3;
end
```

Example 3:

```
always @(posedge clk or posedge a or negedge b)
```


MoreLint Tags

```

begin
  if(a)
    q<=data1;
  else if(b == 1'b0)
    q<=data2; // this is asynchronous condition and
               the first condition is also
               asynchronous
  else
    q<=data3;
end

```

Example 4:

Consider the following example where the value of the *ignore_sync_reset* parameter is set to yes:

```

always@(posedge clk_ff or negedge rst_n)
  begin : pro_ff2
    if (data_width == 1'b0)
      q_q <= y_q ; //No violation, synchronous condition
    else if(rst_n == 1'b1)
      q_q <= z_q; //Violation, asynchronous condition
    else if(rst_n == 1'b0)
      q_q <= d; //Violation, asynchronous condition
    else if (data_width == 1'b1)
      q_q <= y_q ; //No violation, synchronous condition
    else if(rst_n == 1'b1)
      q_q <= d; //Violation, asynchronous condition
    else
      q_q <= d;
  end // block: ff2

```

VHDL

In VHDL, the NonConstReset-ML Tag does not check for synchronous reset/set.

Consider the following example:

```

library ieee;

```

```
use ieee.std_logic_1164.all

process(Clk, SET)
BEGIN
  IF( SET = '1') THEN
    q <= data1;
  ELSIF (Clk'event and Clk='1') THEN
    IF (RESET = '0') THEN
      q <= data1;
    ELSE
      q <= data1;
    END IF;
  END IF;
end process P1;
```

Message Details

Verilog

Message 1

The following message appears at the location where a signal is being assigned a non-constant value under the reset condition:

Invalid reset condition, RHS should be explicit static value and not driven by signal/variable <RHS Expression> [Hierarchy: <hierarchy>]

Message 2

The following message appears at the location where a non-constant variable is read in if/case conditions inside the reset block:

Invalid reset condition, non-static variable read inside reset [Hierarchy: <hier-path>]

Invalid reset condition, non-static variable read inside reset <Expression> [Hierarchy: <hierarchy>]

VHDL

The following message appears at the location where a signal is being assigned a non-constant value under the reset condition:

Invalid reset condition, RHS should be explicit static value

MoreLint Tags

and not driven by signal/variable <RHS Expression> [Hierarchy: <hierarchy>]

Severity

Warning

Suggested Fix

The signals should be reset to constant values.

NonStaticMacro-ML

Macro evaluates to non-static value

Language

Verilog

Description

The *NonStaticMacro-ML* Tag flags macros that are evaluated to non-static values.

Message Details

The following message appears at the location where a macro *<name>* is being assigned a non-static value:

Macro (<name>) evaluates to non-static value

Severity

Warning

Suggested Fix

Use static macros in design.

NonWireSignal-ML

The signal used in a design should preferably be a wire net.

Language

Verilog

Description

The NonWireSignal-ML Tag flags non-wire signals like `wand` and `wor` used in the design.

Message Details

The following message appears at the location where a non-wire signal `<name>` is defined:

```
Non-wire signal '<name>' used in design
```

Severity

Warning

NoOthersInAsgn-ML

Flag other constructs in assignment for slice of a vector

Language

VHDL

Description

The NoOthersInAsgn-ML Tag flags others constructs when used in assignment for slice of a vector.

Consider the following example where a slice of vector variable var1 is being assigned using an others construct:

```
library ieee;
use ieee.std_logic_1164.all

...
variable var1 : std_logic_vector (3 downto 0);
...
var1(1 downto 0) := ( others => '1' );
...
```

Message Details

The following message appears at the location where an OTHERS construct is being used in assignment to a vector slice:

others used in assignment for slice of a vector

Severity

Warning

NoParamMultConcat-ML

Do not use parameter as left operand in multiple concatenation expression

Language

Verilog

Description

The NoParamMultConcat-ML Tag flags parameters used as a left operand in a multiple concatenation expression.

If a parameter is used as the left operand in a multiple concatenation expression, its values might be overwritten for different elaborated modules. This may cause the width mismatch during assignments. For example, consider the following example:

```
...
parameter P = 2;
assign out = {P{1'b1}};
...
```

The above assignment statement may cause problems when the parameter P is passed different values from the top-level for different instances.

Message Details

The following message appears where a parameter *<param-name>* is used as left operand in a multiple concatenation expression:

Parameter '<param-name>' used in the multiple concatenation expression

Severity

Warning

Examples

Consider the following example flagged by the NoParamMultConcat-ML Tag where parameter P1 is used as the left operand:

```
module test;
```

```
integer count;  
parameter P1 = 3;  
initial  
  for(count = { P1{2'b0}} ; count>{P1{2'b1}} ;  
        count = count + 1 )  
    $display("Count = %d", count);  
endmodule
```


NoRealFunc-ML

Functions of type real are not synthesizable

Language

Verilog

Description

The NoRealFunc-ML Tag flags functions that return real values.

Function return values of type real may not be meaningful from design point of view and may not be supported by some synthesis tools.

Message Details

The following message appears at the first line of a function description
<func-name> that returns a real value:

Function '<func-name>' of type real used

Severity

Warning

Examples

The NoRealFunc-ML Tag flags function declaration of the following format:

```
function real <func-name>;
```

NoSigCaseX-ML

Do not use signals in casex and casez constructs

When to Use

Use this Tag to identify signals that are used in casex and casez constructs.

Description

The NoSigCaseX-ML Tag flags signals used in casex and casez constructs.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the first line of a casex/casez construct where signal *<sig-name>* is used:

[WARNING] Signal '*<sig-name>*' used in casex/casez item

Potential Issues

Violation may arise when a casex or casez construct is used in a signal.

Consequences of Not Fixing

Such use of signals may result in unpredictable behavior.

How to Debug and Fix

Select the message that you want to debug.

In the HDL window, it takes you to the line where reported signal has been used as casex/casez item.

To fix the violation, review the code and ensure that the signals are not used in the `casex` and `casez` constructs.

Example Code and/or Schematic

The NoSigCaseX-ML Tag flags the following example where signals `a_1`, `b_1[3]`, `i`, and `j` are used in `casex` construct items:

```
...
reg data_out;
reg clk, sel;
reg[4:0] a_1, b_1;
wire i, j;

always @(posedge clk)
  casex (sel)
    a_1: data_out <= 2'b01;
    {b_1[3], b_1[2]}: data_out <= 2'b11;
    i: data_out <= 2'b10;
    j: data_out <= 2'b00;
    default : data_out <= 2'b0x;
  endcase
...
```

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related report or files.

NoStrengthInput-ML

Synthesis tools ignore strength inputs

Language

Verilog

Description

The NoStrengthInput-ML Tag flags strength specifications.

While strength inputs are allowed in RTL description, they are ignored during synthesis. Hence, you should ensure that your design does not depend on strength values to achieve correct functionality since this behavior will not necessarily map into post-synthesis behavior.

Message Details

The following message appears at the location where strength specification is used:

Strength input used. Synthesis tools ignore strength inputs

Severity

Warning

Examples

The NoStrengthInput-ML Tag flags strength specifications as in the following examples:

```
assign (strong1, pull0) mynet = in;  
AND (strong1, pull0) and1(out, in);  
wire (strong1, pull0) mynet = in;  
triereg (medium) reg1, reg2;
```

NoWidthInBasedNum-ML

Width should be specified for all based numbers

Language

Verilog

Description

The NoWidthInBasedNum-ML Tag flags based numbers without width specification.

By default, based numbers without width specification are assumed to be 32 bit wide. This inference may be different from the actual width and may lead to errors.

Parameters

- *min_based_width*: Default value is zero and reports violation for all based numbers. When you set this parameter to a positive integer value, the *NoWidthInBasedNum-ML* Tag reports violation for those based numbers whose width is greater than the value set for the *min_based_width* parameter.
- *ignore_single_zero*: Default value is no. Set this parameter to yes to enable the *NoWidthInBasedNum-ML* Tag to ignore violations for width unspecified based numbers with single zero.

Message Details

The following message appears at the location where a based number `<num>` that has no width specification, is encountered:

```
width specification missing for based number '<num>'
```

Severity

Warning

Examples

In the following example, signals `out` and `out_task` are assigned based numbers without width specifications:

```
out = 'b0;  
out_task = 'h20;
```

To fix the message, specify the width as follows:

```
out = 1'b0;  
out_task = 6'h20;
```

NoXInCase-ML

Ensure that case expression and case choices do not have 'X'

When to Use

Use this Tag to identify case expressions and case choice that contain don't care value, x.

Description

The *NoXInCase-ML* Tag reports case construct select expressions and case construct clause labels that have don't care value, X.

Tag Exceptions

The *NoXInCase-ML* Tag does not report a violation when case construct select expressions and case construct clause labels are defined in a package.

Default Weight

5

Language

VHDL

Parameter(s)

None

Constraint(s)

None

Message Details

The following message appears at the location of a case construct select expression or a case construct clause label that has X:

[WARNING] Case expression <label> contains X

Potential Issues

Violation may arise when a case expression and a case choice contains don't care value, X.

Consequences of Not Fixing

The X values are ignored by synthesis tools, which may cause pre-synthesis and post-synthesis simulation mismatch.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the statement where 'X' is used in a case statement. The value, X, is evaluated from one of the following sources:

- Direct hard-coded X assignment
- Constant or generic
- Function call, which returns X
- Expression which evaluates to X

To fix the violation, remove the don't care value, X, from the case statement.

Example Code and/or Schematic

Consider the following example:

```
case std_logic_vector("1X0") is
  when "101" => int1 <= int2;
  when "110"|"10X" => int1 <= int2 + 10;
  when others => null;
end case;
```

In the above example the Tag reports a violation as case construct `std_logic_vector` contains x.

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

NullOthers-ML

Others clause of a case statement should not have null statement

Language

VHDL

Description

The NullOthers-ML Tag flags others clause of a case statement that has only null statement.

This ensures complete coverage of case values.

Message Details

The following message appears at the location of a others clause of case statement that has a null statement:

others clause has only null statement

Severity

Warning

Examples

The NullOthers-ML Tag flags the following example where the others clause of the case construct that has a null statement:

```
...
case in2 is
...
    when others => null;
end case;
...
```

NullPort-ML

Null Ports should not be defined

Language

Verilog

Description

The NullPort-ML Tag flags a non-existent port.

Each port should have a unique name. While null ports are allowed in Verilog, using them in mixed-language designs causes errors as there is no equivalent counterpart in VHDL.

Message Details

The following message appears at the location of a non-existent port:
Null Port is defined

Severity

Guideline

Examples

The NullPort-ML Tag flags the following example where a non-existent port is declared:

```
module test(q_in, , q_out);  
    input  q_in;  
    output q_out;  
    ...  
endmodule
```

OneModule-ML

At most one module per file

Language

Verilog

Description

The *OneModule-ML* Tag flags the names of the files that describe more than one module.

A file must contain at most one module description. This simplifies design modifications.

Parameter(s)

[check_all_modules](#): By default, the parameter is set to no. Set the parameter to yes to enable the tag to report a violation for unused modules.

Message Details

The following message appears when a file <file-name> has more than one module described:

File '<file-name>' contains more than one module

Severity

Warning

ParamOverrideMismatch-ML

Mismatch in the number of parameter over-rides and number of parameters in the instantiated module

Language

Verilog

Description

The ParamOverrideMismatch-ML Tag flags those instances where the number of parameters specified in an instance parameter override does not match the number of parameters defined within the instantiated module.

The ParamOverrideMismatch-ML Tag ignores cases where the instance does not contain any parameter override.

By default, the ParamOverrideMismatch-ML Tag ignores instances where parameter is overridden through named association. Set the value of the [check_param_association](#) parameter to *yes* or *ParamOverrideMismatch-ML* to check for such instances. Consider the following example

```
module test();  
  parameter a = 2;  
  parameter b = 2;  
endmodule  
module top();  
  test #(.a(4)) inst();  
endmodule
```

The *ParamOverrideMismatch-ML* Tag reports violation for the above example when the value of the `check_param_association` parameter is set to *yes*.

Set the [check_conversion_function](#) Tag parameter to *yes* to consider the cases where the parameters are declared in v2k style, that is, parameters are declared in the module header.

NOTE: The *ParamOverrideMismatch-ML* Tag supports *generate-for block*.

Message Details

The following message appears at the location where Tag-violating instance *<inst-name>* is encountered:

MoreLint Tags

Mismatch in number of parameter over-rides and number of parameters in the instantiated module in instance '<inst-name>'

Severity

Warning

Examples

Consider the following example:

```
module top (Reset_N, Clk);
    input Reset_N;
    input Clk;
    one #(20,5) o1 (.Reset_N (Reset_N), .Clk(Clk));
endmodule

module one (Reset_N, Clk);
    input Reset_N, Clk;

    reg myreg;
    parameter CPS = 29;
    parameter MCW = 6;
    parameter MCR = 6;

    always @(posedge Clk or negedge Reset_N)
        if (~Reset_N) myreg <= 0;
        else myreg <= !myreg;

endmodule
```

Here, the number of parameter overrides (2) in the instantiation o1 is not same the number of parameters declared (3) in the master module one.

ParamWidthMismatch-ML

Parameter width does not match with the value assigned

Language

Verilog

Description

The ParamWidthMismatch-ML Tag flags parameter assignments where the parameter and the assigned value have different bit-width.

Such assignments are not a problem with synthesis or simulation as a parameter with no type and range specification always holds the type and range of the assigned value. Hence, no truncation or extension of bits occur.

A parameter with a range specification will be in the range of the parameter declaration and will be unsigned. The sign and range in such case is not affected by value overrides.

By default, the range of a signed parameter will be specified by the range of the final value assigned to the parameter after any value overrides have been applied.

The ParamWidthMismatch-ML Tag will report violation if the width range of a parameter is specified and the width of the final value assigned to the parameter is greater than the width of the parameter.

Message Details

The following message appears at the location where a parameter `<name>` of bit-width `<widthp>` is assigned a value of bit-width `<widthv>`:

Parameter '`<name>`' width (`<widthp>`) does not match width (`<widthv>`) of value assigned

Severity

Warning

Examples

Consider the following examples where a violation will be reported if the width range of a parameter is specified and the width is smaller than the

MoreLint Tags

constant value.

```
parameter p1 = 20
```

```
parameter [3:0] p1 = 20 //LHS:4, RHS:5
```

```
parameter [3:0] p1 = 2 //LHS:4, RHS:2
```

```
parameter [3:0] p3 = 7'b1; //LHS:4, RHS:7 (Natural width is 1) //
```

```
parameter [3:0] p4 = 2'b1 // LHS:4, RHS:2 (Natural width is 1) //
```

PartConnPort-ML

Port is unconnected or partially connected

Language

Verilog

Description

The PartConnPort-ML Tag flags unconnected or partially connected ports in module instances.

When a port of a module instance is not connected or only partly connected in the top-level module, the unconnected bits are driven to unknown value (for input ports) or are ignored (for output ports). Thus, the design may not function as expected.

By default [report_open_port](#) parameter is set to `no` and the Tag does not report violations for open terminal connections. Set the `report_open_port` parameter `yes` to enable the *PartConnPort-ML* Tag to report violations for open terminal connections.

Message Details

The following message appears at the location of an instance `<inst-name>` where port `<port-name>` is unconnected or partly connected:

Port '`<port-name>`' is unconnected or partially connected in instance '`<inst-name>`' [Hierarchy: '`<hier-path>`']

Severity

Warning

Examples

Consider the following example where the ports of instance `inst1` of module `lower` are unconnected:

```
module lower (in, out);  
  input in;  
  output out;
```


MoreLint Tags

```
endmodule

module top(inp1,out1);
    input inp1;
    output out1;

    lower inst1 ();
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Port 'in' is unconnected or partially connected in instance 'inst1'

Port 'out' is unconnected or partially connected in instance 'inst1'

PartSelectRange-ML

Vector part select is probably wrongly written, use '+' instead of ':'

Language

Verilog

Description

The PartSelectRange-ML Tag flags vector part selects that are not specified as per LRM.

The PartSelectRange-ML Tag reports a violation if a colon (:) is immediately followed by a plus (+) or a minus (-).

For example, if you want to select bits 50 to 57 from a vector, a, then you should specify the same by writing a[50+:8]. However, if you write a[50+:8] then this will be interpreted as a[50:(+8)]. Hence, you will get a[50:8] instead of a[50:57].

The PartSelectRange-ML Tag reports violation only if there is no other character between : and +. For example, no violation is reported in the following cases:

- temp[150: +50]
- temp[150 : (+50-9)]

Message Details

The following message appears at the location where a vector part select *<vector-with-range>* is written incorrectly:

Vector part select '*<vector-with-range>*' is probably wrongly written

Severity

Warning

Examples

The PartSelectRange-ML Tag reports a violation, if a colon (:) is

immediately followed by + or - , as shown in the following examples:

```
adder[50:+8]
```

```
adder[50:-8]
```

Also, the PartSelectRange-ML Tag reports a violation only if there is no other character between : and +/- . For example, the Tag does not flag a violation in the following cases:

```
adder[50: -P + Q]
```

```
adder[50: P - Q]
```

```
adder[50: (P - Q)]
```

PragmaComments-ML

Detects Pragma Comments

When to Use

Use this Tag to identify pragma comments for the user-specified pragmas.

Description

The PragmaComments-ML Tag reports violation for the pragma comments for the user-supplied pragmas.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

- [attach_violation_tag](#): The default value is no. Set the value of the parameter to yes to attach signature (tag) to the violation message to make violation message unique.
- [pragma_list_ml](#): The default value is synopsys in case of Verilog and default pragma synopsys or pragma in case of VHDL. Specify a comma-separated list of pragmas to find out comments and their presence in RTL for the given pragmas.

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following violation message appears, if the user-specified pragma is found before the module start or on the same line of the module definition, this pragma shall be reported with its comment string as:

[INFO] pragma: (module: module name) pragma comment

Potential Issues

Violation may arise when pragma comments are used for the user-specified pragmas in a module.

Consequences of Not Fixing

Since the violation message is an informational message, it does not have a consequence.

How to Debug and Fix

Double-click the violation message. The HDL viewer window highlights the line, where pragma comment is used. By default, the pragmas reported are the standard Verilog and VHDL pragmas. The detected pragma can also be from the pragma list specified through the [pragma_list_ml](#) parameter.

To fix the violation, remove the pragma comment from the user-specified pragma.

Message 2

The following violation message appears, if the pragma is found a line above the instance or on the same line:

[INFO] pragma: (instance: instance name) pragma comment

Potential Issues

Violation may arise when pragma comments are used for the user-specified pragmas in an instance.

Consequences of Not Fixing

Since the violation message is an informational message, it does not have a consequence.

How to Debug and Fix

Double-click the violation message. The HDL viewer window highlights the line, where pragma comment is used. By default, the pragmas reported are the standard Verilog and VHDL pragmas. The detected pragma can also be from the pragma list specified through the [pragma_list_ml](#) parameter.

To fix the violation, remove the pragma comment from the user-specified pragma.

Message 3

The following violation message appears, if the pragma is found elsewhere in the code:

[INFO] pragma: (RTL_CODE) pragma comment

Potential Issues

Violation may arise when pragma comments are used for the user-specified pragmas in an RTL code.

Consequences of Not Fixing

Since the violation message is an informational message, it does not have a consequence.

How to Debug and Fix

Double-click the violation message. The HDL viewer window highlights the line, where pragma comment is used. By default, the pragmas reported are the standard Verilog and VHDL pragmas. The detected pragma can also be from the pragma list specified through the [pragma_list_ml](#) parameter.

To fix the violation, remove the pragma comment from the user-specified pragma.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

RegInput-ML

Module input port should be registered

Description

The RegInput-ML tag flags nets connected to input ports that are not registered after entering the module boundary.

For the input side, this tag checks whether all the nets connected to the input port are registered or not inside the module.

The RegInput-ML tag flags require that the nets connected to input ports should be connected to a flip-flop data pin within the module.

By default, the RegInput-ML tag ignores all buffers and inverters while checking for the connected flip-flop. Use the [buf_count](#) and [inv_count](#) tag parameters to specify your own number of buffers and inverters to be ignored.

The RegInput-ML tag ignores those nets connected to inputs ports when these nets are used as clock signals or reset signals of flip-flops.

When a net connected to an input port has multiple fan-outs, each fan-out must be registered or must be used as clock signal or reset signal of a flip-flop.

Use the [depth_ml](#) parameter to specify the level of hierarchy to which tag is checked. By default, a check is run for all hierarchy levels. If the [depth_ml](#) parameter is set to 1, then the checking is done only for top modules.

Use the [ignore_inout](#) parameter to `yes` to not report any violation for the inout terminals.

Message Details

The following message appears at the location where input port <port-name> of design unit <du-name> is declared when the net connected to the input port is either not registered or is used as the clock signal or the reset signal of a flip-flop:

Input port '<port-name>' is not registered [module:<du-name>]

Severity

Warning

ReportPortInfo-ML

Generate a report of all ports of top level block and black box instances

When to Use

Use this Tag to generate a report of all ports of top level block and black box instances.

Description

The `ReportPortInfo` Tag generates the [ReportPortInfo Report](#), which dumps the following information about all the top-level modules/entities and ports of black box instances:

- Port Name
- Port Direction
- Port width including left and right index
- Port comment, if any. This is not applicable for black box instances
- Comment source, that is, in-line comment on Verilog module ports or port definitions (input and output), and in-line comment on VHDL entity ports

In case of a multidimensional array, all indexes are displayed. For example, consider a port, `reg [7:0] array [1:0] [2:0] [3:0]`. So, for this port, the following index is displayed:

```
[1:0] [2:0] [3:0] [7:0]
```

In case of a structure/record the complete size of the structure is dumped. For example, consider the following structure:

```
typedef struct {
    bit [7:0] opcode;
    bit [23:0] addr;
}IR;
IR str[7:0];
```

For the above structure, the following index is displayed:

```
((8+24) [7:0])
```


Default Weight

2

Language

Verilog, VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when a report is generated for all ports of top-level blocks and black box instances:

[INFO] Port Information for top design unit has been generated. For details see report ReportPortInfo.rpt;

Potential Issues

The ReportPortInfo-ML Tag generates an informational message when a report is generated for all ports of top-level blocks and black box instances.

Consequences of Not Fixing

Since the violation message is an informational message, it does not have a consequence.

How to Debug and Fix

The ReportPortInfo-ML Tag generates the [ReportPortInfo Report](#). To view the report, perform one of the following steps:

- Double-click the violation message. The ReportPortInfo Report is displayed in the HDL viewer window.
- Generate the report by selecting the [ReportPortInfo Report](#) option. The report is displayed in an external window.

Verify the port information displayed in the ReportPortInfo report.

No fix is required as this is an informational message.

Example Code and/or Schematic

Verilog

Consider the following example:

```
module top( input a, // input for AND gate
input b, // input for AND gate
output z); // output port

and ins(a,b,z);
endmodule
```

In the above example the ReportPortInfo-ML Tag generates a report of all ports and top-level modules for the black box instance.

VHDL

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

entity top is
port(
    d : in bit; --data input port
    clk, rst,srst_abc_n,srst_f : in bit;  --control signals
input port
    q : out bit  --output port
);
end top;

architecture arc of top is
begin
process(clk, rst)
begin
    if (rst = '0') then
        q <= '0';
    elsif (clk'event and clk = '1') then
        if (srst_abc_n = '0') then
            q <= '1';
        else
```

MoreLint Tags

```
        q <= d;  
    end if;  
end if;  
end process;  
end arc;
```

In the above example, the ReportPortInfo-ML Tag generates a report of all top-level modules and ports for the black box instance.

Default Severity Label

Info

Tag Group

Not Applicable

Reports and Related Files

[*ReportPortInfo Report*](#)

ReserveNameSystemVerilog-ML

SystemVerilog IEEE-1800 reserved word used

Language

Verilog, VHDL

Description

Verilog

The *ReserveNameSystemVerilog-ML* Tag reports the following SystemVerilog IEEE-1800 reserved words that are used as object names:

alias	always_comb	always_ff	always_latch	assert
assume	before	bind	bins	binsof
bit	break	byte	chandle	class
clocking	const	constraint	context	continue
cover	covergroup	coverpoint	cross	dist
do	endclass	endclocking	endgroup	endinterface
endpackage	endprogram	endproperty	endsequence	enum
expect	export	extends	extern	final
firstmatch	foreach	forkjoin	iff	ignore_bins
illegal_bins	import	inside	int	interface
intersect	join_any	join_none	local	logic
longint	matches	modport	new	null
package	packed	priority	program	property
protected	pure	rand	randc	randcase
randsequence	ref	return	sequence	shortint
shortreal	solve	static	string	struct
super	Tagged	this	throughout	timeprecision
timeunit	type	typedef	union	unique

MoreLint Tags

var	virtual	void	wait_order	wildcard
with	within	uwire		

VHDL

The *ReserveNameSystemVerilog-ML* Tag reports the following SystemVerilog IEEE-1800 reserved words that are used as object names:

always_comb	always_ff	always_latch	assert_strobe	break
byte	const	continue	coverpoint	cross
dist	do	endclass	endclocking	endgroup
endinterface	endpackage	endprogram	endproperty	endsequence
enum	expect	export	extern	forkjoin
iff	import	int	interface	join_none
local	logic	longint	matches	modport
packed	priority	program	property	protected
rand	randc	randcase	randsequence	ref
sequence	shortint	shortreal	solve	static
struct	super	Tagged	this	throughout
timeprecision	timeunit	typedef	union	unique
var	virtual	void	wait_order	wildcard
within	bit	string		

The Tag does not check the following SystemVerilog IEEE-1800 reserved words because these are reserved words in VHDL itself:

alias	assert	new	null	package	pure
return	with	type			

Message Details

The following message appears at the location where the IEEE-1800 reserved word *<keyword>* is used as an object name:

SystemVerilog IEEE-1800 reserved word '<keyword>' used

Severity

Info

ReserveNameV2K-ML

Verilog 2001 reserved word used

Language

Verilog, VHDL

Description

The ReserveNameV2K-ML Tag flags Verilog 2001 reserved keywords used in the design.

NOTE: *Since by default V2K syntax are supported in VC SpyGlass Lint, the ReserveNameV2K-ML Tag reports violation if the set_option disableV2K yes command is executed and a V2K keyword is present in the design.*

The ReserveNameV2K-ML Tag flags the following Verilog 2001 reserved keywords:

automatic	config	endconfig
endgenerate	generate	genvar
liblist	library	localparam
noshowcancelled	pulsetype_oneevent	pulsetype_ondetect
showcancelled	cell	design
unsigned		

The following Verilog 2001 reserved keywords are not checked in VHDL designs as they are also VHDL reserved keywords:

generate	library	use	unsigned

Message Details

The following message appears at the location where a Verilog 2001 reserved keyword *<keyword>* is used:

verilog 2001 reserved word '<keyword>' used

Severity

Warning

SameDu-ML

Same design unit in Verilog and VHDL

Language

Verilog, VHDL

Description

The SameDu-ML Tag flags design units having same name in Verilog and VHDL designs.

A design unit in Verilog is a module and design unit in VHDL is an entity.

Since VHDL is case-insensitive, an entity with the name ABC in VHDL matches the module with name Abc in Verilog designs. Hence, the SameDu-ML Tag reports a violation.

The SameDu-ML Tag flags the following example as highlighted:

```

module abc(out1, in1)
...
...
endmodule

entity abc is
port (a : IN std_logic;
      q : OUT std_logic );
end abc;
```

Message Details

The following message appears at the location of the module whose name *<module-name>* is same as a VHDL entity name *<ent-name>* (in VHDL filename *<file-name>*, line number *<num>*):

Module with name '*<module-name>*' is also present in VHDL '*<ent-name>*' (Filename :*<file-name>*, Linenumber :*<num>*)

Severity

Info

SameLabelsInGenerate-ML

Flag violation when same labels are used inside different generate block

Language

VHDL

Description

The SameLabelsInGenerate-ML Tag flags same labels used in generate constructs creating different component instantiations.

Consider the following example where the same label `label_inside_generate` is used in two generate blocks that are creating different component instantiations (`comp_instance1` and `comp_instance2`):

```
library ieee;
use ieee.std_logic_1164.all

...
for i in 0 to 11 generate
  label_inside_generate:comp_instance1
end generate;

for i in 0 to 11 generate
  label_inside_generate:comp_instance2
end generate;
```

The SameLabelsInGenerate-ML Tag flags all cases of same label used while creating different component instantiations using generate constructs in a design unit. The SameLabelsInGenerate-ML Tag does not flag same label used while creating the same component instantiation in two generate constructs in a design unit.

NOTE: *The SameLabelsInGenerate-ML Tag supports generate-if and generate-for blocks.*

Message Details

The following message appears at the location where label `<label>` is used while creating component instantiations in a generate construct when

MoreLint Tags

the same label was already used at line *<num>* of file *<file-name>* while creating another set of component instantiations in a generate construct:

Label '*<label>*' is used more than once inside generate statement [Previous used at line No.: '*<num>*' (File Name: '*<file-name>*')]]

Severity

Warning

SameLoopIndexUsed-ML

Do not use same for-loop index variable inside different always @* blocks.

Language

Verilog

Description

The SameLoopIndexUsed-ML Tag flags same for-loop indexes is used in different always block and where the sensitivity list is V2K style (@* or @(*)).

The SameLoopIndexUsed-ML Tag checks only between those always blocks that have V2K style sensitivity lists.

If more than one always block sets and references a loop index variable, all blocks referencing that variable are triggered, and the design falls into an infinite loop.

The SameLoopIndexUsed-ML Tag flags the following example as highlighted:

```
always @*
  for (k=0; k<10; k=k+1)
    ...

always @*
  for (k=0; k<20; k=k+2)
    ...
```

Consider another example, where same for-loop index is used in different always blocks and the sensitivity list is not V2k style:

```
always @*
  for (k=0; k<10; k=k+1)
    ...

always @(a or b)
  for (k=0; k<20; k=k+2)
    ...
```

Message Details

The following message appears at the location where the same index `<idx-name>` in for-loop is used in multiple always blocks (previous use at line `<num>`, filename `<file-name>`):

Index '`<idx-name>`' in for-loop used in multiple always blocks
[Previous used at line No.: `<num>`, File Name: `<file-name>`]

Severity

Warning

Suggested Fix

The design should be rewritten to use local iteration variables that are declared inside each always block. Or, each block should use a separate globally declared iteration variable.

SelfAssignment-ML

Same operand on both sides of combinational assignment detected

Language

Verilog

Description

The SelfAssignment-ML Tag flags assignments in combinational and sequential always constructs where the same operand is used on the both sides.

By default, the SelfAssignment-ML Tag checks assignments in combinational always constructs only. Set the [checkallblocks](#) Tag parameter for the SelfAssignment-ML Tag to check assignments in both combinational and sequential always constructs.

By default, the *SelfAssignment-ML* Tag checks for all types of procedural blocks. If you set the [checkcontassign](#) parameter to `yes`, this Tag ignores all types of procedural blocks (`always` and `initial`) and the Tag reports violations for continuous assignment statements if the LHS of expression is of `net` type.

While such assignments may not be a problem for sequential circuits, they may result in combinational loops in combinational circuits. Such assignments in `for` loop constructs may create priority circuit with deeply nested dependency.

By default, the [ignoreforloop](#) parameter is set to `no` and the *SelfAssignment-ML* Tag reports violations for the self assignments inside the `for` loop. Set this parameter to `yes` to not report violations for such cases.

Message Details

The following message appears at the location of an assignment where the same operand `<oprnd>` is used on both sides of the assignment:

Same operand '`<oprnd>`' used on both sides of an assignment

Severity

Warning

Examples

Some examples of assignments flagged by the SelfAssignment-ML Tag are as follows where w1, wire2 is assigned on both sides:

```
assign w1 = w1 + in;  
assign {wire1,wire2} = in + wire2;  
out[0] <= out && in;
```

SelfDeterminedExpr-ML

Self determined expression present in the design.

Language

Verilog

Description

The SelfDeterminedExpr-ML Tag flags use of self determined arithmetic expressions present in the design.

NOTE: *The Tag does not consider the LHS width of an assignment while processing expressions that are defined as self-determined as per the LRM.*

If an arithmetic expression is used as an operand with any of the following operators, this Tag considers the complete expression as a self determined arithmetic expression:

- Unary operator: ~&, ~|, ~^, &, |, ^, !, ^
- Logical operator: &&, ||
- Magnitude operator: <=, >=, <, >
- Equivalence operator: ==, ===, !=, !==
- shift operator: >>, <<, **, >>>, <<<
- Conditional operator: i?j:k
- All concatenation and multi-concatenation expressions
- System functions \$signed() and \$unsigned()

The SelfDeterminedExpr-ML Tag flags if a self determined expression is used in any of the following constructs, and there is actually a loss in precision in self determined arithmetic expression:

- Assignment statements
- Expression in case-select, condition statement of if-elsif, while, for-loop
- Expression used in port connection during module instantiation
- Expression passed to task and functions as an argument

By default, the [handle_shift_in_concat](#) parameter is set to yes and the SelfDeterminedExpr-ML tag reports a violation for the shift operations in a concatenation. Set this parameter to no to enable the *SelfDeterminedExpr-*

ML tag to not report a violation for the shift operations in a concatenation.

By default, no violation is reported for truncation due to the left shift operator inside a `$signed/$unsigned` system call, when value becomes zero. Set the [handle_signed_truncation](#) parameter to `yes` to allow the *SelfDeterminedExpr-ML* Tag to report such violations. For example:

```
$signed(2'b11 << 2)
```

By default, no violation is reported if an argument type (arithmetic expression) does not match with `$signed/$unsigned` system call. Set the [strict](#) parameter to `yes` or *SelfDeterminedExpr-ML* to flag such violations. For example:

```
wire [3:0] a;
wire [3:0] b;
wire signed [3:0] c;
wire [4:0] div_signed5, div_signed6, div_signed7;
```

Here, `a, b` is unsigned. Therefore, operation `(a/b)` is always unsigned, which is not matching with the `$signed` system call.

By default, the *SelfDeterminedExpr-ML* Tag does not process expressions involving constants to check for violations. Set the value of the [handle_const_expressions](#) parameter to `yes` to enable the *SelfDeterminedExpr-ML* Tag to process expressions involving constants to check for violations.

Below code report violation when the value of the [strict](#) parameter is set to `yes`.

```
assign div_signed5 = $signed(a / b); //signed
//argument type, that is, (a/b): unsigned
```

Also, the Tag does not report violation for the following case:

```
assign div_signed6 = $signed(a / b); //unsigned
//argument type, that is, (a/b): unsigned
```

Apart from the above constructs, this Tag also reports a violation if there is a loss of precision after performing arithmetic operation. For example:

```
temp1=$signed(a/2'b11); //No Loss of precision
temp1=$signed(a-2'b11); // Loss of precision
temp1=$signed(a+1'b1); // Loss of precision
```

Loss in precision is considered when the natural width of the a self-

determined expression exceeds the LRM width of the same expression.

In case of assignments, loss in precision is considered only if the natural width of the self-determined expression exceeds the LRM width of the same expression and LHS width of the assignment. In case of *concat* element, the Tag does not check for the LHS width of the assignment. For example:

```
wire [1:0] a1,a2,a3;
assign out3 = {(a3+(a1+a2)), 3'b110};
assign out3 = {(a3+1)+a2, 3'b110};
```

The SelfDeterminedExpr-ML Tag does not report expressions involving constants. For example, no violation is reported in the following case:

```
parameter a1 = 18;
out1[8:0] = {(18 + 18 + 4), 3'b110};
out2[8:0] = {(a1 + 18) + 4, 3'b110};
```

Set the [check_lhs_rhs_assign_width](#) parameter to *yes* to not report violation for assignments where the LHS width is greater than the RHS width. By default, this parameter is set to *no*. See the following examples.

Example 1

Consider the following code snippet:

```
module test();

logic [1:0]l1;
logic [1:0]l2;
logic l3;

always_comb
begin
  l3 = (l1 * l2)<< 1; //Violaton
end
endmodule;
```

For this example, the *SelfDeterminedExpr-ML* Tag reports a violation because the LHS is smaller than the RHS of the assignment.

Example 2

Consider the following code snippet:

MoreLint Tags

```
module test();

logic [1:0]l1;
logic [1:0]l2;
logic [8:0]l3;

always_comb
begin
  l3 = (l1 * l2)<< 1; //No-Violation
end
endmodule;
```

For the above example, the *SelfDeterminedExpr-ML* Tag does not report violation because the LHS is greater than the RHS of the assignment.

Message Details

The following message appears for a Tag-violating expression *<expr>* found in module *<module-name>*:

self determined expression '*<expr>*' found in module '*<module-name>*' [Hierarchy: '*<hier-path>*']

Severity

Warning

SensListRepeat-ML

A signal is repeated multiple times in the sensitivity list

Language

Verilog, VHDL

Description

The SensListRepeat-ML Tag flags signals that are repeated in the sensitivity list of an always/process construct.

Repeating the same signal is redundant and is most likely a result of typographical error.

NOTE: *The SensListRepeat-ML Tag supports generate-if, generate-for, and generate-case blocks.*

Message Details

The following message appears at the location of a sensitivity list of the always/process construct where the signal is repeated:

Signal '<width>' (or some of its bits) repeated in the sensitivity list [Hierarchy: '<hier-path>']

Severity

Warning

Examples (Verilog)

In the following example, signal in1 repeated in the always construct is flagged by the SensListRepeat-ML Tag:

```
module mod_Test1 ( in1, in2, out1 );
  input in1, in2;
  output out1;

  reg out1;

  always @(in1 or in2 or in1)
  begin
    out1 = in1&in2;
  end
endmodule
```

```
    end  
endmodule
```

Examples (VHDL)

In the following example, signal `clk` repeated in the process construct is flagged by the SensListRepeat-ML Tag:

```
library ieee;  
use ieee.std_logic_1164.all  
  
entity ent is  
    port(d : in BIT; clk : in BIT_VECTOR(1 downto 0);  
          q : out BIT);  
end ent;  
  
architecture arc of ent is  
    begin  
        process(d,clk(0),clk)  
            begin  
                if (clk(1)'event and clk(1) = '1') then  
                    q <= d;  
                end if;  
            end process;  
        end arc;
```

SetBeforeRead-ML

Variable may not be set before being read (VHDL)
Signal is read before being set (Verilog)

Language

Verilog, VHDL

Description

Verilog

The [SetBeforeRead-ML](#) Tag reports those signals that are read before they are set using an assignment within the same `always` block.

The Tag does not report a violation if a signal is not set at all inside the `always` block. This Tag does not check for sequential `always` blocks, i.e., the Tag checking is done for pure combinational and latch inferred `always` blocks.

For conditional constructs, the signal is considered set only if it is set in all the branches of the conditional construct. In addition, two different `case/if-else` constructs are treated as different sets of assignments and a violation is reported if a signal is not set in all branches in one `case/if-else` block but read in any one branch in other `case/if-else` block.

The Tag reports a violation for the following cases:

- A signal is read and then set using blocking assignment.
- A signal, which is not present in the sensitivity list, is set using nonblocking assignment before or after it is read.

The Tag does not report a violation in the following cases:

- A signal is set before it is read using blocking assignment.
- A signal, which is present in the sensitivity list, is set using nonblocking assignment statement before or after it is read.

By default, the Tag does not check for output ports. Set the [check_output_ports](#) parameter to `yes` to check for output ports also. Here, the output ports should be set and read in the same `always` block. No violation is reported if an output port is set in other `always` block and

read in another `always` block.

In addition, when the [check_output_ports](#) parameter is set to `yes`, the Tag also checks for function return values and function/task output variables.

By default, the Tag also checks for non-port variables that are read within the scope of tasks or functions.

By default, the tag reports violation for multiple hierarchies. Set the value of the [report_only_from_one_hierarchy](#) parameter to `yes` to suppress the hierarchy field and report only one violation per assignment

By default, the *SetBeforeRead-ML* tag reports violations for signals for which the following conditions are true:

- Signal is read with non-static bit select expression.
- Some overlapping bits are set before the read statement.

Set the value of the [ignoreNonStaticOverlap](#) parameter to `yes` to not report violation for the signals that meet the above conditions.

By default, the value of the [avoid_nba_always_latch](#) parameter is set to `no`. Set the value of the parameter to `yes` to report violations for signals that are read in `always_latch` blocks, and set in non-blocking assignment before or after they are read.

VHDL

The [SetBeforeRead-ML](#) Tag reports process variables that are read before they are set.

A variable carries an X value until it is initialized. When a variable is read before being assigned any value in a `process` construct, this results in an X value propagation. This situation may produce undesired simulation results and may also lead to mismatch between pre- and post-synthesis simulation results.

If a signal is set in the `if` condition of the asynchronous reset branch, the Tag does not report a violation when the signal is read in the `else-if` or `else` block of the `if` condition.

NOTE: *In conditional expressions, a variable is assumed to be initialized if the variable is assigned in all branches of the conditional expression. As no expression reduction*

or matching is done, two different conditional expressions are always considered as different conditions even if they are logically same.

NOTE: *The Tag does not report shared variables because a shared variable is used in different process constructs and is not bound to any particular process construct.*

Message Details

Verilog

The following message appears at the location where the signal `<name>` is read before it is set in an `always` construct:

Signal '`<name>`' read before it is set [Hierarchy: '`<hier-path>`']

The following message appears at the location where the signal `<name>` is read in a task or function description when the signal is not set before the task or function call (at line `<num>` in file `<file-name>`):

Signal '`<name>`' read before it is set in task/function call at line number `<num>` [file:`<file-name>`][Hierarchy: '`<hier-path>`']

VHDL

The following message appears at the location where the variable `<name>` is read before it is set in a `process` construct:

variable `<name>` should be assigned before being read

Severity

Warning

Suggested Fix

Signals should be set before being read.

Examples

Verilog

Example 1

Consider the following example:

```
module top(out1,out2);
```


MoreLint Tags

```

reg a,b;
output out1,out2;
reg out1,out2;
always@(a)
begin
    out2<=a;
    a=1'b1; //blocking assignment
    b<=1'b0;
    out1<=b; //b is not in the sensitivity list
end
endmodule

```

In the above example, the [SetBeforeRead-ML](#) Tag reports two violations because the variable *a* is read and then set using a blocking assignment and signal (that is not present in the sensitivity list) is set using a nonblocking assignment before it is read. The following messages are reported by this example:

signal 'a' read before it is set[Hierarchy: ':top:']

signal 'b' read before it is set[Hierarchy: ':top:']

Example 2

Consider the following example:

```

always (b || c)
    a = b; // No violation for b as it is present in the sensitivity list

```

In the above example, the signal *b* is being read without setting it. But the [SetBeforeRead-ML](#) Tag does not report a violation for the signal *b* as it is present inside the sensitivity list.

Example 3

Consider the following example:

```

module top ();
    bit sig;

    always_comb
    begin
        bit localSig = sig; // Violation for sig because it
        //is read before set
    end
endmodule

```

```

        sig = localSig; // No Violation for localSig because it was set while
//declaring before read here.

```

```

    end
endmodule

```

VHDL

Consider the following example in which the variable V is initialized to 0 in the if condition of the asynchronous reset branch.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
port (Clk, Reset: in std_logic);
end test;

architecture rtl of test is
signal Rdy : std_logic;
begin
Process (Clk, Reset)
    Variable V : integer RANGE 0 to 2;
    Begin
        if Reset = '0' then
            V := 0;
        Elsif Clk'event and Clk = '1' then
            If Rdy = '0' then
                ElsIf V = 2 then
                    V := 0;
                end if;
            end if;
        end if;
    end process;

end rtl;

```

The [SetBeforeRead-ML](#) Tag does not report a violation when the signal V is read in the else-if block of the if condition.

SetResetConverge-ML

Fan-in cone of Set and Reset pin of a flip-flop/latch converge

Language

Verilog, VHDL

Description

The SetResetConverge-ML Tag flags flip-flops/latches where the fan-in cones of their set and reset pins converge.

Same net driving both the set pin and the reset pin of a flip-flop/latch can cause glitches.

When the `enable_Tag_checking_per_ip` parameter is specified through the command line, the Tag reports multiple violations, which are in different IPs, corresponding to the flip-flop/latch instance. That is, if set/reset of two different flip-flops are converging at a single point and both the flip-flops/latches are in different IPs, only then the two violations are reported per converging point.

No violation of flip-flop or latch is reported if their set or reset pin is connected to tied-high/tied-low.

Parameters

- *ignore_chain_cell*: By default, this parameter is set to `no` and the Tag reports only one violation per reset or set source. Set this parameter to `yes` or Tag name to report all violations corresponding to reset or set source.

Message Details

The following message appears at the location where the output of flip-flop `<flop-name>` when the fan-in cones of its set pin `<set-name>` and reset pin `<rst-name>` converge at net `<net-name>`:

```
[ERROR] <Flop|Latch> '<flop-name>' having reset '<rst-name>'
and set '<set-name>' converges at net '<net-name>'
```

Severity

Warning

Suggested Fix

Separate the common driver of set and reset to avoid glitches.

Example Code and/or Schematic

Verilog

Consider the following example:

```
module setreset(in1, in2,in3, clk, out);
    input in1, in2,in3, clk;
    output out;

    reg out;

    wire rst, prst;

    assign rst = in1 & in2;
    assign prst = in1 | in2;

    always @ (posedge clk or negedge rst or negedge prst) begin
        if (~rst) out = 1'b0;
        else if (~prst) out = 1'b1;
        else out = in3;
    end
endmodule
```

In the above example, the *SetResetConverge-ML* Tag reports a violation because the same nets *in1* and *in2* are driving both the set pin and the reset pin of the flip-flop. The following message is reported by this example:

Flop 'setreset.out' having reset 'setreset.rtlc_N0' and set 'setreset.rtlc_N1' converges at net 'setreset.in2'

The schematic for the above example is shown below:

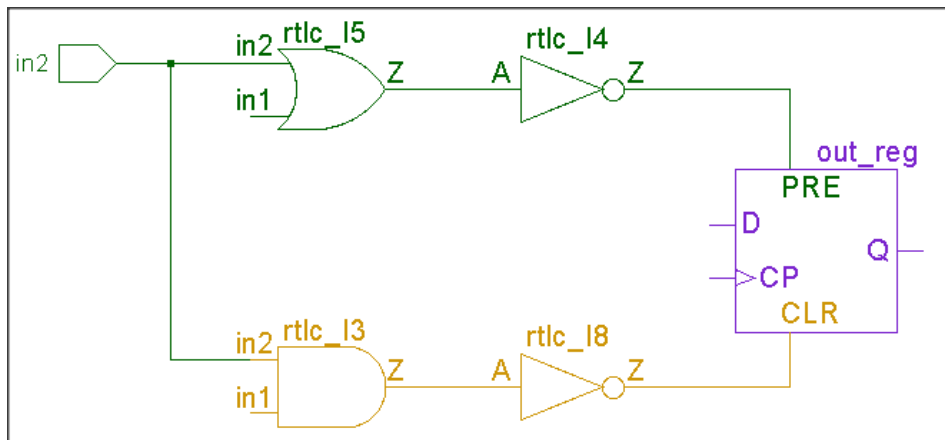


FIGURE 7. Incremental schematic

The following are the color legends for this schematic:

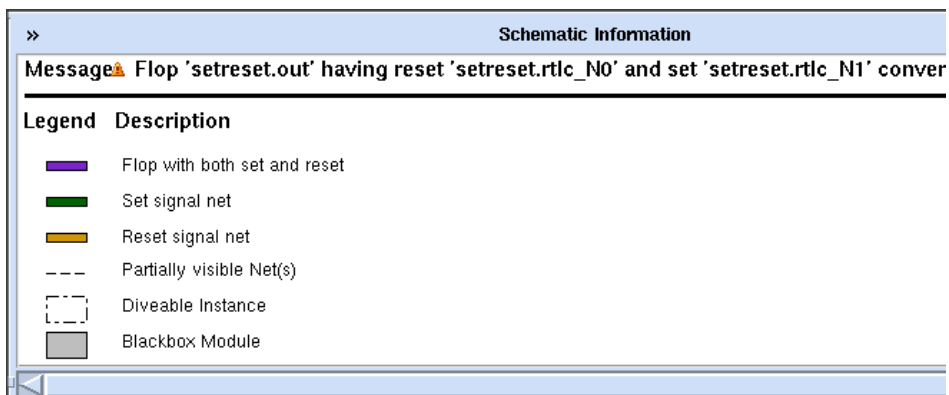


FIGURE 8. Legends for the incremental schematic

VHDL

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

entity top is
port(in1: in std_logic;
      in2: in std_logic;
      in3: in std_logic;
      clk: in std_logic;
      out1: out std_logic);
end top;

architecture arch of top is
  signal rst:std_logic;
  signal prst:std_logic;
  begin
    rst <= in1 and in2;
    prst <= in1 or in2;
    process (clk,in1,in2) is
      begin
        if ((not rst)='1') then
          out1 <= '0';
        elsif ((not prst)='1') then
          out1 <= '1';
        elsif (clk= '1' and clk'event) then
          out1 <= in3;
        end if;
      end process;
    end arch;
```

In the above example, the *SetResetConverge-ML* Tag reports a violation because the same nets *in1* and *in2* are driving both the set pin and the reset pin of the flip-flop. The following message is reported by this example:

Flop 'top.out1' having reset 'top.rtlc_N0' and set 'top.rtlc_N2' converges at net 'top.in2'

SignedUnsignedExpr-ML

Do not mix signed & unsigned variables/constants in expressions

Language

Verilog

Description

The SignedUnsignedExpr-ML Tag flags if signals of type, signed and unsigned, are mixed together to form a single expression, or if there is a type conversion (signed to unsigned, or unsigned to signed) in an assignment or in a comparison statement.

This Tag checks for the type conversions (signed to unsigned, or unsigned to signed) in the following cases:

- Conversion in an assignment, if the value of the *strict* parameter is set to *yes*.
- Conversion during comparison in constructs like *case*, *if*, etc.
- Conversion during binary operations in an expression/sub-expression

This Tag considers an unsized integer constant (for example, 4) both as signed and unsigned to avoid spurious violations.

Set the value of the *dccompat* Tag parameter to *yes* to consider expressions containing unsized integer constants as signed expression and perform Tag checking according to Design Compiler conventions.

For example, the Tag does not report any violation for the following example:

```
wire w=1
```

However, the Tag reports a violation for the following example, if the value of the *dccompat* Tag parameter is set to *yes*:

```
wire w1=1+2
```

The SignedUnsignedExpr-ML Tag checks the left operand of a binary expression in case of the following operators:

- Logical right-shift (>>) and arithmetic right-shift (>>>).
- Logical left-shift (<<) and arithmetic left-shift (<<<).
- Exponential power operator (**)

By default, the value of the *ignoreforindex* parameter is set to *no* and this Tag reports violations related to *for* loop and *while* loop indexes. Set the value of the *ignoreforindex* parameter to *yes* to ignore such violations.

When a bit select or a part select is used as a loop index and the *ignoreforindex* parameter is set to *yes*, then violations for the whole array are ignored. For example:

```
integer j[3];

for(j[0] = 1; j[0] < 3; j[0]++)
    j[1] = j[1] + 3'b111;
```

In the above example, no violation is reported for whole array bits in *j* as *j* is marked as *for/while* loop index.

By default, the *check_all* parameter is set to *no* and this Tag does not check for all of the data types. Set the *check_all* parameter to *yes* to enable the *SignedUnsignedExpr-ML* Tag to check for all of the data types.

Message Details

The following message appears at the location of an expression that has a signed part *<expr2>* and an unsigned part *<expr1>*:

Unsigned expression '*<expr1>*' used with signed expression '*<expr2>*'*<stmt>*

Where, *<stmt>* can be in assignment or in comparison depending upon whether the type (signed or unsigned) conversion has occurred in an assignment statement or in a comparison statement.

Severity

Warning

Examples

The SignedUnsignedExpr-ML Tag reports a violation in the following examples due to type conversion (signed to unsigned, or unsigned to signed):

Example 1

```
reg signed [8:0] floor;
```


MoreLint Tags

```

wire    [8:0]    sum;
always @(posedge clk)
    floor <= sum[3:0]; //unsigned to signed

```

Example 2

```

input [1:0] cr_scale;
reg signed [8:0] mux_scale;
reg sm_correct_dly1;
always @(*) begin
    case (sm_correct_dly1)
        1'b1 : mux_scale = cr_scale; // unsigned to signed
        default : mux_scale = cr_scale; // don't care
    endcase
end

```

Example 3

```

reg signed [1:0] mux_scale;
always @(*) begin
    case (mux_scale) // signed to unsigned
        2'd0: begin end
        2'd1: begin end
        2'd3: begin end
    endcase
end

```

Example 4

```

reg signed [1:0] mux_scale;
reg [3:0] out;
wire signed [3:0] in2;
wire [3:0] in1;
always @(*)
    if (mux_scale == 2'd2) // signed to unsigned
        out[3:0] = in1;
    else

```

```
out[3:0] = in2; // signed to unsigned
```

SynchReset-ML

Do not use synchronous reset in the design

Language

Verilog, VHDL

Description

The SynchReset-ML Tag flags synchronous resets in the design.

Use of synchronous reset may lead to many optimized implementation of reset, some of which may lead to ambiguous initial states. Therefore, avoid using synchronous resets in the initial reset.

Message Details

The following message appears at the location of a synchronous reset
<rst-name>:

Synchronous reset '*<rst-name>*' used

Severity

Warning

TaskFileName-ML

FileName should be '<taskname>.v' or '<taskname>.sv' for a file, which contains only task definition(s)

Language

Verilog

Description

The TaskFileName-ML Tag flags source files that contain only task definitions but the file name is not same as the name of one of the tasks described in the source file.

The TaskFileName-ML Tag also checks source files that contain task definitions and descriptions of unused functions.

For Tag-violating source files, the message reports the name of the first task found in the source file.

Message Details

The following message appears for a Tag-violating source file where the first task is named *<task-name>*:

Use name '*<task-name>.<file-extn>*' for file containing task definition

Where *<file-extn>* can be *v* or *sv* depending upon the extension of the source file.

Severity

Guideline

UnConstrLoop-ML

Number of iterations in <for | while> -loop exceeds max limit (Verilog)
Unconstrained loops should not be used (VHDL)

Language

Verilog, VHDL

Description

Verilog

The *UnConstrLoop-ML* Tag flags a while/for loop whose condition always remain true, that is, the loop has no break control or there is no way for the loop to terminate. Such a condition keeps the loop running and exceeds the maximum iteration limit. Such while/for loops are not synthesizable and should not be left unconstrained.

By default, the maximum loop iteration limit is 2048. You can change this maximum limit by using the following command:

```
set_app_var lint_max_loop_limit <value>
```

Where, *<limit-value>* is the integer value that you want to set as the maximum iteration limit for the loop.

VHDL

The *UnConstrLoop-ML* Tag flags following type of unconstrained loops which may result into infinite loops:

- loop ...end loop
- while (<always true condition>) ... end while
- forever...end
- repeat (<number of times to be repeated>)...end

Message Details

Verilog

The following message appears when a <while/for> loop exceeds maximum iteration limit <max-limit> in the design:

Number of iterations in <while/for>-loop exceeds maximum allowable limit (<max-limit>)

VHDL

The following message appears at the location of an unconstrained loop:
Unconstrained loop found

Severity

Error (Verilog)/ Warning (VHDL)

Examples

Verilog

The *UnConstrLoop-ML* Tag flags the following example as the while loop condition is true with no terminating condition:

```
reg out1, a;

always @(sel)
begin
    a = 1'b1;        // Loop variable 'a' is assigned
                    // value here
    while (a > 0)    // Loop condition is true and there
                    // is no terminating condition
    begin
        case(sel)
            1'b0: out1 <= in1;
            1'b1: out1 <= in2;
        endcase
    end
end
```

VHDL

The *UnConstrLoop-ML* Tag flags the following example that has an unconstrained loop:

```
library ieee;
use ieee.std_logic_1164.all
```

MoreLint Tags

```
process (in1)
  constant c : integer := 1;
  begin
    loop
      null;
    end loop;
```

UnrecSynthDir-ML

Synthesis directive is not recognized

Language

Verilog, VHDL

Description

The UnrecSynthDir-ML Tag flags unrecognized synthesis directives.

By default, the UnrecSynthDir-ML Tag flags synthesis directives other than the following:

dc_script_begin	dc_script_end	enum	full_case
map_to_module	parallel_case	return_port_name	state_vector
template	translate_off	translate_on	infer_mux

In addition, the UnrecSynthDir-ML Tag flags synthesis directives other than the following when specified with a list of object names:

async_set_reset	async_set_reset_local	async_set_reset_local_all
dont_infer_multibit	enum	infer_multibit
infer_mux	label	label_applies_to
map_to_module	one_cold	one_hot
resource	return_port_name	state_vector
sync_set_reset	sync_set_reset_local	sync_set_reset_local_all

The UnrecSynthDir-ML Tag recognizes synthesis directives of the following types:

Verilog

```
// synopsys <directive>  
or  
/* synopsys <directive> */
```


VHDL

```
-- synopsys <directive>
```

or

```
-- pragma <directive>
```

Use the [synth_dir_list](#) Tag parameter to specify your own list of recognized synthesis directives (overriding the default list). Then, the UnrecSynthDir-ML Tag flags all other synthesis directives.

The VC SpyGlass Lint `-Xspyglass_pragma=<value>` command affects the UnrecSynthDir-ML Tag.

Use the [check_all_pragmas](#) parameter to report violations for unrecognized pragmas, which are semicolon-separated or space-separated in a single line comment.

Message Details

The following message appears at the location of an unrecognized synthesis directive of type `<pragma>`:

```
Synthesis directive '<synth-dir>' is not recognized for pragma
'<pragma>'
```

Severity

Warning

Examples

The UnrecSynthDir-ML Tag flags the following example that has an unrecognized synthesis directive:

```
module test(a, b);
    input a;
    output b;
    assign b = a; //synopsys my_directive
endmodule
```

UseSVAAlways-ML

Checks the use of System Verilog constructs for modeling of procedural combinatorial and sequential logic

When to Use

Whenever an always block is used in System Verilog design.

Description

This Tag makes sure that the System Verilog constructs (`always_comb`, and `always_ff`) are used for modeling of procedural combinatorial and sequential logic. In this Tag, latches are considered as combinational logic.

Default Weight

5

Language

Verilog, System Verilog

Parameter(s)

strict: By default, the Tag reports only if an SV construct is there inside the module. Then the Tag suggests to use `always_ff`, if an `always` block is sequential or `always_comb`, if an `always` block is combinational.

When you set the `strict` parameter, the Tag does not check for SV construct inside the module. The Tag suggests to use `always_ff` if any `always` block infers a flip-flop, `always_latch` if any `always` block infers a latch, or `always_comb` if any `always` block is combinational.

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears when an `always` block is inferring the sequential logic and `always_ff` construct is not used:

[WARNING] Preferably use `always_ff` construct for sequential

Logic inferring flip-flop

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

Double click the violation message. The HDL viewer window highlights the line where violation is detected. Fix the violation by using `always_ff` construct.

Message 2

The following message appears when an `always` block is inferring combinational logic and `always_comb` construct is not used:

[WARNING] Preferably use `always_comb` construct for procedural combinatorial code

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

Double click the violation message. The HDL viewer window highlights the line where violation is detected. Fix the violation by using `always_comb` construct.

Example Code and/or Schematic

Example 1

In the following example, the *UseSVA/ways-ML* Tag reports a violation because `always` block is inferring sequential logic and `always_ff` construct is not used:

```
always @ (posedge clk)    // Violation for UseSVAways-ML
for always_ff
begin
    signal2 <= signal2 + in1;
```

```
        out2 = in2;  
    end
```

Example 2

In the following example, the *UseSVAalways-ML* Tag reports a violation because always block is inferring combinational logic and always_comb construct is not used:

```
always @ (clk)    // Violation for UseSVAalways-ML for  
always_comb  
begin  
    signal2 <= signal2 + in1;  
    out2 = in2;  
end
```

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

WrapInstance-ML

Black box cell should not be contained on its own in a wrapper module

Language

Verilog, VHDL

Description

The WrapInstance-ML Tag flags unwanted black box instances in a wrapper design unit.

The WrapInstance-ML Tag requires you to specify the names of the allowed cells in a design unit using the [wrapperDuCellList](#) Tag parameter as in the following example:

```
configure_lint_tag_parameter -tag WrapInstance-ML -parameter  
wrapperDuCellList -value "DU1 c1 c2 c3, DU2 c2 c5"
```

Here, instances of cells c1, c2, and c3 are allowed in the design unit DU1 where instances of cells c2 and c5 are allowed in the design unit DU2.

Then, the WrapInstance-ML Tag flags all other black box instances (instances of cells other than the respective allowed cells or instances of design units whose RTL description is not available) in design units DU1 and DU2.

For mapping FPGA design to an ASIC implementation, it is necessary that there should not be any black box cell in a wrapper design unit. For performing this, the WrapInstance-ML Tag checks that all instantiated cells are from a specified set only.

NOTE: *The WrapInstance-ML Tag supports generate-for block.*

Message Details

The following message appears at the location where a Tag-violating black box cell `<bb-name>` is instantiated on inside a wrapper design unit `<du-name>`:

Instantiation of blackbox '`<bb-name>`' in the design unit '`<du-name>`' is not correct

Severity

Warning

ConstantInput-ML

Input to a particular module is tied to a constant value

Language

Verilog, VHDL

Description

The *ConstantInput-ML* Tag checks signals connected to the input ports of an instance, including grey box, and reports a violation if an input is tied to a constant value signal.

By default, the Tag ignores only VC SpyGlass Lint-generated buffers and inverters in the path. Set the [allow_combo_logic_base](#) parameter to *yes* to ignore all VC SpyGlass Lint-generated combinational elements in the path.

By default, the Tag reports only one violation for a port per instance inside the *generate-for* construct. Set the [strict](#) parameter to *yes* or *ConstantInput-ML* to report all iterations of Tag-violating instances in the *generate* construct.

By default, the Tag does not report violations for gate instances. Set the [check_gate_primitives](#) parameter to *yes* to report violations for gate instances.

By default, the Tag reports all the instances of constant-driven scenarios in the design. Set the value of the [report_only_first](#) parameter to *yes* to report only one instance of the forward cone starting from the constant input.

By default, the *ConstantInput-ML* Tag does not report violations for black box instances, which are instantiated at the top-level. Set the value of the [check_blackboxes](#) parameter to *yes* to report violations for black boxed instances.

By default, the tag reports violation for only higher-level hierarchies. Set the value of the [report_all_hier](#) parameter to *yes* to report violation for lower-level hierarchies as well

NOTE: *The ConstantInput-ML Tag supports the generate-for block.*

Message Details

The following message appears at the location of the instance *<inst-name>* where the input pin *<pin-name>* is connected to a

constant value variable/net *<name>* of the value *<value>*:

Input '*<pin-name>*' to Instance '*<inst-name>*' is assigned a constant, signal '*<name>*' all bit(s) set to value '*<value>*'
[Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the instance hierarchy.

Severity

Warning

HangingInstInput-ML

Net connected to input port of instance is undriven

Language

Verilog, VHDL

Description

The HangingInstInput-ML Tag flags undriven nets connected to instance input ports.

The HangingInstInput-ML Tag flags the hanging nets shown in red color in the following figure:

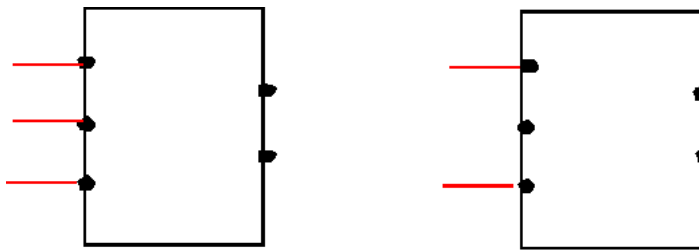


FIGURE 9. Hanging nets

The HangingInstInput-ML Tag checks the nets connected to instance input ports and does not report unconnected instance input ports.

The HangingInstInput-ML Tag does not check nets connected to instance inout ports.

By default, the HangingInstInput-ML Tag traverses hierarchically inside the user-instantiated cells to check for undriven inputs. Set the [*checkInHierarchy*](#) Tag parameter to `no` to prevent the HangingInstInput-ML Tag check inside the hierarchy. Then, the HangingInstInput-ML Tag checks only those input pins that are loaded (that is, the input pin is connected to input pin of a combinational gate, a sequential cell instance, a user-instantiated cell, or a black box instance in the parent design unit).

The HangingInstInput-ML Tag assumes the input pins of black box instances to be driven.

The HangingInstInput-ML Tag ignores instances of stopped design units (design units specified with the `set_blackbox -designs { design-unit }` command). All nets connected to the input ports of instances of such modules are assumed to be driven.

The HangingInstInput-ML Tag assumes black box input/output pins as driven.

For a transistor to function as an on-off switch, the gate of a cmos transistor should be at a defined state and at an electrical level of logic, 1 or 0. If the gate of a transistor is left floating, the corresponding transistor can get into a partially on, *limbo*, state.

In the context of a logic gate, such a *partially on* transistor can lead to an indeterminate logic value and analog electrical state on the output of that gate.

For an inverter, if the input is left floating, both the n and p transistors can become *partially on* transistors, thereby creating a steady state path from Vdd to ground. In such cases, the output of a transistor would be in a non-digital state (i.e., somewhere between logic 1 and 0). This state would then propagate to all the input pins of gates in the fan-out cone of the inverter. Those gates, in turn, would produce a non-digital output and would propagate further. Thus, potentially large number of nodes in a chip would be in an indeterminate non-digital state.

In addition, the steady state path created between Vdd and ground through the inverter transistors leads to an excessive current draw from Vdd. When many nodes are in this indeterminate state, a large amount of current is drawn from Vdd resulting in a burn-out or latch-up of the chip.

If the [handle_greybox](#) parameter is set to `yes`, then the net connected to grey box terminal is considered as driven.



FIGURE 10. Grey box terminal

Design Impact

Functionality, testability, and reliability

Message Details

The following message appears at the location of instance where net `<net-name>` connected to input terminal `<input-term-name>` is an undriven net:

Net '`<net-name>`' connected to input port of instance is 'undriven'

Severity

Warning

HangingInst-ML

Block is not connected

Language

Verilog, VHDL

Description

The HangingInst-ML Tag flags hanging instances.

The HangingInst-ML Tag flags those instances where all terminals are hanging (either unconnected or connected to hanging nets) as shown in the following figure:

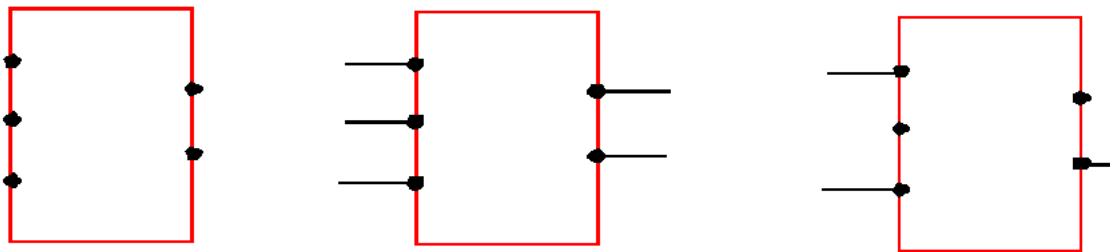


FIGURE 11. Hanging nets

Message Details

The following message appears at the location of a hanging instance

<name>:

Block *<name>* is not connected properly

Severity

Warning

HangingInstOutput-ML

Net connected to output port of instance is unconnected

Language

Verilog, VHDL

Description

The HangingInstOutput-ML Tag flags unconnected nets that are connected to instance output ports.

The HangingInstOutput-ML Tag flags the hanging nets shown in red color in the following figure:

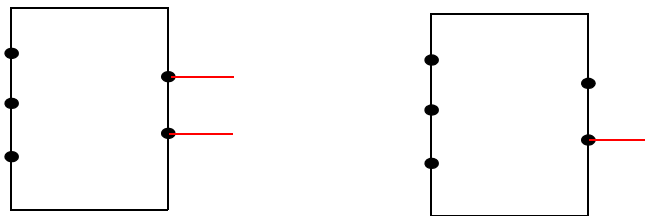


FIGURE 12. Hanging nets

The HangingInstOutput-ML Tag does not check instance inout ports.

Message Details

The following message appears at the location of instance where net `<net-name>` connected to output terminal `<output-term-name>` is an unconnected net:

Net '`<net-name>`' connected to output port of instance is 'unconnected'

Severity

Warning

NoBusPartClock-ML

Bus bits or slices should not be used as clocks

Language

Verilog, VHDL

Description

The NoBusPartClock-ML Tag flags bus bits or slices used as clocks.

The NoBusPartClock-ML Tag checks for all the clock signals starting from source to the point-of-use.

Using scalar signals as clocks helps in avoiding problems during clock tree insertion step of the layout phase.

Message Details

The following message appears at the location where the output of a flip-flop that is triggered by a Tag-violating clock *<clk-name>*, is assigned:

clock name '*<clk-name>*' is not a simple name

Severity

Warning

Examples (Verilog)

The NoBusPartClock-ML Tag flags the following example where flip-flop out1 is clocked by a bus-bit clk[0]:

```
...
always @ (posedge clk[0])
    out1 <= in1;
...
```

Examples (VHDL)

The NoBusPartClock-ML Tag flags the following example where flip-flop q is clocked by a bus-bit clk(0):

```
library ieee;
```

MoreLint Tags

```
use ieee.std_logic_1164.all

process(clk(0), rst)
begin
  if(rst = '1') then
    q <= '0';
  elsif (clk(0)'event and clk(0) = '1') then
    q <= d;
  end if;
end process;
```

NoFeedThrus-ML

Block should not contain feed-throughs

Language

Verilog, VHDL

Description

The NoFeedThrus-ML Tag flags feed-throughs.

A feed-through is a direct connection (ignoring buffers) from an input port to an output port of a block. Circuits that contain feed-throughs can create problem in ASIC environments. Hence, the FPGA designs should not have feed-throughs to ensure correct mapping of the design to an ASIC implementation.

By default, the NoFeedThrus-ML Tag reports only one violation for any one instance of the module. If you set the value of the [ignore_chain_cell](#) parameter set to *yes* or *<Tag-name>*, the NoFeedThrus-ML Tag reports all violations corresponding to all instantiation of a module.

Feed-through issue relates to timing predictability. Usually, large blocks are provided with some feed-throughs. These may be used by timing-critical signals (outside the block), so that they do not have to travel around the large block but could simply pass through it.

For a soft-IP (even gate-level but not APRed), the timing through this feed-through might not be known (this becomes important because the signal using it is supposed to be timing-critical).

Further, the signal (of the whole chip) to which this block is connected could have potential signal-integrity impact. It could either be influenced by other signals around it or could influence other signals inside the block. Thus, block-level timing-analysis could be significantly impacted depending on the frequency and drive of the signal which is connected to this feed-through.

By default, the [checkTopDu](#) parameter is set to *no* and the *NoFeedThrus-ML* Tag reports violation for feedthrough connections present in every module. If you set the parameter value to *yes* or *NoFeedThrus-ML*, then the Tag reports violation for feedthrough connection present in the top module only.

By default, the Tag ignores only VC SpyGlass Lint-generated buffers and

inverters in the path. Set the [allow_combo_logic_base](#) parameter to `yes` to ignore all combinational elements in the path, which are treated as feedthrough connections.

For example:

```
module top (input in0, in1, output out);  
    and(out, in0, in1);  
endmodule
```

In this example, the Tag reports the following violation message:

There is feed-through from input 'top.in0' to output 'top.out'

When you set [categorize_violations](#) parameter to `yes`, the violations are reported with **Error** and **Warning** severity labels.

- The violations for the feedthrough paths that are passing through buffers and inverters are labeled as Error.
- The violations for the feedthrough paths passing through other combination logics are labeled as Warning.

Message Details

The following message appears at the location of a feed-through from input port `<in-port>` to output port `<out-port>`:

There is feed-through from input '<in-port>' to output '<out-port>'

Severity

Warning, Error

ReEntrantOutput-ML

The re-entrant outputs should be avoided

Language

Verilog, VHDL

Description

The ReEntrantOutput-ML Tag flags the instance output terminals that are directly connected (ignoring buffers) to an input terminal of the same instance.

Message Details

The following message appears at the location of an instance where the output terminal `<out-pin>` is directly connected to the input terminal `<in-pin>`:

Output port '`<out-pin>`' is driving the input port '`<in-pin>`' of the same instance

Severity

Guideline

RegInputOutput-ML

Register the module output and input ports.

When to Use

Use this Tag to identify primary port signals that are not registered before/after entering or before/after leaving a module boundary.

Description

The RegInputOutput-ML Tag checks if an input is registered either before entering the module (instance) or after it has entered the module boundary. Similarly, it checks whether an output is registered before leaving the module or after leaving the module boundary.

The tag performs a check for all the hierarchies of a module.

For the input side, this Tag checks whether all the nets connected to the input port are registered or not inside the module.

Prerequisites

The RegInputOutput-ML Tag requires the prerequisite Tag named *Prereqs_RegInputOutputs* to be run to consider block pragmas for its Tag-checking.

Tag Exceptions

The RegInputOutput-ML Tag does not report violation for clock/reset input ports. The Tag traverses from input port and ignores all combinational gates (BUFF, INV, OR, AND, XOR, NAND, NOR, XNOR, and multiplexer inputs only) to check for signals used as clock/reset pin of a flip-flop. The Tag does not cross boundary of a module or instance during this traversal, unless the [handle_hier_clock_reset](#) parameter is set. If any port acts as a clock/reset, then the RegInputOutput-ML Tag does not report any violation.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

- ***buf_count***: The default value is -1. Set the value of the parameter to any positive integer value to specify the maximum number of buffers allowed between the input/output port and register input/output.
- ***CHECKSIBLING***: The default value is yes. Set the value of the parameter to no to enable the tag to report the sibling hierarchies also.
- ***chkTopModule***: The default value is no. Set the value of the parameter to yes to check only for all the top-level design units or those top-level design units with VC SpyGlass Lint block pragmas specified.
- ***depth_ml***: The default value of the parameter is -1 and the Tag checks for all levels of hierarchy. Set the value of the parameter to any integer value to enable the Tag to check at the specified level of hierarchy.
- ***inv_count***: The default value is 1. Set the value of the parameter to any positive integer value to specify the maximum number of inverters allowed between the input/output port and register input/output. You can also set the value of the parameter to -1 to ignore all inverters.
- ***ignore_sync_reset***: The default value is no. Set the value of the parameter to yes to ignore the gates generated due to synchronous reset of flip-flops.
- ***ignore_unloaded_port***: The default value is no. Set the value of the parameter to yes to ignore the violations for inputs ports that are unloaded.
- ***handle_hier_clock_reset***: The default value is no. Set the value of the parameter to yes to enable the *RegInputOutput-ML* Tag to go beyond the module boundary to detect clock and reset ports.

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears at the location where an input port *<port-name>* is first read if the input port is not registered before or after entering a module boundary in hierarchy *<hier-path>*:

[WARNING] Port '<port-name>' is not registered [Hierarchy: '<hier-path>']

Potential Issues

Violation may arise when an input port signal is not registered before or after entering a module boundary.

Consequences of Not Fixing

If a block is intended to be reusable, it must have predictable timing behavior. Input ports that are not registered have timing behavior dependent on internal delays which vary depending on target technology and process variations. Registering all inputs assures that timing is controlled by the clock, rather than by these potentially unpredictable parameters.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

The following message appears at the location where an output port <port-name> is first assigned if the output port is not registered before or after leaving a module boundary in hierarchy <hier-path>:

[WARNING]Port '<port-name>' is not registered [Hierarchy: '<hier-path>']

Potential Issues

Violation may arise when an output port is not registered before or after leaving a module boundary.

Consequences of Not Fixing

If a block is intended to be reusable, it must have predictable timing behavior. The output ports that are not registered have timing behavior dependent on internal delays which vary depending on target technology and process variations. Registering all outputs assures that timing is controlled by the clock, rather than by these potentially unpredictable parameters.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the

line where the port net is being used.

Also, the Incremental Schematic of the violation message shows the unregistered output ports.

To fix the violation, ensure that the signal is registered either before or after leaving or entering a module boundary.

Example Code and/or Schematic

Verilog

Consider the following example:

```
module test(clk,in1,out1,out2);

    input clk;
    input [3:0] in1;
    output [3:0] out1;
    reg [3:0] out1;
    inout [3:0] out2;
    always @(in1)
        out1[2:0] = in1 [3:1];
    always @(clk)
        out1[3] = in1[0];
endmodule
```

In the above example, the RegInputOutput-ML Tag reports a violation as the module output port, out1 [2:0] and input port, in1 [3:1] are not registered.

VHDL

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all;

entity ent1621_Test1 is
    port ( in1_ent1621_Test1, in2_ent1621_Test1 : in std_logic;
          out_ent1621_Test1 : out std_logic );
end ent1621_Test1;
```

MoreLint Tags

```
architecture structure_ent1621_Test1 of ent1621_Test1 is
begin
    out_ent1621_Test1 <= in1_ent1621_Test1 and
    in2_ent1621_Test1;
end structure_ent1621_Test1;
```

In the above example, the RegInputOutput-ML Tag reports a violation as the port, out_ent1621_Test1, is not registered.

Default Severity Label

Warning

Tag Group

No Applicable

Reports and Related Files

No related reports or files.

ResetFlop-ML

All the flip-flops should have either asynchronous set/reset or synchronous set/reset

Language

Verilog, VHDL

Description

The ResetFlop-ML Tag flags flip-flop descriptions that have neither asynchronous set/reset nor synchronous set/reset.

You can also use the [synch_reset_list](#) parameter to provide a comma-separated list of synchronous reset signals used in the design. The ResetFlop-ML Tag traverses in the fan-in cone of the D-pin of a flip-flop, and if a signal in this fan-in cone matches with the name in list, then flip-flop is considered to have a reset.

NOTE: *If the D pin of the flip-flop is tied to supply, then the Tag will try to search for valid synchronous reset on the enable path of that flip-flop.*

It traverses through various combinational gates, such as, buffer, inverter, or, nor, and, nand, xor, xnor and mux to find the reset. The traversal never goes outside the boundary of the module in which the flip-flop is present.

The ResetFlop-ML Tag also flags flip-flop descriptions that have set/reset descriptions but do not use scalar signals or bit-selects.

All flip-flops used in the design should have initial set/reset pins for initializing the state of the design. If a flip-flop is not controllable from external pin, it would have an undefined value after initialization and may produce undesirable results.

By default, the ResetFlop-ML Tag reports violation for flip-flops, which are inferred from multidimensional arrays. Set the value of the [ignore_multidim](#) parameter to `yes` to suppress such violations.

Message Details

The following message appears at the location where the output `<name>` of a flip-flop description that has neither asynchronous set/reset nor synchronous set/reset, is first assigned:

Flip-flop '`<name>`' has no set or reset. [Hierarchy: '`<hier-path>`']

Severity

Warning

Example

VHDL

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

if (clk'event and clk = '1') then
  q_out1 <= d_in1;
```

In the above example, flip-flop, q_out1 is described without a set or reset description:

In the following example, flip-flop q_out2 has a reset description but uses a multi-bit signal:

```
if (clk'event and clk = '1') then
  if (reset = "00") then
    q_out2 <= (others => '1');
  else
    q_out2 <= d_in2;
  end if;
```

Verilog

Consider the following example:

```
module casel(clock, srst, din);
input clock;
input srst;
input [7:0] din;
reg [7:0] qOut;
always@(posedge clock)
  if(!srst)
  begin
    qOut[1] = 0;
    qOut[2] = din[2];
  end
```

```
else
begin
    qOut[1] = din[1];
    qOut[2] = 0;
end
endmodule
```

In the above example, the ResetFlop-ML Tag tries to find the synchronous reset, `srst`, itself.

However, if the Tag algorithm is not able to detect the synchronous reset itself, you can specify the synchronous reset by setting the value of the [synch_reset_list](#) parameter to appropriate value.

RptNegEdgeFF-ML

Negative edged flip-flops should not be used in the design.

Language

Verilog, VHDL

TagDescription

The RptNegEdgeFF-ML Tag flags negative edged flip-flops and positive edged flip-flops with inverted clock signal.

For lib cells, the positive edged flip-flops are detected when the related pin attribute of pin (D) is set to CLK and its `timing_type` attribute is set to `setup_rising`. The negative edged flip-flops are detected when the related pin attribute of pin (D) is set to CLK and its `timing_type` attribute is `setup_falling`.

By default, the *RptNegEdgeFF-ML* Tag traverses through buffers and inverters in the clock path to check whether the clock signal is inverted. Set the value of the [allow_combo_clock](#) parameter to `yes` to enable the *RptNegEdgeFF-ML* Tag to traverse through combinational logic to check whether the clock signal is inverted.

NOTE: *False violations are reported if a clock path has an odd number of inverters and the path has a self loop.*

Message Details

The following message appears at the location of the negative edged flip-flop `<flop-name>` :

Negative Edged flip-flop `<flop-name>` should not be used. Number of inverters in clock fanin are `<num>`

The following message appears at the location of the positive edged flip-flop `<flop-name>` with its clock signal inverted:

Positive Edged Flip-Flop `<flop-name>` with negative edge clock should not be used. Number of inverters in clock fanin are `<num>`

The Incremental Schematic of the violation message highlights the flip-flop and the clock path.

Example Code and/or Schematic

Example

Consider the following example:

```
module packed1 (input clk, input [7:0] in1, in2, output reg
[7:0] out1);
bit [3:0] [4:0] [0:7] my_bit;
logic [5:0] [14:0] [0:8] my_logic;
reg [13:0] [4:10] [0:9] my_reg;

always @ (posedge clk)
begin
    my_bit[3][4] = in1 - in2;
    my_logic[0][14] = in1 + in2;
    my_reg[13][10] = in1[0] ? my_bit[3][4] : my_logic[0][14];
end

always @ (negedge clk)
begin : For_out
    out1 = my_reg[13][10];
end
endmodule
```

VC SpyGlass Lint generates the following message for this example:

Positive Edged Flip-Flop output net name packed1.out1[7:0] with negative edge clock should not be used. Number of inverters in clock fanin are 1

Also, consider the following figure:

MoreLint Tags

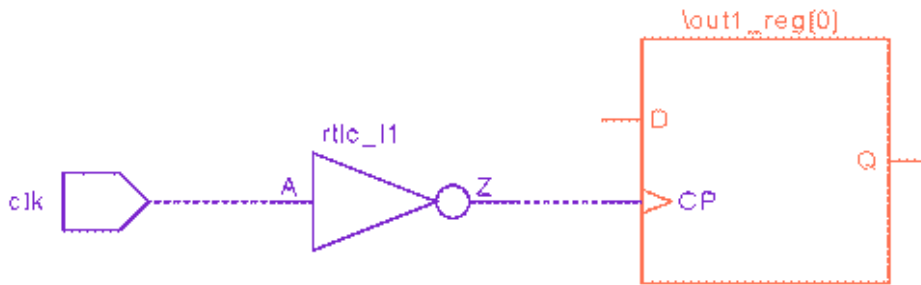


FIGURE 13. Incremental schematic

In the above figure, the the flip-flop and the clock path are highlighted.

Severity

Info

SameControlNDataNet-ML

Control (set/reset) signal used as data for the same flip-flop.

Language

Verilog, VHDL

Description

The SameControlNDataNet-ML Tag flags flip-flops where control signals are used as data for the same flip-flop.

The SameControlNDataNet-ML Tag reports violation if the same net drives a control pin (set/reset) and the data pin.

Such a design causes problems in automatic test pattern generation (ATPG).

By default, the SameControlNDataNet-ML Tag ignores internally generated buffers and even number of inverters between the path of control pin and data pin.

If you set the [ignore_inverters](#) Tag parameter to *yes*, then all internally generated inverters will be ignored, along with the buffers.

Message Details

The following message appears at the location where flip-flops having set or reset pin `<pin-type> <pin-name>` is connected to data of the same flip-flop `<flip-flop-name>`:

```
<pin-type> '<pin-name>' is connected to data pin of same flop  
'<flip-flop-name>'
```

Severity

Warning

Suggested Fix

Use different signals for data input and control input.

TristatePort-ML

Tristate port detected

Language

Verilog, VHDL

Description

The TristatePort-ML Tag flags tristate ports.

When a port is assigned a tristate value, it is generally not a problem. Timing requirements are more stringent for tristates and you need to take care of driver strengths. Also, tristate ports are not recommended in FPGA based-designs.

Set the value of the [strict](#) Tag parameter to `yes` or `TristatePort-ML` to ignore chip-level (top-level) ports.

Parameters

[CHECK_Z_ASSIGN](#): By default, this parameter is set to 0. This indicates that the tag will not report violations on Z assignment cases.

If you set this parameter to 1, the tag will report violations on Z assignment cases.

Message Details

The following message appears at the location where a port `<port-name>` is being assigned a tristate value:

Tristate port '`<port-name>`' detected

Severity

Info

Examples (Verilog)

Consider the following example where the port `out1` is being assigned a tristate value:

```
always @(posedge clk)
  case (rst)
    1'b1 : out1 = in1;
    1'b0 : out1 = 1'bz;
  endcase
```

For this example, VC SpyGlass Lint generates the following message:

Tristate port 'out1' detected

Examples (VHDL)

Consider the following example where the port output is being assigned a tristate value:

```
library ieee;
use ieee.std_logic_1164.all

process ( in1, in2 )
begin
  if ( in1 > in2 ) then
    output <= 'Z';
  elsif ( in1 < in2 ) then
    output <= '0';
  else
    output <= 'X';
  end if;
end process;
```

For this example, VC SpyGlass Lint generates the following messages:

Tristate port 'output' detected

TristateSig-ML

Tristate signal detected

Language

Verilog, VHDL

Description

The TristateSig-ML Tag flags tristate signals.

When a signal is assigned a tristate value, it is generally not a problem. Timing requirements are more stringent for tristates and you need to take care of driver strengths. Also, tristate signals are not recommended in FPGA based-designs.

Exceptions

The *TristateSig-ML* tag does not report violation for the following conditions:

- There is a constant on enable.
- Data pin of TriState buffer is connected to constant.

Parameters

CHECK_Z_ASSIGN: By default, this parameter is set to 0. This indicates that the tag will not report violations on Z assignment cases.

If you set this parameter to 1, the tag will report violations on Z assignment cases.

Message Details

The following message appears at the location where a signal *<sig-name>* is being assigned a tristate value:

Tristate signal '*<sig-name>*' detected

Severity

Info

Examples (Verilog)

Consider the following example where the signal `r1` is being assigned a tristate value:

```
reg r1;

always @(posedge clk)
  case (rst)
    1'b1 : r1 = in1;
    1'b0 : r1 = 1'bz;
  endcase
```

For this example, VC SpyGlass Lint generates the following message:

Tristate signal 'r1' detected

Examples (VHDL)

Consider the following example where the signal `sig1` is being assigned a tristate value:

```
library ieee;
use ieee.std_logic_1164.all

process(en, in1, sig1)
begin
  if (en = '0') then
    sig1 <= 'Z';
    out1 <= in1;
  else
    sig1 <= in1;
    out1 <= 'Z';
  end if;

  out2 <= out1;
end process;
```

For this example, VC SpyGlass Lint generates the following messages:

Tristate signal 'sig1' detected

Undriven-ML

Undriven nets/terminals detected in the design

The Undriven-ML Tag group comprises of the [UndrivenNet-ML](#), [UndrivenOutPort-ML](#), [UndrivenInTerm-ML](#), and [UndrivenOutTermNLoaded-ML](#) Tags.

UndrivenInTerm-ML

Detects undriven but loaded input terminal of an instance

When to Use

Use this Tag to identify undriven but loaded input terminal of an instance.

Description

The UndrivenInTerm-ML Tag reports violation for input terminals that are not driven but are loaded.

An input terminal is assumed to be loaded, if it is connected to an input pin of a combinational gate, a sequential cell instance, or a user-instantiated cell in the parent design unit.

The UndrivenInTerm-ML Tag does not cross boundary of a library cell in any of its traversal. If a net is connected to an input (output) terminal of a library cell, then the net is considered as loaded (driven).

For other input terminals, the first connected output pin of a combinational gate, a sequential cell instance, or a user-instantiated cell is assumed to be the driver.

A gate of a CMOS transistor needs to be at a defined state and electrical level of logic 1 or 0, for the transistor to function as an on-off switch. If the gate of a transistor is left floating, then the corresponding transistor can get into a limbo state of partially on.

In the context of a logic gate, such a partially-on transistor can lead to indeterminate logic value and analog electrical state on the output of that gate.

For an inverter, if the input is left floating, both the *n* and *p* transistors can be partially on creating a steady state path from VDD to ground.

In such cases, the output of a transistor would be in a non-digital state, that is, somewhere between logic 1 and 0. This state would then propagate to all the input pins of gates in the fan-out cone of the inverter. Those gates, in turn would produce a non-digital output and would propagate further. Thus potentially large number of nodes in a chip would be in the indeterminate non-digital state.

Also, the steady state path created between VDD and ground through the

inverter transistors leads to an excessive current draw from VDD. When many nodes are in this indeterminate state, a large amount of current is drawn from VDD leading to a burn-out or latch-up of the chip.

Results of Tags, such as UndrivenInTerm-ML, UnloadedNet-ML, UndrivenNUnloaded-ML, and checkNetDriver may get interchanged for signals that are explicitly assigned the value of 0 or 1 and are not driving anything. As a result, when a particular violation is expected, some other violation may be reported. Consider the following example:

```
assign b = 1'b0; // "b" is not even being used
```

For the above example, VC SpyGlass Lint reports a violation for the UndrivenNUnloaded-ML Tag. Ideally, VC SpyGlass Lint should have reported the UnloadedNet-ML tag violation in this case.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- ***apply_on_lib_cell_only***: The default value is no. Set the value of the parameter to yes check the LIB cells for undriven logic.
- ***checkTopDu***: The default value is no. Therefore, the Tag checks for all DU's. If you set the value of the checkTopDu parameter to Tag name, then the Tag checks instances only at top level. Also, while detecting the driver, the Tag traverses down the hierarchy till it hits any combination, sequential, or black box acting as a driver. Additionally, if you set the value of the checkTopDu parameter to yes, the UndrivenInTerm-ML Tag traverses forward to find a load through the hierarchy. It ignores all inverters and buffers while traversing back and forward.
- ***ignore_module***: The default value is no. Set the value of the parameter to yes to ignore all the RTL buffers within and outside the module boundary.
- ***ignoreRtlBuffer***: The default value is yes. Set the value of the parameter to no to consider the RTL buffers during tag checking.

NOTE: *If the checkTopDu parameter is set then the ignoreRtlBuffer Tag parameter will not be effective.*

- ***checkRTLInst***: The default value is `no`. Set the value of the parameter to `yes` to check for undriven instances generated by VC SpyGlass Lint for RTL assignment descriptions. If AND, OR, NAND, or NOR gates are disabled with constant supply (0 for AND/NAND gate and 1 for OR/NOR gate) on any input, then the UndrivenInTerm-ML Tag does not reports violation for any input terminal for the gate. Also, to check for the constant supply, only buffers are ignored.
- ***checkInHierarchy***: The default value is `yes`. Set the value of the parameter to `no` to not check inside the hierarchy for a valid driver or load.
- ***ignore_unloaded_inst***: The default value is `no`. Set the value of the parameter to `yes` to ignore combinational blocks so that this Tag traverses down the hierarchy to search for a valid load.

NOTE: *Ensure that value of the ***checkInHierarchy*** parameter is set to `yes` to enable the functionality of the ***ignore_unloaded_inst*** parameter.*

- ***report_floating_source***: The default value is `no`. Set the value of the parameter to `yes` to report the floating net source.

NOTE: *Set the value of the ***checkInHierarchy*** parameter to `yes` to enable the functionality of the ***report_floating_source*** parameter.*

- ***handle_greybox***: The default value is `yes` and the UndrivenInTerm-ML Tag considers the input terminal (properly loaded from inside) of module instance connected to the grey box terminal as driven due to its connectivity with the complete grey box. Therefore, the Tag does not report any violation in this case. Set the value of the parameter to `no` to report violation for the net connected to the grey box terminal as undriven or unloaded, depending on the terminal direction.

Also, the UndrivenInTerm-ML Tag reports violation for input terminals that are not driven but are loaded through a stopped module instance when the `handle_greybox` parameter is set to `yes` (default mode).

- ***check_default_value***: This parameter is used for **VHDL**. The default value of the parameter is `yes`. Set the value of the parameter to `no` to enable the UndrivenInTerm-ML Tag to report violation for undriven input terminals of instances that are left open and have a default value in the component port declaration or the entity port declaration.

- ***ignore_inout***: The default value of the parameter is `no`. Set the value of the parameter to `yes` to enable the UndrivenInTerm-ML Tag to ignore the violations for inout terminals.
- ***checkfullbus***: By default, this parameter is set to `no`. Set this parameter to `yes` if you do not want to report violation for vector signals if at least one bit of vector terminal is driven.
- ***ignore_deliberately_unconnected***: The default value is `no`. Set the value of the parameter to `yes` to ignore violations for terminals, which are deliberately unconnected in Verilog.
- ***skip_lib_cell_checking***: By default, this parameter is set to `no`. Set the `skip_lib_cell_checking` parameter to `yes` to not report violations for library cells.
- ***ignore_unused_flop***: By default, this parameter is set to `no`. Set this parameter to `yes` to ignore the basic combinational cells (buffer, inverter, OR, AND, NOR, NAND, XOR, XNOR, MUX) and checks across hierarchy to search for valid load. When a flop or latch is reached, the tag further checks the down the hierarchy until a module boundary is reached to see if the flop/latch output is used. If this parameter is not set, the flop/latch is not considered as a valid load.

NOTE: *The Effect of this parameter will only be seen when checkInHierarchy is set.*

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears at the location of instance containing input terminal `<term-name>` when the input terminal is undriven but loaded:

[WARNING] Detected undriven input terminal `<term-name>`

Potential Issues

Violation may arise when the Tag detects an undriven input terminal.

Consequences of Not Fixing

The following lists various scenarios and consequences of not fixing the violation:

- For a transistor to function as an on-off switch, the gate of a cmos transistor should be at a defined state and at an electrical level of logic, 1 or 0. If the gate of a transistor is left floating, the corresponding transistor can get into a partially on, *limbo*, state.
- In the context of a logic gate, such a *partially on* transistor can lead to an indeterminate logic value and analog electrical state on the output of that gate.
- Following are the consequences, if the input of an inverter is left floating:
 - ? Both the n and p transistors can become *partially on* transistors, thereby creating a steady state path from Vdd to ground. In such cases, the output of a transistor would be in a non-digital state (i.e., somewhere between logic 1 and 0).
 - ? This non-digital state would then propagate to all the input pins of gates in the fan-out cone of the inverter. Those gates, in turn, would produce a non-digital output and would propagate further. Thus, potentially large number of nodes in a chip would be in an indeterminate non-digital state.
 - ? The steady state path created between Vdd and ground through the inverter transistors leads to an excessive current draw from Vdd resulting in a burn-out or latch-up of the chip.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line of module instantiation, for which violation is reported. The violation message contains the name of the undriven input terminal.

Also, the Incremental Schematic of the violation message highlights the undriven input terminal.

To fix the violation, ensure that a valid driver is present for a loaded input terminal.

Message 2

The following message appears at the location of instance containing input terminal `<term-name>` when the net of an input terminal is undriven but loaded:

[WARNING] Detected undriven input terminal `<term-name>` undriven net `<net_name>`

Potential Issues

Violation may arise when the Tag detects an undriven net.

Consequences of Not Fixing

The following lists various scenarios and consequences of not fixing the violation:

- For a transistor to function as an on-off switch, the gate of a cmos transistor should be at a defined state and at an electrical level of logic, 1 or 0. If the gate of a transistor is left floating, the corresponding transistor can get into a partially on, *limbo*, state.
- In the context of a logic gate, such a *partially on* transistor can lead to an indeterminate logic value and analog electrical state on the output of that gate.
- Following are the consequences, if the input of an inverter is left floating:
 - ? Both the n and p transistors can become *partially on* transistors, thereby creating a steady state path from Vdd to ground. In such cases, the output of a transistor would be in a non-digital state (i.e., somewhere between logic 1 and 0).
 - ? This non-digital state would then propagate to all the input pins of gates in the fan-out cone of the inverter. Those gates, in turn, would produce a non-digital output and would propagate further. Thus, potentially large number of nodes in a chip would be in an indeterminate non-digital state.
 - ? The steady state path created between Vdd and ground through the inverter transistors leads to an excessive current draw from Vdd resulting in a burn-out or latch-up of the chip.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights all the instances related to the violating path. The violation message contains the name of the undriven input terminal. It also reports the name of the undriven net, when the value of the [report_floating_source](#) parameter is set to on.

Also, the Incremental Schematic of the violation message highlights the complete path.

To fix the violation, ensure that a valid driver is present for a loaded input terminal.

Example Code and/or Schematic

Example 1

Consider the following example:

```
module M1 (input W1);
M2 modm(w1);
endmodule
```

```
module M2(input in1, in2, output op);
assign op = in1 & in2;
endmodule
```

VC SpyGlass Lint generates the following message for this example:

Detected undriven input terminal M1.modm.in2

Also, consider the following figure:

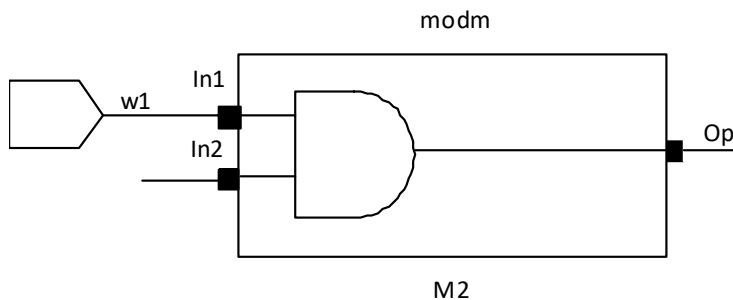


FIGURE 14. Loaded but undriven port

In the above figure, the UndrivenInTerm-ML Tag flags a violation message for In2 port, which is loaded but undriven.

Example 2

Consider the following example:

```
module myFF (q, d, clk);
output q;
input d, clk;
```

MoreLint Tags

```

        FD1 ff(.Q(q), .D(d), .CP(clk));
    endmodule

    module actualtop();
        wire [7:0] op;
        wire [7:0] in2;
        wire [7:0] in1;
        wire clk1,clk2;
        unconLibFF tmp(op, in2, in1, clk1, clk2);
        myFF ff1(,op[5], clk1);
    endmodule
    module unconLibFF(op, in2, in1, clk1, clk2);
        output [7:0] op;
        input [7:0] in1, in2;
        input clk1, clk2;
        oneMore mod(op[5],in1[4]);
    endmodule

    module oneMore(c,d);
        output c;
        input d;
        mybuff check(c,d);
    endmodule

    module mybuff(a,b);
        output a;
        input b;
        wire w1 = b;
        wire w2;
        assign a = w2;
    endmodule

```

The *UndrivenInTerm-ML* Tag reports violation for the above RTL because the net, `actualtop.tmp.mod.check.w2`, is loaded but undriven. Please note that the violation message is reported only when the values of the [checkInHierarchy](#) parameter and [report_floating_source](#) parameter is set to **yes**.

Following is the Incremental Schematic corresponding to the violation

message reported for the above RTL:

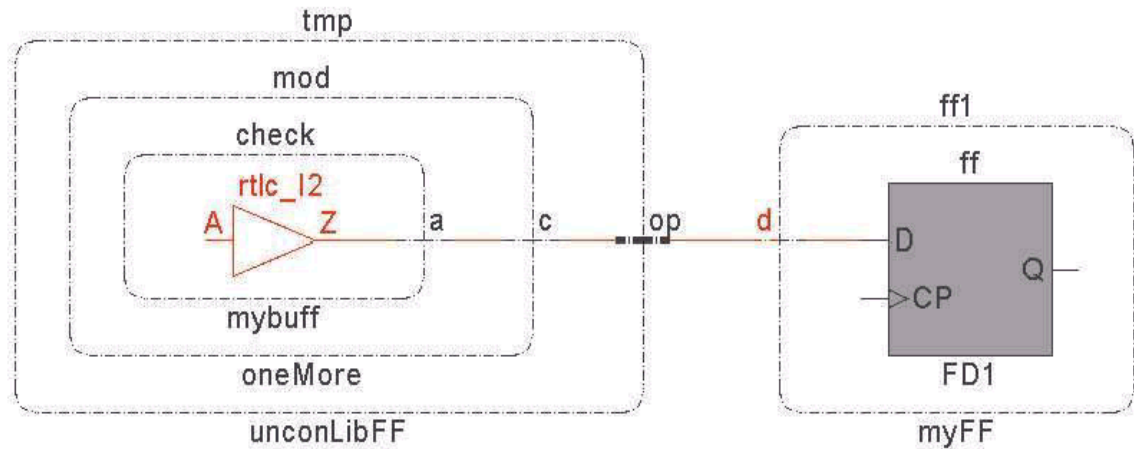


FIGURE 15. Incremental schematic

Example 3

Consider the following example:

```
module test();
  wire x;
  bottom bot(x);
endmodule
module bottom(inout1);
  inout inout1;
  wire b, c;
  assign b = inout1 & c;
endmodule
```

For this example, the *UndrivenInTerm-ML* Tag ignores violations for the inout terminal `inout1` as the *ignore_inout* parameter is set to yes.

Default Severity Label

Warning

MoreLint Tags

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

UndrivenNet-ML

Undriven but loaded net is detected in the design

Language

Verilog, VHDL

Description

The UndrivenNet-ML Tag flags nets that are not driven but are loaded.

The UndrivenNet-ML Tag checks only those nets which are described in a module and have load or driver using RTL construct like assignment statements. For example:

```
out = in1 & in2;
```

In the above example, nets, `in1` and `in2`, have load but no driver.

The UndrivenNet-ML Tag is not applicable for those nets that have load/driver through another module instances (connected to instance input terminal). To check for undriven nets connected to modules instances, use the [UndrivenInTerm-ML](#) Tag.

The UndrivenNet-ML Tag does not report any violation for undriven port nets. For these nets, the [UnloadedInPort-ML](#) and [UndrivenOutPort-ML](#) Tags report violation.

By default, the UndrivenNet-ML Tag checks for all DU's. However, if the `checkTopDu` parameter is set, then the Tag checks only top level nets and while detecting the driver, the Tag traverses through the hierarchy till it detects any combination, sequential, or black box acting as a driver. The Tag reports violation only for top level undriven nets.

The UndrivenNet-ML Tag does not cross boundary of a library cell in any of its traversal. If a net is connected to an input (output) terminal of a library cell, then the net is considered as loaded (driven).

While traversing the design to find the drivers of top-level nets, the UndrivenNet-ML Tag ignores buffers and inverters.

By default, the UndrivenNet-ML Tag considers combinational blocks to search for a valid load. Set the [ignore_unloaded_inst](#) parameter and [checkInHierarchy](#) parameter to yes to ignore the combinational blocks so that this Tag traverses down the hierarchy to search for a valid load.

By default, the UndrivenNet-ML Tag considers the net (properly loaded) connected (driven by) to grey box terminal as driven. Set the value of [handle_greybox](#) parameter to no to consider the net connected to the grey box terminal as undriven.

By default, the [checkInHierarchy](#) Tag parameter is set to yes to traverse through the hierarchy to search for valid driver/load.

Set the [ignore_module](#) Tag parameter to yes to ignore all the RTL buffers within and outside the module boundary.

NOTE: *If the checkTopDu parameter is set then the ignoreRtlBuffer Tag parameter will not be effective.*

For other nets, the first connected output pin of a combinational gate, a sequential cell instance, or a user-instantiated cell is assumed to be the driver.

By default, the UndrivenNet-ML tag ignores the RTL buffers. Set the value of the parameter to [ignoreRtlBuffer](#) parameter to no to consider the RTL buffers during tag checking.

By default, the tag reports violations for multiple instances of the same module, that is, sibling modules. To ignore the sibling violations, set the value of the [ignore_sibling_viol](#) parameter to yes.

By default, the [ignore_unused_flop](#) parameter is set to no. Set this parameter to yes to ignore the basic combinational cells (buffer, inverter, OR, AND, NOR, NAND, XOR, XNOR, MUX) and checks across hierarchy to search for valid load. When a flop or latch is reached, the tag further checks the down the hierarchy until a module boundary is reached to see if the flop/latch output is used. If this parameter is not set, the flop/latch is not considered as a valid load.

Message Details

The following message appears at the location where net `<net-name>` is first used when the net is undriven but loaded:

Detected undriven net `<net-name>`

Severity

Warning

Examples

Example 1

Consider the following example:

```
module internalNet (op, in1, in2);
    output [3:0] op;
    input  [3:0] in1, in2;

    wire [7:0] internalNet;

    assign op[0] = in1[0] & in2[0];
    assign op[1] = in1[1] & in2[1];
    assign op[2] = in1[2] & in2[2];
    assign op[3] = internalNet[2] & internalNet[3];
endmodule
```

The UndrivenNet-ML Tag reports the following warning message for the above example:

Detected undriven net internalNet.internalNet[3:2]

Example 2

Consider the following example:

```
module test(tap_spare2_out,mode_control_bsr_scanout);
    output tap_spare2_out;
    input mode_control_bsr_scanout;
    assign mode_control_bsr_scanout = tap_spare2_out;
endmodule
```

In the above example, the UndrivenNet-ML Tag does not report violation for tap_spare2_out and mode_control_bsr_scanout. However, the [UnloadedInPort-ML](#) and [UndrivenOutPort-ML](#) Tags would report violation for the above example.

UndrivenNUnloaded-ML

Undriven and Unloaded nets/terminals detected in the design

Language

Verilog, VHDL

Description

The UndrivenNUnloaded-ML Tag flags nets and terminals that are neither driven nor loaded.

See the [UndrivenNet-ML](#) and [UnloadedNet-ML](#) Tags for description of how net drivers and net loads are inferred. See the [UndrivenInTerm-ML](#) and [UnloadedOutTerm-ML](#) Tags for description of how terminal drivers and terminal loads are inferred.

By default, the tag reports violation for the output ports. Set the value of the [check_input_port](#) parameter to yes to disable violation on output ports.

To ignore all the RTL buffers within and outside the module boundary, set the [ignore_module](#) tag parameter to yes.

CHECKSIBLING: The default value is yes. Set the value of the parameter to no to enable the tag to report the sibling hierarchies also.

NOTE: *If the checkTopDu parameter is set then the ignoreRtlBuffer Tag parameter will not be effective.*

The UndrivenNUnloaded-ML Tag does not cross boundary of a library cell in any of its traversal. If a net is connected to an input (output) terminal of a library cell, then the net is considered as loaded (driven).

NOTE: *Results of Tags, such as UndrivenInTerm-ML, UnloadedNet-ML, UndrivenNUnloaded-ML, and checkNetDriver may get interchanged for signals that are explicitly assigned the value of 0 or 1 and are not driving anything. As a result, when a particular violation is expected, some other violation may be reported. Consider the following example:*

```
assign b = 1'b0; // "b" is not even being used
```

For the above example, VC SpyGlass Lint reports a violation for the UndrivenNUnloaded-ML Tag. Ideally, VC SpyGlass Lint should have reported the UnloadedNet-ML Tag violation in this case.

By default, the UndrivenNUnloaded-ML Tag traverses down the hierarchy to search for valid driver/load. Set the [checkInHierarchy](#) Tag parameter to no to avoid checking inside the hierarchy for a valid driver or load.

By default, the UndrivenNUnloaded-ML Tag considers combinational blocks to search for a valid load. Set the [ignore_unloaded_inst](#) parameter and [checkInHierarchy](#) parameter to yes to ignore the combinational blocks so that this Tag traverses down the hierarchy to search for a valid load.

By default, the UndrivenNUnloaded-ML Tag does not report violation for output terminals (undriven from inside) of a module instance that are loaded by a grey box. The output terminals are considered as loaded (though undriven) due to its connectivity with complete grey box. However, if you set the value of the [handle_greybox](#) parameter to no, the UndrivenNUnloaded-ML Tag reports violation for output terminals of a module instance that are loaded by a grey box.

By default, the *UndrivenNUnloaded-ML* tag ignores the RTL buffers. Set the value of the parameter to [ignoreRtlBuffer](#) parameter to no to consider the RTL buffers during tag checking.

By default, the *UndrivenNUnloaded-ML* tag performs tag checking on nets. Set the value of the parameter [ignore_nets](#) to disable violations on nets.

By default, the *ndrivenNUnloaded-ML* tag does not report violation for all the nets. Set the value of the [reportallnets](#) to disable violations on nets.

The UndrivenNUnloaded-ML tag will not report violations for the last fanout port if the connected outside hierarchy signal is violated. This will reduce noise for this rule.

In UndrivenNUnloaded-ML, the CheckSibling callback is enabled for the entire flow. It can report multiple violations from different hierarchies.

In UndrivenNUnloaded-ML, the parameter EFFORT_LEVEL default value is 100000.

VHDL

By default, the UndrivenNUnloaded-ML Tag does not report violation, if the input/output terminals of an instance are left open intentionally using OPEN keyword and are also unloaded/undriven. Set the value of [report_mark_open](#) parameter to yes to report such violations.

Message Details

The following message appears at the location where net *<net-name>* is first used when the net is neither driven nor loaded:

Detected undriven and unloaded(unconnected) net <net-name>
 The following message appears at the location of instance containing terminal <term-name> when the terminal is neither driven nor loaded:
 Detected undriven and unloaded(unconnected) terminal <term-name>

Severity

Warning

Examples

Verilog

Consider the following example:

```
module top(in1,in2);
  wire [6:0] out1;
  input in1,in2;

  assign out1[1]= in1 & in2;

endmodule
```

In the above example, the *UndrivenNUnloaded-ML* Tag reports a violation as the out1[6:2, 0] is neither driven nor loaded (unused). The following message is reported by this example:

Detected undriven and unloaded(unconnected) net top.out1[6:2, 0]

VHDL

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

entity top is
  port (in1,in2: in Bit) ;
end entity top ;
```

```
architecture bev of top is
  signal out1:bit_vector (6 downto 0);
begin
  out1(1)<=in1 and in2;
end architecture bev;
```

In the above example, the *UndrivenNUnloaded-ML* Tag reports a violation because the out1[6:2, 0] is neither driven nor loaded (unused). The following message is reported by this example:

```
Detected undriven and unloaded(unconnected) net top.out1[6:2,
0]
```

UndrivenOutPort-ML

Undriven but loaded output port of a module detected

Language

Verilog, VHDL

Description

The UndrivenOutPort-ML Tag flags output ports that are not driven but are loaded.

An output port is assumed to be loaded if it is connected to input pin of a combinational gate, a sequential cell instance, or a user-instantiated cell. The top-level output ports are assumed to be loaded.

The UndrivenOutPort-ML Tag does not cross boundary of a library cell in any of its traversal. If a net is connected to an input (output) terminal of a library cell, then the net is considered as loaded (driven).

While traversing the design to find the drivers of top-level output ports, the UndrivenOutPort-ML Tag ignores buffers.

Set the [ignore_module](#) Tag parameter to `yes` to ignore all the RTL buffers within and outside the module boundary.

NOTE: *If the `checkTopDu` parameter is set then the `ignoreRtlBuffer` Tag parameter will not be effective.*

For other output ports, the first connected output pin of a combinational gate, a sequential cell instance, or a user-instantiated cell is assumed to be the driver.

By default, the UndrivenOutPort-ML Tag traverses down the hierarchy to search for valid driver/load. If you set the [checkInHierarchy](#) Tag parameter to `no`, the UndrivenOutPort-ML Tag does not check inside the hierarchy.

By default, the UndrivenOutPort-ML Tag considers combinational blocks to search for a valid load. Set the [ignore_unloaded_inst](#) parameter and [checkInHierarchy](#) parameter to `yes` to ignore the combinational blocks so that this Tag traverses down the hierarchy to search for a valid load.

By default, the UndrivenOutPort-ML Tag does not report violation for output ports of a module driven by a grey box. The output port is considered as driven due to its connectivity with complete grey box. Set the value of the [handle_greybox](#) parameter to `no` to report violation for output ports of a

module driven by a grey box.

By default, the UndrivenOutPort-ML Tag considers output ports of a grey box module as driven. However, if the value of [handle_greybox](#) Tag parameter is set to no, then the UndrivenOutPort-ML Tag reports output ports of grey box modules as undriven.

By default, the *UndrivenOutPort-ML* tag ignores the RTL buffers. Set the value of the parameter to [ignoreRtlBuffer](#) parameter to no to consider the RTL buffers during tag checking.

Message Details

The following message appears at the location where output port *<port-name>* is declared when the output port is undriven but loaded:

Detected undriven output port <port-name>

Severity

Warning

UndrivenOutTermNLoaded-ML

Undriven output pins connected to instance input

Language

Verilog, VHDL

Description

The UndrivenOutTermNLoaded-ML Tag flags input port of a module that is connected to an undriven but loaded output port of another module as shown in the following figure:

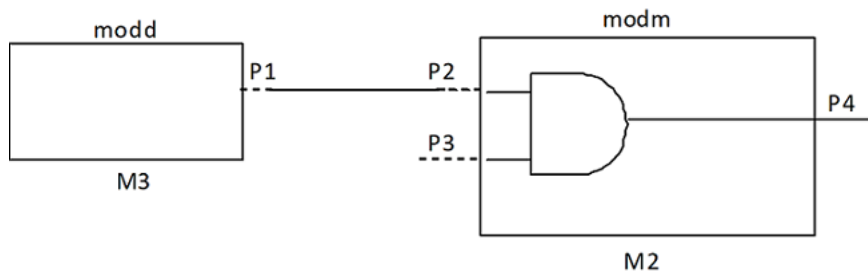


FIGURE 16. Undriven but loaded output port

In the above figure, the UndrivenOutTermNLoaded-ML Tag flags a violation message for output port P1, which is loaded but undriven.

By default, the UndrivenOutTermNLoaded-ML Tag considers combinational blocks to search for a valid load. Set the [ignore_unloaded_inst](#) parameter and [checkInHierarchy](#) parameter to `yes` to ignore the combinational blocks so that this Tag traverses down the hierarchy to search for a valid load.

By default, the UndrivenOutTermNLoaded-ML Tag considers the output ports of grey box modules as driven. If the value of [handle_greybox](#) Tag parameter is set to `no`, then the UndrivenOutTermNLoaded-ML Tag reports output ports of grey box modules as undriven, if they are loaded from outside.

Message Details

The following message appears at the location where the input terminal

<input-terminal-name> of a module is connected to an undriven output terminal *<output-terminal-name>* of another module:

Input terminal '*<input-terminal-name>*' is connected to undriven output terminal '*<output-terminal-name>*'

Severity

Warning

Example

Consider the following example

```
module M1 ();  
wire w1;  
M3 modd(w1);  
M2 modm(w1);  
endmodule
```

```
module M2(input in1, in2, output op);  
assign op = in1 & in2;  
endmodule
```

```
module M3 (output op);  
wire w2;  
endmodule
```

VC SpyGlass Lint generates the following violation message for this example:

Input terminal M1.modm.in1 is connected to undriven output terminal M1.modd.op

UniqueInputOutputSampling-ML

Checks the existence of unique output flip-flops to output interface and unique input flip-flops to input interface

When to Use

Not applicable

Description

This Tag reports violations for output ports and input ports that have no unique flip-flop sampling at the top level. For input ports, sampling should be at D pin of a unique flip-flop. Output ports sampling should be from Q pin of a unique flip-flop. Inout ports are not considered for Tag violation.

When flip-flops are considered for sampling, other elements such as latches and combo elements are not considered by this Tag.

While fan-out is travelling for input ports, all the generated buffers, inverters and sync AND gates are ignored by this Tag.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears input/output port is not sampled uniquely:

[WARNING] <input|output> port '<port-name>' is not sampled uniquely

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

View the Incremental Schematic of the violation message. For input ports, all the instances and all the nets are highlighted from input port to the improperly sampled design objects in the fan-out. Similarly, for output ports, complete schematic is highlighted from the output flip-flop to various design objects in the fan-out of the output flip-flop. Review the design to resolve the sampling to be unique.

Example Code and/or Schematic

Example 1

In the following example, the input port is improperly sampled to D pins of two flip-flops. Therefore, a violation message is reported for the input port in:

```
module top(in, clk, out1, out2);

input in, clk;
output out1, out2;
reg q1, q2;

assign out1 = q1;
assign out2 = q2;

always @ (posedge clk)
    q1 = in;

always @ (posedge clk)
    q2 = in;

endmodule
```

Example 2

In the following example, the Q pin of flip-flop is improperly sampled to two different output ports. Therefore a violation is reported for the output ports out1 and out2.

MoreLint Tags

```
module top(in, clk, out1, out2);  
  input in, clk;  
  output out1, out2;  
  reg q;  
  
  assign out1 = q;  
  assign out2 = q;  
  
  always @ (posedge clk)  
    q = in;  
  
endmodule
```

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

UnloadedInPort-ML

Unloaded but driven input port of a module detected

Language

Verilog, VHDL

Description

The *UnloadedInPort-ML* Tag reports input ports that are driven but are not loaded.

An input port is assumed to be driven if it is connected to the output pin of a combinational gate, a sequential cell instance, or a user-instantiated cell. The top-level input ports are assumed to be driven.

The Tag does not cross boundary of a library cell in any of its traversal. If a net is connected to an input (output) terminal of a library cell, the net is considered as loaded (driven).

While traversing the design to find the load of top-level input ports, the Tag ignores buffers.

Set the *ignore_module* parameter to *yes* to ignore all the RTL buffers within and outside the module boundary.

NOTE: *If the *checkTopDu* parameter is set, the *ignore_module* parameter will not be effective.*

For other input ports, the first connected input pin of a combinational gate, a sequential cell instance, or a user-instantiated cell is assumed to be the load.

By default, the Tag traverses down the hierarchy to search for valid driver or load. The *checkInHierarchy* parameter is set to *yes* to enable the UnloadedInPort-ML Tag check inside the hierarchy.

By default, the *UnloadedInPort-ML* Tag considers combinational blocks to search for a valid load. Set the *ignore_unloaded_inst* parameter and *checkInHierarchy* parameter to *yes* to ignore the combinational blocks so that this Tag traverses down the hierarchy to search for a valid load.

By default, the Tag does not report violation for input ports of a module that are loaded by a grey box. The input port is considered as loaded due to its connectivity with the complete grey box. Set the *handle_greybox* parameter to *no* to report a violation for input ports of a module that are

loaded by a grey box.

By default, the Tag considers the input ports of grey box modules as loaded and does not report a violation in this case. Set the [handle_greybox](#) parameter to `no` to consider the input ports of grey box modules as unloaded and report a violation for the same.

By default, the [report_hier_path](#) parameter is set to `yes` and the Tag reports the complete hierarchical path in the violation message. If you set this parameter to `no`, only the unloadedInPort terminal is reported, which could lead to merging of violations from different modules based on the same port names.

By default, the *UnloadedInPort-ML* tag ignores the RTL buffers. Set the value of the parameter to [ignoreRtlBuffer](#) parameter to `no` to consider the RTL buffers during tag checking.

By default, the [check_only_top_module](#) parameter is set to `no` and the *UnloadedInPort-ML* tag reports violation for all driven but unloaded ports of an input port. Set the value of the parameter to `yes` to report violation for top-level port.

Message Details

The following message appears at the location where the input port `<port-name>` is declared when the input port is driven but not loaded:

Detected unloaded(unconnected)input port <port-hierarchy>

Where, `<port-hierarchy>` represents complete hierarchy of the input port `<port-name>`.

Severity

Warning

Examples

Verilog

Example 1

Consider the following example:

```
module foCountZero(op, in1, in2);
    output [3:0]    op;
```

```
input  [3:0]  in1, in2;  
  
    wire w1 = in1[2] & in2[2];  
endmodule
```

For this example, VC SpyGlass Lint generates the following violation messages:

Detected unloaded(unconnected)input port foCountZero.in2[3, 1:0]

Detected unloaded(unconnected)input port foCountZero.in1[3, 1:0]

The schematic for the above example is shown below:

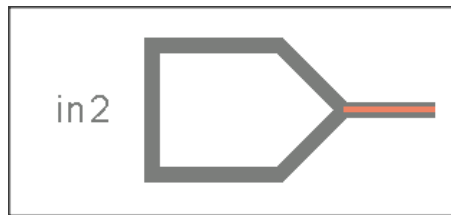


FIGURE 17. Incremental schematic

The following are the color legends for this schematic:

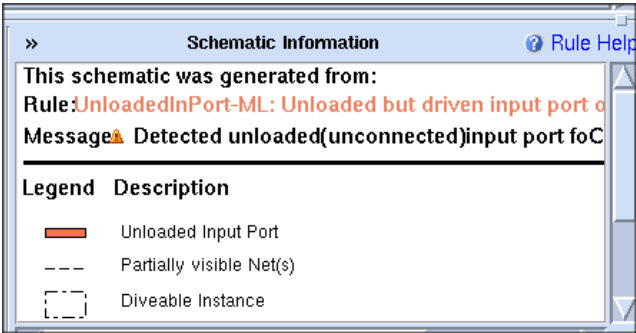


FIGURE 18. Legends for the incremental schematic

Example 2

Consider the following example:

```
//file top.v
`include "sTag1.v"
`define WIDTH 4

module topunit (clk, rst, din, dout);

input  clk, rst;
input  [3:0] din;
output [3:0] dout;

wire [`WIDTH-1:0] sint5a, sint2a;
wire [2-1:0] sint5b, sint2b;

//fully connected
sTag1 # (4) i1(.ck(clk), .re(rst), .d(din[3:0]), .q(sint2a),
.q2(sint5a));
sTag1 # (2) i2(.ck(clk), .re(rst), .d(din[3:0]), .q(sint2b),
.q2(sint5b));
sTag1 # (2) i3(.ck(clk), .re(rst), .d(din[3:0]), .q(sint2b),
.q2(sint5b));
```

```

assign dout = sint2a | sint2b | sint5a | sint5b;
endmodule
//end of file top.v

//file sTagel.v
module sTagel (ck, re, d, q, q2);

parameter n = 2;
input  ck, re;
 [4:0] d;    //forcefully wide connection
output [n-1: 0] q, q2;

    wire [n-1: 0] q2;
    reg [n-1: 0] q;

always @(posedge ck or negedge re)
begin: ff
    if (re == 1'b0)
        q <= 2'b0;
    else
        q <= d[n-1:0];
    end
endmodule
//end of file sTagel.v

```

For this example, VC SpyGlass Lint generates the following violation message:

Detected unloaded(unconnected)input port topunit.i2.d[3:2]

Example 3

Consider the following example:

```

module bbox (a, b);
    parameter p = 0;
    input [3:0] a;
    output [3:0] b;

    assign b[p] = a[p];
endmodule

```


MoreLint Tags

```

module top(in,out);
    input [7:0] in;
    output [7:0] out;
    wire a,f;
    bbox b1 (.a(in[3:0]), .b(out));
    bbox #(3)b2 (.a(in[7:4]), .b(out));
endmodule

```

For this example, if the [report_hier_path](#) parameter is set to yes, the following violation messages are reported, showing the complete hierarchical path:

Detected unloaded(unconnected)input port top.b2.a[2:0]

Detected unloaded(unconnected)input port top.b1.a[3:1]

For the above example code, if the [report_hier_path](#) parameter is set to no, the following violation message is reported:

Detected unloaded(unconnected)input port a[3:0]

VHDL

Consider the following example:

```

library ieee;
use ieee.std_logic_1164.all

entity top is
port (in1,in2: in bit_vector(3 downto 0);
      op: out bit) ;
end entity top ;

architecture bev of top is
begin
    op<=in1(2) and in2(2);
end architecture bev;

```

In the above example, the *UnloadedInPort-ML* Tag reports a violation because the in1[3, 1:0] and in2[3, 1:0] are driven but are not loaded.

The following messages are reported by this example:

Detected unloaded(unconnected)input port top.in2[3, 1:0]

Detected unloaded(unconnected)input port top.in1[3, 1:0]

UnloadedNet-ML

Unloaded but driven net detected in the design

Language

Verilog, VHDL

Description

The UnloadedNet-ML Tag flags nets that are driven but are not loaded.

The UnloadedNet-ML Tag checks **only** those nets which are described in a module and have driver using RTL construct like assignment statements, but do not have load. For example:

```
out = in1 & in2;
```

In the above example, the net, out, has driver.

NOTE: Tag UnloadedNet-ML also flags nets from output of a flip-flop that are being fed to the flip-flops own input, but are actually not being used in the design.

The UnloadedNet-ML Tag is not applicable for those nets that are driven through another module instances (connected to instance output terminal). To check for unloaded nets connected to modules instances, use the [UnloadedOutTerm-ML](#) Tag.

The UnloadedNet-ML Tag does not report any violation for unloaded port nets. For these nets, the [UnloadedInPort-ML](#) and [UndrivenOutPort-ML](#) Tags report violation.

NOTE: Results of Tags, such as UndrivenInTerm-ML, UnloadedNet-ML, UndrivenNUnloaded-ML, and checkNetDriver may get interchanged for signals that are explicitly assigned the value of 0 or 1 and are not driving anything. As a result, when a particular violation is expected, some other violation may be reported. Consider the following example:

```
assign b = 1'b0; // "b" is not even being used
```

For the above example, VC SpyGlass Lint reports a violation for the UndrivenNUnloaded-ML Tag. Ideally, VC SpyGlass Lint should have reported the UnloadedNet-ML Tag violation in this case.

A net is assumed to be driven if it is connected to output pin of a combinational gate, a sequential cell instance, or a user-instantiated cell in the parent design unit. When the checkTopDu parameter is set, the UnloadedNet-ML Tag traverses across the hierarchy (ignoring buffers and inverters), if required to find the connected output pins for top-level nets.

While traversing the design to find the loads of top-level nets, the UnloadedNet-ML Tag ignores buffers and inverters.

Set the *ignore_module* Tag parameter to *yes* to ignore all the RTL buffers within and outside the module boundary.

NOTE: *If the checkTopDu parameter is set then the ignoreRtlBuffer Tag parameter will not be effective.*

For other nets, the first connected input pin of a combinational gate, a sequential cell instance, or a user-instantiated cell is assumed to be the load.

The UnloadedNet-ML Tag does not cross boundary of a library cell in any of its traversal. If a net is connected to an input (output) terminal of a library cell, then the net is considered as loaded (driven).

By default, the UnloadedNet-ML Tag traverses down the hierarchy to search for valid driver/load. Set the *checkInHierarchy* Tag parameter to *no* to avoid checking inside the hierarchy for a valid driver or load.

By default, the UnloadedNet-ML Tag considers combinational blocks to search for a valid load. Set the *ignore_unloaded_inst* parameter and *checkInHierarchy* parameter to *yes* to ignore the combinational blocks so that this Tag traverses down the hierarchy to search for a valid load.

By default, the UnloadedNet-ML Tag considers the net (properly driven) connected (loaded through) to the grey box terminal as loaded. Set the value of *handle_greybox* parameter to *no* to consider the net connected to the grey box terminal as unloaded.

By default, the *UnloadedNet-ML* tag does not consider constants/supply as driver for a net. Set the value of the *CONSIDER_SUPPLY_AS_DRIVER* to *yes* to consider constant/supply as driver for a net.

By default, the *UnloadedNet-ML* tag ignores the RTL buffers. Set the value of the parameter to *ignoreRtlBuffer* parameter to *no* to consider the RTL buffers during tag checking.

Message Details

The following message appears at the location where net *<net-name>* is first used when the net is driven but not loaded:

Detected unloaded(unconnected) net *<net-name>*

Severity

Warning

Example

Consider the following example:

```
module test(tap_spare2_out,mode_control_bsr_scanout);  
  output tap_spare2_out;  
  input mode_control_bsr_scanout;  
  assign mode_control_bsr_scanout = tap_spare2_out;  
endmodule
```

In the above example, the *UnloadedNet-ML* Tag does not report violation for `tap_spare2_out` and `mode_control_bsr_scanout`. However, the *UnloadedInPort-ML* and *UndrivenOutPort-ML* Tags would report violation for the above example.

UnloadedOutTerm-ML

Unloaded but driven output terminal of an instance detected

Language

Verilog, VHDL

Description

The UnloadedOutTerm-ML Tag flags nets and terminals that are not loaded.

An output terminal is assumed to be driven if it is connected to output pin of a combinational gate, a sequential cell instance, or a user-instantiated cell in the parent design unit. When the `checkTopDu` parameter is set, the UnloadedOutTerm-ML Tag traverses across the hierarchy (ignoring buffers and inverters), if required to find the connected output pins for the input terminals of top-level instances.

The UnloadedOutTerm-ML Tag does not cross boundary of a library cell in any of its traversal. If a net is connected to an input (output) terminal of a library cell, then the net is considered as loaded (driven).

While traversing the design to find the loads of output terminals of top-level instances, the UnloadedOutTerm-ML Tag ignores buffers and inverters.

Set the *ignore_module* Tag parameter to *yes* to ignore all the RTL buffers within and outside the module boundary.

NOTE: *If the `checkTopDu` parameter is set then the `ignoreRtlBuffer` Tag parameter will not be effective.*

For other output terminals, the first connected input pin of a combinational gate, a sequential cell instance, or a user-instantiated cell is assumed to be the load.

By default, the UnloadedOutTerm-ML Tag considers combinational blocks to search for a valid load. Set the *ignore_unloaded_inst* parameter and *checkInHierarchy* parameter to *yes* to ignore the combinational blocks so that this Tag traverses down the hierarchy to search for a valid load.

By default, the UnloadedOutTerm-ML does not report violation for output terminal (properly driven from inside) of a module instance that is loaded by a grey box. The output terminal is considered as loaded due to its connectivity with the complete grey box. However, if you set the value of

the [handle_greybox](#) parameter to no, the UnloadedOutTerm-ML Tag reports violation for output terminal of a module instance that is loaded by a grey box.

Also, the UnloadedOutTerm-ML Tag reports violation for output terminals that are not loaded but are driven through a stopped module instance when the [handle_greybox](#) parameter is set to yes (default mode).

For VHDL, by default, if the output terminals of an instance are left open intentionally using the OPEN keyword, then the UnloadedOutTerm-ML Tag does not report any violation. Set the value of [report_mark_open](#) parameter to yes to flag these violations.

By default, the Tag reports violations for terminals, which are deliberately unconnected in Verilog. Set the value of the [ignore_deliberately_unconnected](#) parameter to yes to ignore violations for terminals, which are deliberately unconnected in Verilog.

By default, the UnloadedOutTerm-ML tag ignores the RTL buffers. Set the value of the parameter to [ignoreRtlBuffer](#) parameter to no to consider the RTL buffers during tag checking.

By default, the UnloadedOutTerm-ML tag reports violation for terminals, which are deliberately unconnected in Verilog. When the value of the [ignore_deliberately_unconnected](#) parameter is set to yes, the UnloadedOutTerm-ML tag does not report violations for such cases.

By default, the *UnloadedOutTerm-ML* tag reports a violation for all the output terminals of an instance that does not have a load but has a driver. Set the value of the [ignore_out_term](#) parameter to a comma or space-separated list of PERL regular expression to enable the tag to not report a violation for the specified output terminals.

To perform checks only for LIB cells, set the value of the [only_apply_on_leaf_cell](#) parameter to yes.

Message Details

The following message appears at the location of instance containing output terminal *<term-name>* when the output terminal is left open:

Detected unloaded(unconnected) output terminal *<term-name>*
(Left open)

The following message appears at the location of instance containing output terminal *<term-name>* when the output terminal is connected to

MoreLint Tags

floating net *<net-name>*:

Detected unloaded(unconnected) output terminal *<term-name>*
(Connected to floating net '*<net-name>*')

For VHDL, the following message appears at the location of instance containing output terminal *<term-name>* when the output terminal is intentionally left open using the OPEN keyword:

Detected unloaded(unconnected) output terminal *<term-name>*
(OPEN)

Severity

Warning

CheckKeywordsOfCaseStmt-ML

tag will check the usage of unique/unique0/priority keyword with case, casez, casex and case inside statements

Description

The CheckKeywordsOfCaseStmt-ML tag checks if unique/unique0/priority keyword is used with case, casez, casex and case inside statements. By default, the tag reports violations for any type of case statement (case, casez, casex, case inside) without any prefix keywords (unique/unique0/priority).

Use the following parameters to mention the keyword to be used with the type of case statement. The tag reports violations, if you use any other keyword other than these unique/unique0/priority keywords.

- case_option
- casez_option
- casex_option
- case_inside_option

Language

Verilog

Parameter(s)

None

Constraint(s)

- [*case_option*](#): Specifies the keyword that must be used with case statements.
- [*casez_option*](#): Specifies the keyword that must be used with casez statements.
- [*casex_option*](#): Specifies the keyword that must be used with casex statements.
- [*case_inside_option*](#): Specifies the keyword that must be used with case inside statements.

Messages and Suggested Fix

The following message appears when the `unique/unique0/priority` keyword is not used with the case statement.

One of important keywords must be used with case statement with expression `<case expression> [Hierarchy:<hierarchy>]`

Example Code and/or Schematic

Consider the following example:

Example 1:

```
logic [3:0] y;  
logic [1:0] a;  
logic      en;  
always_comb begin  
    y = '0;  
    case ({en,a})  
        3'b100: y[a] = 1'b1;  
        3'b101: y[a] = 1'b1;  
        3'b110: y[a] = 1'b1;  
        3'b111: y[a] = 1'b1;  
    endcase  
end
```

By default, the tag reports a violation.

Example 2:

```
logic [3:0] y;  
logic [1:0] a;  
logic      en;  
always_comb begin  
    y = '0;
```

```

priority case ({en,a})
  3'b100: y[a] = 1'b1;
  3'b101: y[a] = 1'b1;
  3'b110: y[a] = 1'b1;
  3'b111: y[a] = 1'b1;
endcase
end

```

By default, the tag does not report a violation for the case statement.

- If the PRIORITY value is given to the `case_options` parameter, the tag does not report violation.
- If only the UNIQUE value is given to the `case_options` parameter, then the tag reports violation.

Example 3:

```

always @(*)
begin
  priority case (tmpSig) inside
    3'b00? : q<=1;
    3'b0?0 : q<=2;
    3'b?00 : q<=3;
  endcase
end

```

By default, the tag does not report a violation for the case statement.

- If the PRIORITY value is given to the `case_inside_options` parameter, the tag does not report violation.
- If only the UNIQUE0 value is given to the `case_inside_options` parameter, then the tag reports violation.

Default Severity Label

Error

MoreLint Tags

tag Group

Not Applicable

Reports and Related Files

No reports or related files.

ComplexModport-ML

Detects complex modport declarations

Description

The *ComplexModport-ML* tag reports violations for the following scenarios of complex modport declarations that are not supported by DC.

- Modport declaration inside generate block.
- Output ports of modport that use concatenation or multiple concatenation.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following messages are reported when complex modport declarations are detected inside generate block and the output ports of the modport contains concatenation respectively:

Modport <modport name> of interface <interface name> is declared inside generate block.

Modport <modport name> of interface <interface name> contains concatenation in output port <port name>

Potential Issues

The violation message states the issue.

Consequences of Not Fixing

Not fixing the violation can create issues when running DC.

Example Code and/or Schematic

Example 1

In the following example, a violation is reported since the modport declaration is detected inside generate block:

```
interface inf (input clk, rst, W_R, output addr);
logic [3:0] req, gnt;
logic clk_new, reset;
generate
    for (genvar i=0; i< num_agents; i++) begin: arb
        modport P2 (input .req (req[i]), clk_new, reset,
output .gnt (gnt[i]) ); // Violation since Modport
declaration inside generate block
    end
endgenerate
endinterface
```

Example 2

In the following example, no violation is reported since the modport declaration is not detected inside generate block:

```
interface inf (input clk, rst, W_R, output addr);
logic [3:0] req, gnt;
logic clk_new, reset;
modport P2 (input .req (req + 3), clk_new, reset, output .gnt
(gnt[0]) ); // No violation
endinterface
```

Example 3

In the following example, a violation is reported since the output port contains concatenation:

```
interface arb_if (input clk_new);
parameter num_agents =3;
```

```

    logic reset;
    logic [num_agents-1:0] req;
    logic [num_agents-1:0] gnt;
    modport P2 (input .req (req + 3), clk_new, reset, output .gnt
    ({gnt[0],gnt[1]}) ); // Violation since output port with
    concatenation
endinterface

```

Example 4

In the following example, a violation is reported since the output port contains multiple concatenation:

```

interface arb_if (input clk_new);
parameter num_agents =3;
    logic reset;
    logic [num_agents-1:0] req;
    logic [num_agents-1:0] gnt;
    modport P2 (input .req (req + 3), clk_new, reset, output .gnt
    ({2{gnt[0]}}) ); // Violation since output port with
    multiple concatenation
endinterface

```

Example 5

In the following example, no violation is reported since the input port contains concatenation:

```

interface arb_if (input clk_new);
parameter num_agents =3;
    logic reset;
    logic [num_agents-1:0] req;
    logic [num_agents-1:0] gnt;
    modport P2 (input .req ({req[0],req[1]}), clk_new, reset,
    output .gnt (gnt) ); // No Violation since concatenation is on input
    port declaration

```

MoreLint Tags

endinterface

Default Severity Label

Warning

tag Group

morelint

Reports and Related Files

No related reports or files.

ConcatUnsizeNumbers-ML

tag will report violations for concatenation of unsized numbers

Language

Verilog

Description

The *ConcatUnsizeNumbers-ML* tag reports violation when a concatenation contains an unsized constant. The compiler cannot calculate the correct size of the concatenation. The size is assumed to be 32 bits.

Parameter(s)

None

Constraints

None

Message Details

The following message appears when concatenation contains an unsized constant:

Concatenations cannot have unsized number '<value>' [Hierarchy: '< hierarchy >']

Potential Issues

The violation message is reported, when concatenation contains an unsized constant.

Consequences of Not Fixing

Not fixing the violation may result in unexpected design behavior.

How to Debug and Fix

To debug the violation, click the violation message. SpyGlass highlights the violating signal in the HDL viewer.

To fix the violation, ensure that the concatenation does not contain unsized constant.

Examples

Consider the following example where the *ConcatUnsizeNumbers-ML* tag reports a violation:

```
module test();

parameter W1 = 1;
parameter W2 = 3'b1;
parameter int W3 = 1;
parameter [3:0] W4 = 1;
parameter W5 = 'b10101;

wire a;
wire b;
wire c;

assign a = { b , 1 } ;      // violation
assign b = { a , 'b101 } ; // violation
assign b = { a , 3'b101 } ; // no violation

assign c = { b , W1 } ; //violation
assign c = { b , W2 } ; // no violation
assign b = { a , W3 } ; // no violation
assign b = { a , W4 } ; // no violation
assign b = { a , W5 } ; //violation

endmodule
```

Default Severity Label

Warning

tag Group

morelint

Reports and Related Files

No related reports or files.

DetectNonBlockGenFor-ML

For-loop inside generate statements must have a begin/end block

Description

The DetectNonBlockGenFor-ML tag reports violation if a begin/end block is not specified in a `for` loop inside a `generate` statement.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following violation message appears at the line where the violation has occurred:

For loop <For loop statement> does not have a begin-end block.

The violation message must point to the file and the line where the violation has occurred:

- If the definition is present in one of the input RTL files, the violation message points to the file and line number where the definition of identifier is found.
- If the definition is present in one of the include file, the violation message points to the file and line number where the include command is found.

How to Debug and Fix

Click the violation message in the console line. SpyGlass points to the line where the violation has occurred (file and line number are reported in the violation). Specify the begin/end block in the `for` loop inside the `generate` statement.

Example Code and/or Schematic

Consider the following example:

Example 1:

```
module test;
    genvar i;
    generate
        for (i=0; i < 4; i=i+1) //Violation - No begin-end found
            test_inst inst();
        endgenerate
    endmodule
```

```
module test_inst;
endmodule
```

In the above example, the DetectNonBlockGenFor-ML tag reports a violation as the begin-end block is not specified in the `for` loop inside the `generate` statement.

Example 2:

```
//test.v
module test;
    genvar i;
    generate
        for (i=0; i < 4; i=i+1) begin: MEM //No Violation
            test_inst inst();
        end
    endgenerate
endmodule
```

MoreLint Tags

```
module test_inst;  
endmodule//test.v end
```

In the above example, the DetectNonBlockGenFor-ML tag does not report a violation as the begin-end block is specified in the `for` loop inside the `generate` statement.

Example 3:

```
module test();  
integer i;  
reg [7:0] ram [0:255];
```

```
initial begin  
    for (i = 0; i < 256; i = i + 1) //No Violation - for loop not inside  
generate statement  
        ram[i] <= 0;  
    end  
endmodule
```

In the above example, the DetectNonBlockGenFor-ML tag does not report a violation as the `for` loop is not specified inside the `generate` statement.

Default Severity Label

Warning

tag Group

Not Applicable

Reports and Related Files

No related reports or files.

DirectTopInputToInout-ML

Checks the connection between top input and inout ports

When to Use

Use this tag to check the connectivity of top input ports.

Description

The *DirectTopInputToInout-ML* tag reports connections where top-level input is connected to inout port.

If the top-level input port is directly connected (ignoring SpyGlass generated buffers and inverters) to inout port of any instance, which is instantiated inside top module, then a violation is reported.

Only SpyGlass generated buffers/inverters are ignored to check the connection between input port and inout port. The tag checking is stopped when it comes across any other combinatorial/sequential/blackbox cell.

If the top-level input is connected to multiple ports/terminals in the fanout, and at least one fanout is inout port, then a violation is reported for the top-level input port.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is reported when a top-level input port is connected to an inout port:

[WARNING] Top input port <input-port-name> is connected to inout port <inout-port-name> (Instance: <instance-name>)

Potential Issues

The violation message is reported when a top-level input port is connected to an inout port.

Consequences of Not Fixing

This tag is used to follow better coding practice.

How to Debug and Fix

Double-click the violation message and open the schematic. The port is highlighted. Now trace the violating path in the incremental schematic window.

Example Code and/or Schematic

Consider the following example:

```
module sub (a,z,io);
    input a;
    output z;
    inout io;

    assign z = ~io;
endmodule

module top(q1,q5,k);
    output q1;
    output q5;
    input k;

    wire q3 = k;
    wire q4;

    sub inst_sub(.a(q1),.z(q4),.io(q3));
    assign q5 = q4;

endmodule
```

For the above example, the *DirectTopInputToInout-ML* tag reports the following violation message because top-level input port k is connected to inout port io:

Top input port 'k' is connected to inout port 'io' (Instance: 'inst_sub')

The top input port will be highlighted in the schematic.

Default Severity Label

Warning

tag Group

Not Applicable

Reports and Related Files

No related reports or files.

ShiftRegister-ML

Reports case coding style shift register

Description

If a case is written in the following style, then it is considered as shift register.

- The primary data (example: `data_in_sel`) locations will be shift left/right based on the increase/decrease in the case condition value.
- The primary data (example: `data_in_sel`) locations will be shift left/right based on the increase/decrease in the front index "0" bits number and back index "0" bits number.
- The increase/decrease in the front index "0" bits number and back index "0" bits number can be over one, but the increase/decrease value should be fixed per case condition.

The tag will not report violations if the count of case expression is less than 16.

Language

Verilog

Parameter(s)

- [*report_case_type*](#): The default value is `""`. Reports violations for `casez`, `casex`, `case_inside`, `unique`, `unique0` and `priority` parameter values. By default, the tag reports violations for case expressions.
- [*case_condition_count*](#): The default value is 16. Indicates the minimum number of case conditions that should be present for the tag to report violations.

Constraint(s)

None

Messages and Suggested Fix

Message

The following message appears at the location where shift register is detected in the case expression that is greater than the specific number:

[WARNING] shift register detected for signal <signal-name> in case. [Hierarchy: <hierarchy>]

Potential Issues

Violation may arise when the number of case expression is greater than a specific number.

Consequences of Not Fixing

In cases where the number of case expressions is greater than the specific number 16, the coding style will be synthesized to a large multiplexer or a large combinational logic.

Not fixing the violation will impact the timing closure and there might be some changes to be made in the design structure in the later stage. Overall, it impacts the design schedule.

How to Debug and Fix

When the number of case expression is greater than the specific number, you must split a big case statement to several small case statements to improve timing for high speed designs.

Example Code and/or Schematic

Verilog

Consider the following example:

MoreLint Tags

Example 1:

```

always @(*) begin
  case(put)
    11'd0 : data_shift = (1024 d0, data_in_sel, 1);
    11'd1 : data_shift = (1023 d0, data_in_sel, 1 d0);
    11'd2 : data_shift = (1022 d0, data_in_sel, 2 d0);
    11'd3 : data_shift = (1021 d0, data_in_sel, 3 d0);
    11'd4 : data_shift = (1020 d0, data_in_sel, 4 d0);
    11'd5 : data_shift = (1019 d0, data_in_sel, 5 d0);
    11'd6 : data_shift = (1018 d0, data_in_sel, 6 d0);
    11'd7 : data_shift = (1017 d0, data_in_sel, 7 d0);
    11'd8 : data_shift = (1016 d0, data_in_sel, 8 d0);
    11'd9 : data_shift = (1015 d0, data_in_sel, 9 d0);
    11'd10 : data_shift = (1014 d0, data_in_sel, 10 d0);
    11'd11 : data_shift = (1013 d0, data_in_sel, 11 d0);
    11'd12 : data_shift = (1012 d0, data_in_sel, 12 d0);
    11'd13 : data_shift = (1011 d0, data_in_sel, 13 d0);
    11'd14 : data_shift = (1010 d0, data_in_sel, 14 d0);
    11'd15 : data_shift = (1009 d0, data_in_sel, 15 d0);
    11'd16 : data_shift = (1008 d0, data_in_sel, 16 d0);
    11'd17 : data_shift = (1007 d0, data_in_sel, 17 d0);
    11'd18 : data_shift = (1006 d0, data_in_sel, 18 d0);
    11'd19 : data_shift = (1005 d0, data_in_sel, 19 d0);
    11'd20 : data_shift = (1004 d0, data_in_sel, 20 d0);
    11'd21 : data_shift = (1003 d0, data_in_sel, 21 d0);
  endcase
end

```

Front index decrease 1 bit or more

back index increase 1 bit or more

```

always @(*) begin
  case(put)
    11'd0 : data_shift = (1000 d0, data_in_sel, 24 d0);
    11'd1 : data_shift = (1002 d0, data_in_sel, 22 d0);
    11'd2 : data_shift = (1004 d0, data_in_sel, 20 d0);
    11'd3 : data_shift = (1006 d0, data_in_sel, 18 d0);
    11'd4 : data_shift = (1008 d0, data_in_sel, 16 d0);
    11'd5 : data_shift = (1010 d0, data_in_sel, 14 d0);
    11'd6 : data_shift = (1012 d0, data_in_sel, 12 d0);
    11'd7 : data_shift = (1014 d0, data_in_sel, 10 d0);
    11'd8 : data_shift = (1016 d0, data_in_sel, 8 d0);
    11'd9 : data_shift = (1018 d0, data_in_sel, 6 d0);
    11'd10 : data_shift = (1020 d0, data_in_sel, 4 d0);
    11'd11 : data_shift = (1022 d0, data_in_sel, 2 d0);
    11'd12 : data_shift = (1024 d0, data_in_sel, 0 d0);
  endcase
end

```

back index decrease 2 bit

front index increase 2 bit

```

always @(*) begin
  case (put)
    11'd0 : data_shift = {data_in_sel[51:0]}, 1'd0};
    11'd1 : data_shift = {data_in_sel[510:0]}, 2'd0};
    11'd2 : data_shift = {data_in_sel[509:0]}, 3'd0};
    11'd3 : data_shift = {data_in_sel[508:0]}, 4'd0};
    11'd4 : data_shift = {data_in_sel[507:0]}, 5'd0};
    11'd5 : data_shift = {data_in_sel[506:0]}, 6'd0};
    11'd6 : data_shift = {data_in_sel[505:0]}, 6'd0};
  endcase

```

back index increase 1 bit

primary data (data_in_sel) width decrease 1 bit

Example 2:

```

case (put_1)
  10'd0 : data_shift_1 = {1024'd0, data_in_sel_1, 1'd0};
  10'd1 : data_shift_1 = {1024'd0, data_in_sel_1, 2'd0};
  10'd2 : data_shift_1 = {1024'd0, data_in_sel_1, 3'd0};
  10'd3 : data_shift_1 = {1024'd0, data_in_sel_1, 4'd0};
  10'd4 : data_shift_1 = {1024'd0, data_in_sel_1, 5'd0};
  10'd5 : data_shift_1 = {1024'd0, data_in_sel_1, 6'd0};
  10'd6 : data_shift_1 = {1024'd0, data_in_sel_1, 7'd0};
  10'd7 : data_shift_1 = {1024'd0, data_in_sel_1, 8'd0};
  10'd8 : data_shift_1 = {1024'd0, data_in_sel_1, 9'd0};
  10'd9 : data_shift_1 = {1024'd0, data_in_sel_1, 10'd0};
  10'd10 : data_shift_1 = {1024'd0, data_in_sel_1, 11'd0};
  10'd11 : data_shift_1 = {1024'd0, data_in_sel_1, 12'd0};
  10'd12 : data_shift_1 = {1024'd0, data_in_sel_1, 13'd0};
  10'd13 : data_shift_1 = {1024'd0, data_in_sel_1, 14'd0};
  10'd14 : data_shift_1 = {1024'd0, data_in_sel_1, 15'd0};
  default : data_shift_1 = 'd0;
endcase

```

There will be no violation for the above case as the number of case expressions is less than 16.

Example 3:

```

case (put)
  11'd0 : data_shift = {1024'd0, data_in_sel, 0'd0};
  11'd1 : data_shift = {1023'd0, data_in_sel, 1'd0};
  11'd2 : data_shift = {1021'd0, data_in_sel, 2'd0};
  11'd3 : data_shift = {1018'd0, data_in_sel, 3'd0};
  11'd4 : data_shift = {1014'd0, data_in_sel, 4'd0};
  11'd5 : data_shift = {1009'd0, data_in_sel, 5'd0};
  11'd6 : data_shift = {1004'd0, data_in_sel, 6'd0};
  11'd7 : data_shift = {999'd0, data_in_sel, 7'd0};
  11'd8 : data_shift = {994'd0, data_in_sel, 8'd0};
  11'd9 : data_shift = {993'd0, data_in_sel, 9'd0};
  11'd10 : data_shift = {992'd0, data_in_sel, 10'd0};
  11'd11 : data_shift = {991'd0, data_in_sel, 11'd0};
  11'd12 : data_shift = {990'd0, data_in_sel, 12'd0};
  11'd13 : data_shift = {989'd0, data_in_sel, 13'd0};
  11'd14 : data_shift = {988'd0, data_in_sel, 14'd0};
  11'd15 : data_shift = {987'd0, data_in_sel, 15'd0};
  11'd16 : data_shift = {986'd0, data_in_sel, 16'd0};
endcase

```

There will be no violations for the above example as the change of the index is not constant among the case expression.

Example 4:

```

always @(*) begin
  case (put)
    11'd0 : data_shift = {1024'd0, data_in_sel, 0'd0};
    11'd1 : data_shift = {1023'd0, data_in_sel, 1'd0};
    11'd2 : data_shift = {1022'd0, data_in_sel, 2'd0};
    11'd3 : data_shift = {1021'd0, data_in_sel, 3'd0};
    11'd4 : data_shift = {1020'd0, data_in_sel, 4'd0};
    11'd5 : data_shift = {1019'd0, data_in_sel, 5'd0};
    11'd6 : data_shift = {1018'd0, data_in_sel, 6'd0};
    11'd7 : data_shift = {1017'd0, data_in_sel, 7'd0};
    11'd8 : data_shift = {1016'd0, data_in_sel, 8'd0};
    11'd9 : data_shift = {1015'd0, data_in_sel, 9'd0};
    11'd10 : data_shift = {1014'd0, data_in_sel, 10'd0};
    11'd11 : data_shift = {1013'd0, data_in_sel, 11'd0};
    11'd12 : data_shift = {1012'd0, data_in_sel, 12'd0};
    11'd13 : data_shift = {1011'd0, data_in_sel, 13'd0};
    11'd14 : data_shift = {1010'd0, data_in_sel, 14'd0};
    11'd15 : data_shift = {1009'd0, data_in_sel, 15'd0};
    11'd16 : data_shift = {1008'd0, data_in_sel, 16'd0};
    11'd17 : data_shift = {1007'd0, data_in_sel, 17'd0};
    11'd18 : data_shift = {1006'd0, data_in_sel, 18'd0};
    default : data_shift = '0;
  endcase
end

```

There is a violation as there are more than 16 case conditions and follows shift register definition.

There will be only one assignment support inside a case condition. If more than one assignment is found, then no violations are reported.

MoreLint Tags

Example 5:

```

always @(*) begin
  case (put)
    11'd0 : data_shift = {1024'd0, data_in_sel, 0'd0};
    11'd1 : data_shift = {1023'd0, data_in_sel, 1'd0};
    11'd2 : data_shift = {1022'd0, data_in_sel, 2'd0};
    11'd2 : data_shift2 = {1022'd0, data_in_sel, 2'd0};
    11'd3 : data_shift = {1021'd0, data_in_sel, 3'd0};
    11'd4 : data_shift = {1020'd0, data_in_sel, 4'd0};
    11'd5 : data_shift = {1019'd0, data_in_sel, 5'd0};
    11'd6 : data_shift = {1018'd0, data_in_sel, 6'd0};
    11'd7 : data_shift = {1017'd0, data_in_sel, 7'd0};
    11'd8 : data_shift = {1016'd0, data_in_sel, 8'd0};
    11'd9 : data_shift = {1015'd0, data_in_sel, 9'd0};
    11'd10 : data_shift = {1014'd0, data_in_sel, 10'd0};
    11'd11 : data_shift = {1013'd0, data_in_sel, 11'd0};
    11'd12 : data_shift = {1012'd0, data_in_sel, 12'd0};
    11'd13 : data_shift = {1011'd0, data_in_sel, 13'd0};
    11'd14 : data_shift = {1010'd0, data_in_sel, 14'd0};
    11'd15 : data_shift = {1009'd0, data_in_sel, 15'd0};
    11'd16 : data_shift = {1008'd0, data_in_sel, 16'd0};
    11'd17 : data_shift = {1007'd0, data_in_sel, 17'd0};
    11'd18 : data_shift = {1006'd0, data_in_sel, 18'd0};
    default : data_shift = '0;
  endcase
end

```

No violations as one case condition has more than one assignment.
LHS of the assignment statement does not need to be same.

Example 6:

```

always @(*) begin
  case (put)
    11'd0 : data_shift1 = {1024'd0, data_in_sel, 0'd0};
    11'd1 : data_shift2 = {1023'd0, data_in_sel, 1'd0};
    11'd2 : data_shift3 = {1022'd0, data_in_sel, 2'd0};
    11'd3 : data_shift4 = {1021'd0, data_in_sel, 3'd0};
    11'd4 : data_shift5 = {1020'd0, data_in_sel, 4'd0};
    11'd5 : data_shift6 = {1019'd0, data_in_sel, 5'd0};
    11'd6 : data_shift7 = {1018'd0, data_in_sel, 6'd0};
    11'd7 : data_shift8 = {1017'd0, data_in_sel, 7'd0};
    11'd8 : data_shift9 = {1016'd0, data_in_sel, 8'd0};
    11'd9 : data_shift = {1015'd0, data_in_sel, 9'd0};
    11'd10 : data_shift = {1014'd0, data_in_sel, 10'd0};
    11'd11 : data_shift = {1013'd0, data_in_sel, 11'd0};
    11'd12 : data_shift = {1012'd0, data_in_sel, 12'd0};
    11'd13 : data_shift = {1011'd0, data_in_sel, 13'd0};
    11'd14 : data_shift = {1010'd0, data_in_sel, 14'd0};
    11'd15 : data_shift = {1009'd0, data_in_sel, 15'd0};
    11'd16 : data_shift = {1008'd0, data_in_sel, 16'd0};
    11'd17 : data_shift = {1007'd0, data_in_sel, 17'd0};
    11'd18 : data_shift = {1006'd0, data_in_sel, 18'd0};
    default : data_shift = '0;
  endcase
end

```

Violation is reported as LHS of the assignment does not matter.

Parameter support for specifying how many case conditions should be checked before reporting violations through `case_condition_count`, parameter. The default value is 16.

Example 7:

```

always @(*) begin
  case (put)
    11'd0 : data_shift = {1024'd0, data_in_sel, 0'd0};
    11'd1 : data_shift = {1023'd0, data_in_sel, 1'd0};
    11'd2 : data_shift = {1022'd0, data_in_sel, 2'd0};
    11'd3 : data_shift = {1021'd0, data_in_sel, 3'd0};
    11'd4 : data_shift = {1020'd0, data_in_sel, 4'd0};
    11'd5 : data_shift = {1019'd0, data_in_sel, 5'd0};
    11'd6 : data_shift = {1018'd0, data_in_sel, 6'd0};
    11'd7 : data_shift = {1017'd0, data_in_sel, 7'd0};
    11'd8 : data_shift = {1016'd0, data_in_sel, 8'd0};
    11'd9 : data_shift = {1015'd0, data_in_sel, 9'd0};
    11'd10 : data_shift = {1014'd0, data_in_sel, 10'd0};
    11'd11 : data_shift = {1013'd0, data_in_sel, 11'd0};
    11'd12 : data_shift = {1012'd0, data_in_sel, 12'd0};
    11'd13 : data_shift = {1011'd0, data_in_sel, 13'd0};
    11'd14 : data_shift = {1010'd0, data_in_sel, 14'd0};
    11'd15 : data_shift = {1009'd0, data_in_sel, 15'd0};
    11'd16 : data_shift = {1008'd0, data_in_sel, 16'd0};
    11'd17 : data_shift = {1007'd0, data_in_sel, 17'd0};
    11'd18 : data_shift = {1006'd0, data_in_sel, 18'd0};
    default : data_shift = '0;
  endcase
end

```

No violations with `case_condition_count` set to 32, but violation is reported if it is less than 19.

By default, only case expressions are violated. Violations can be reported for `casex`, `casez` and `case` inside using `check_case_type` parameter.

Example 8

```

always @(*) begin
  case (put)
    11'd0 : data_shift1 = {1024'd0, data_in_sel, 0'd0};
    11'd1 : data_shift2 = {1023'd0, data_in_sel, 1'd0};
    11'd2 : data_shift3 = {1022'd0, data_in_sel, 2'd0};
    11'd3 : data_shift4 = {1021'd0, data_in_sel, 3'd0};
    11'd4 : data_shift5 = {1020'd0, data_in_sel, 4'd0};
    11'd5 : data_shift6 = {1019'd0, data_in_sel, 5'd0};
    11'd6 : data_shift7 = {1018'd0, data_in_sel, 6'd0};
    11'd7 : data_shift8 = {1017'd0, data_in_sel, 7'd0};
    11'd8 : data_shift9 = {1016'd0, data_in_sel, 8'd0};
    11'd9 : data_shift = {1015'd0, data_in_sel, 9'd0};
    11'd10 : data_shift = {1014'd0, data_in_sel, 10'd0};
    11'd11 : data_shift = {1013'd0, data_in_sel, 11'd0};
    11'd12 : data_shift = {1012'd0, data_in_sel, 12'd0};
    11'd13 : data_shift = {1011'd0, data_in_sel, 13'd0};
    11'd14 : data_shift = {1010'd0, data_in_sel, 14'd0};
    11'd15 : data_shift = {1009'd0, data_in_sel, 15'd0};
    11'd16 : data_shift = {1008'd0, data_in_sel, 16'd0};
    11'd17 : data_shift = {1007'd0, data_in_sel, 17'd0};
    11'd18 : data_shift = {1006'd0, data_in_sel, 18'd0};
    default : data_shift = '0;
  endcase
end

```

MoreLint Tags

```

always @(*) begin
  casez (put)
    11'd0 : data_shift1 = {1024'd0, data_in_sel, 0'd0};
    11'd1 : data_shift2 = {1023'd0, data_in_sel, 1'd0};
    11'd2 : data_shift3 = {1022'd0, data_in_sel, 2'd0};
    11'd3 : data_shift4 = {1021'd0, data_in_sel, 3'd0};
    11'd4 : data_shift5 = {1020'd0, data_in_sel, 4'd0};
    11'd5 : data_shift6 = {1019'd0, data_in_sel, 5'd0};
    11'd6 : data_shift7 = {1018'd0, data_in_sel, 6'd0};
    11'd7 : data_shift8 = {1017'd0, data_in_sel, 7'd0};
    11'd8 : data_shift9 = {1016'd0, data_in_sel, 8'd0};
    11'd9 : data_shift = {1015'd0, data_in_sel, 9'd0};
    11'd10 : data_shift = {1014'd0, data_in_sel, 10'd0};
    11'd11 : data_shift = {1013'd0, data_in_sel, 11'd0};
    11'd12 : data_shift = {1012'd0, data_in_sel, 12'd0};
    11'd13 : data_shift = {1011'd0, data_in_sel, 13'd0};
    11'd14 : data_shift = {1010'd0, data_in_sel, 14'd0};
    11'd15 : data_shift = {1009'd0, data_in_sel, 15'd0};
    11'd16 : data_shift = {1008'd0, data_in_sel, 16'd0};
    11'd17 : data_shift = {1007'd0, data_in_sel, 17'd0};
    11'd18 : data_shift = {1006'd0, data_in_sel, 18'd0};
    default : data_shift = '0;
  endcase
end

```

```

always @(*) begin
  case (put) inside
    11'd0 : data_shift1 = {1024'd0, data_in_sel, 1'd0};
    11'd1 : data_shift2 = {1023'd0, data_in_sel, 1'd0};
    11'd2 : data_shift3 = {1022'd0, data_in_sel, 2'd0};
    11'd3 : data_shift4 = {1021'd0, data_in_sel, 3'd0};
    11'd4 : data_shift5 = {1020'd0, data_in_sel, 4'd0};
    11'd5 : data_shift6 = {1019'd0, data_in_sel, 5'd0};
    11'd6 : data_shift7 = {1018'd0, data_in_sel, 6'd0};
    11'd7 : data_shift8 = {1017'd0, data_in_sel, 7'd0};
    11'd8 : data_shift9 = {1016'd0, data_in_sel, 8'd0};
    11'd9 : data_shift = {1015'd0, data_in_sel, 9'd0};
    11'd10 : data_shift = {1014'd0, data_in_sel, 10'd0};
    11'd11 : data_shift = {1013'd0, data_in_sel, 11'd0};
    11'd12 : data_shift = {1012'd0, data_in_sel, 12'd0};
    11'd13 : data_shift = {1011'd0, data_in_sel, 13'd0};
    11'd14 : data_shift = {1010'd0, data_in_sel, 14'd0};
    11'd15 : data_shift = {1009'd0, data_in_sel, 15'd0};
    11'd16 : data_shift = {1008'd0, data_in_sel, 16'd0};
    11'd17 : data_shift = {1007'd0, data_in_sel, 17'd0};
    11'd18 : data_shift = {1006'd0, data_in_sel, 18'd0};
    default : data_shift = '0;
  endcase
end

```

By default, there will be no violation for the above example. However, if you set `casez`, `casex` and `case_inside` as parameter value for `report_case_type` then there will be violations.

Default Severity Label

Warning

tag Group

No Applicable

Reports and Related Files

No related reports or files.

UseParamWidthInOverriding-ML

Parameters should not be overridden without specifying the width in instantiations

Description

The *UseParamWidthInOverriding-ML* tag must explicitly specify the width of overriding parameter if the default parameter width is greater than 32 bits in instantiations. Else, the tag will report violations.

Default Weight

32

Language

Verilog

Parameter(s)

- *check_below_32*: When set to *yes*, reports violations (in addition to reporting violations to the default behaviour) if the:
 - ☐ Default width of the sized parameters is less than or equal to 32 bits.
 - ☐ Parameter is overridden by an unsized based number, where the width of the unsized based number is greater than the default width.
- *unsized_param_cutoff*: Reports violations if the parameter is overridden by an unsized based number, where the width of the unsized based number is greater than and equal to the value given in the parameter.

NOTE: *This parameter is effective only if the *check_below_32* parameter is set to *yes*.*

Constraint(s)

None

Messages and Suggested Fix

The following message appears when the width of the overriding parameter is not specified:

[warning] Parameter '<parameter_name>' should not be overridden without specifying the width in expr '<expr>' of instance '<instance_name>'. [Hierarchy: <hierarchy>]

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

Double click the violation message, SpyGlass points to the RTL line where the parameter is passed. Specify the width of the overriding parameter.

Example Code and/or Schematic**Example 1**

In the following example, the tag reports violation as the overriding parameter width is not specified.

```
TEST # (  
    . WIDTH('d35),  
    . LENGTH ('d50) // violation , width is not  
specified  
    )  
    U_TEST (  
    );  
module TEST ();  
parameter WIDTH = 32'd20;  
parameter LENGTH = 33'd10; // default width is 33 (>32)  
endmodule
```

Example 2

In the following example, the tag does not report violation as the overriding parameter width is not specified.

```
TEST # (  
    . WIDTH('d35),  
    . LENGTH (33'd50) // no-violation, width is specified as 33  
    )
```

```

        U_TEST (
        );

        module TEST ();

        parameter WIDTH = 32'd20;
        parameter LENGTH = 33'd10; // default width is 33 (>32)
        endmodule

```

Example 3

In the following example, the tag reports violation as the overriding parameter width is not specified.

```

module sample();
    parameter integer width = 20; // default width is 32
    parameter longint length = 10; // default width is 64
    (>32) , longint type
    reg a;
endmodule

module test();

    sample #(
'd35,
'b11      // violation , width is not specified
    ) inst();
endmodule

```

Example 4

Consider the following example:

```

typedef longint my_type; // time, real, realtime

```

```

        module sample();
        parameter my_type length = 10;  // default width is 64
(>32)
        reg a;
        assign a = 2;
        endmodule

```

```

        module test();
        sample #('b11) inst(); // violation , width is not
specified
        endmodule

```

Example 5

Consider the following example:

```

////////-> parameters in .prj file <-////////
configure_lint_tag_parameter -tag
UseParamWidthInOverriding-ML -parameter check_below_32
-value yes
configure_lint_tag_parameter -tag UseParamWidthInOverriding-
ML -parameter unsized_param_cutoff -value 17
////////////////////////////////////////

```

```

        module sample();
        parameter shortint length = 10;  // default width is 16
(<32)
        reg a;
        assign a = 2;
        endmodule

        module test();
        sample #('h1F1F1) inst();  // violation , width is not

```


MoreLint Tags

```
specified & actual width of 'h1F1F1is 17 (>=17)
endmodule
```

Default Severity Label

Warning

Tag Group

Not applicable

Reports and Related Files

No related reports or files.

IncompatibleSVAssign-ML

SV assign operators should not be used in expressions

When to Use

Use this tag to detect SV assign operators used in expressions.

Description

The *IncompatibleSVAssign-ML* tag reports a violation if the SV assignment operator has not been used as a single statement with the corresponding variable.

The tag reports a violation for the following SV assign types:

++, --, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, <<<=, >>>=

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Incompatible sv assignment '<expression>' detected

Potential Issues

A violation message is reported when an incompatible SV assignment is detected in the expression.

Consequences of Not Fixing

Not fixing the violation may result in VER-700 errors in the DC for the design.

How to Debug and Fix

To debug the violation, click the violation message, SpyGlass points the RTL line where the SV assignment operator is used.

Examples and/or Schematic

Example 1

```
module test (input in, output reg out);  
  reg tmp;  
  always@ (*)  
    out <= tmp++; // violation  
endmodule
```

Example 2

```
module test (input in, output reg [2:0] out);  
  always_comb  
  begin  
    for (int i = 0; i<=2; i++) // no-violation  
      out[i] = in;  
  end  
endmodule
```

Example 3

```
module test (input in, output reg [2:0] out);  
  always_comb  
  begin  
    for (int i = 0; i<=2; i=++i) // violation  
      out[i] = in;  
  end  
endmodule
```

Example 4

```
module test (input in, clk, output reg out);
```

```
int i;
always@ (posedge clk)
begin
  if (i++) // violation
  out = 0;
  else
  out = in;
end
endmodule
```

Example 5

```
module test (input in, output out);
  int i;
  int j;
  assign j = i++; // violation
  assign out = j;
endmodule
```

Example 6

```
module test (sel);
  int i;
  int a;
  int b;
  input sel;
  always_ff @ (posedge sel)
  begin
    i--; // no-violation
    a <= i--; // violation
    b <= a-- + i; // violation
  end
endmodule
```

MoreLint Tags

```
end  
endmodule
```

Example 7

```
module test (sel);  
  int i;  
  int a;  
  int b;  
  input sel;  
  always_ff @ (posedge sel)  
  begin  
    a <= --i; // violation  
    b <= ++a; // violation  
  end  
endmodule
```

Example 8

```
module test (input in, output reg out);  
  parameter PARAM = 1;  
  int i;  
  int j;  
  always @ (posedge in)  
  if (PARAM == 0) // static condition  
    out <= i++; // violation  
  else  
    out <= j++; // violation  
endmodule
```

Example 9

```
module test(sel,out);
```

```
int i;
int a;
int b;
input sel;
output reg out;
always_ff @ (posedge sel)
begin
    out <= i++? a: b; // violation
end
endmodule
```

Example 10

```
module test(out);
function my_func (input in);
    my_func = ~in;
endfunction
int i;
int a;
int b;
out reg out;
always @ (*)
    out = my_func(i++); // violation
endmodule
```

Example 10

```
module test(out);
function my_func (input in);
    my_func = ~in;
endfunction
```

MoreLint Tags

```
int i;
int a;
int b;
out reg out;
always @ (*)
out = my_func(i++); // violation
endmodule
```

Example 10

```
module test(out);
function my_func (input in);
my_func = ~in;
endfunction
int i;
int a;
int b;
out reg out;
always @ (*)
out = my_func(i++); // violation
endmodule
```

Example 11

```
module test (input in, output reg out);
reg tmp;
always @ (*)
out <= {tmp++}; // violation
endmodule
```

Example 12

```
module test (input in, output out);
```

```
function my_func;  
  int i;  
  return i++; // violation  
endfunction  
assign out = in;  
endmodule
```

Example 13

```
module test (input in, output reg out);  
  int i, j;  
  always@(*)  
    out = i >> j++; // violation  
endmodule
```

Example 14

```
module test (input in, output reg out);  
  int i, j;  
  always@(*)  
    out = i >> j++; // violation  
endmodule  
  
module test(sel);  
  int i;  
  int a;  
  input sel;  
  always_ff @(posedge sel)  
  begin  
    i++; // no-violation  
    a <= (i++); // violation  
    ++i; // no-violation  
    a <= (++i); // violation  
  end  
endmodule
```


MoreLint Tags

```
i--; // no-violation
a <= (i--); // violation
--i; // no-violation
a <= (--i); // violation
i+=2; // no-violation
a <= (i+=2); // violation
i-=2; // no-violation
a <= (i-=2); // violation
i*=2; // no-violation
a <= (i*=2); // violation
i/=2; // no-violation
a <= (i/=2); // violation
i%=2; // no-violation
a <= (i%=2); // violation
i&=2; // no-violation
a <= (i&=2); // violation
i|=2; // no-violation
a <= (i|=2); // violation
i^=2; // no-violation
a <= (i^=2); // violation
i<<=2; // no-violation
a <= (i<<=2); // violation
i>>=2; // no-violation
a <= (i>>=2); // violation
i<<<=2; // no-violation
a <= (i<<<=2); // violation
i>>>=2; // no-violation
a <= (i>>>=2); // violation
```

```
a <= (i ==? 2); // no-violation  
a <= (i !=? 2); // no-violation  
end  
endmodule
```

Default Severity Label

Warning

Reports and Related Files

None

IncrementalForLoop-ML

Tag will report violations for usage of ++ and - operators in the for loop.

When to Use

Use this tag to detect ++ and - operators in a for loop.

Description

The IncrementalForLoop-ML tag reports violation when ++ and - operators are used in a for loop.

However, the tag will not run when enableSV is turned off as there are syntax errors for usage of ++ and --

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] <++|--> used in the for loop with variable <variable-name> [hierarchy <hierarchy>]

Potential Issues

The violation message is reported when ++ or -- operators are detected for a for loop.

Consequence of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

Click the violation message to highlight the ++/-- statement of the for loop in the HDL viewer.

Example Code and/or Schematic

Example 1

```
module test;
reg [3:0] out;
    reg [3:0] in;
always_comb begin
integer j;

for (j = 0; j < 4; j++) //violation as operation ++ used
in for loop

out[j] = in[j];
end
endmodule
```

Example 2

```
module test;
reg [3:0] out;
    reg [3:0] in;
always_comb begin

integer j;
for (j = 0; j < 4; j=j+1) // no Violation
out[j] = in[j];
end
endmodule
```

Example 3

```
module test;
reg [3:0] out;
    reg [3:0] in;
genvar j;
for (j = 0; j < 4; j++) //violation as operation ++ used
in for loop
assign out[j] = in[j];
```

MoreLint Tags

```
endgenerate
endmodule
```

Example 4

```
module test;
reg [3:0] out;
  reg [3:0] in;
genvar j;
for (j = 0; j < 4; j =j + 1) //no violation
assign out[j] = in[j];
endgenerate
endmodule
```

Example 5

```
module test;
reg [3:0] out;
  reg [3:0] in;
always_comb begin
integer j;
for (j = 3; j < -1; j--) //violation as operation ++ used
in for loop

out[j] = in[j];
end
endmodule
```

Example 6

```
module test;
reg [3:0] out;
  reg [3:0] in;
always_comb begin
integer j;
for (j = 3; j < -1; j=j-1) // no Violation
out[j] = in[j];
```

```
end  
endmodule
```

Example 7

```
module test;  
reg [3:0] out;  
  reg [3:0] in;  
genvar j;  
for (j = 3; j < -1; j--) //violation as operation ++ used  
in for loop  
assign out[j] = in[j];  
endgenerate  
endmodule
```

Example 8

```
module test;  
reg [3:0] out;  
  reg [3:0] in;  
genvar j;  
for (j = 3; j < -1; j = j - 1) //no violation  
assign out[j] = in[j];  
endgenerate  
endmodule
```

Default Severity Label

Warning

Tag Group

Not Applicable

Reports and Related Files

No related reports or files

DisallowUnknownInCase-ML

Tag will check the usage of 'x', 'z' or '?' in all case statements.

Description

The DisallowUnknownInCase-ML tag checks the usage of 'x', 'z' or '?' in all case statements (case, casez, casex and case inside statements).

Parameter(s)

allow_types: The default value is none. Set the value of the parameter to one or comma separated list of 'x', 'z' and '?' to ignore reporting violations for usage of given types.

Constraint(s)

None

Message Details

The *DisallowUnknownInCase-ML* tag reports the following violation message:

unknown placeholders are used in case statement [Hierarchy: '<hier-path>']

Tag Severity

Error

Examples (Verilog)

Consider the following example that infers a MUX:

Example 1: No Violation

```
logic [3:0] y;
logic [1:0] a;
logic      en;

always_comb begin
    y = '0;
```

```
case ({en,a})
  3'b100: y[a] = 1'b1;
  3'b101: y[a] = 1'b1;
  3'b110: y[a] = 1'b1;
  3'b111: y[a] = 1'b1;
endcase
end
```

Case 2: Violation

```
logic [3:0] y;
logic [1:0] a;
logic      en;

always_comb begin
  y = '0;
  priority case ({en,a})
    3'b100: y[a] = 1'b1;
    3'b10x: y[a] = 1'b1;
    3'b110: y[a] = 1'b1;
    3'b111: y[a] = 1'b1;
  endcase
end
```

Case 3: No violations if '?' is provided to parameter allow_types.

```
always @(*)
begin
  priority case (tmpSig) inside
    3'b00? : q<=1;
```


MoreLint Tags

```
        3'b0?0 : q<=2;  
        3'b?00 : q<=3;  
    endcase
```

```
end
```

Case 4: No violations if 'z' is provided to parameter allow_types.

```
always_comb begin  
    y = '0;  
    case ({en,a})  
        3'b10z: y[a] = 1'b1;  
        3'b110: y[a] = 1'b1;  
        3'b111: y[a] = 1'b1;  
    endcase  
end
```

DivideByConstant-ML

Unverifiable formality issue due to division of variable signal by constant.

When to Use

Use this tag to identify variable signals divided by a constant.

Description

This tag reports violations, if variable signal is divided by a constant.

Parameter(s)

[numerator_size](#): Default value is 0. Set the value of the parameter to an integer value to enable the *DivideByConstant-ML* tag to report violations only for dividers whose numerator size (width, according to the LRM) is greater than or equal to the cutoff size specified using this parameter.

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Signal '<variable signal name>' name is divided by constant '<constant expression>' [Hierarchy: '<hierarchy>']

Potential Issues

The violation message is reported when a signal name is divided by the constant.

Consequences of Not Fixing

Not fixing the violation may result in unexpected design behavior.

How to Debug and Fix

Click the violation message to display in the HDL viewer, the instance where division by constant is present.

Example Code and/or Schematic

Example 1

```
module DVD2( in , out );
input  wire [31:0] in;
output reg  [31:0] out;

always @(in) begin
    out = in / 2; //violation
end

endmodule
```

Example 2

```
module DVD4( in , out );

input  wire [31:0] in;
output reg  [31:0] out;
parameter    VAL = 4;

always @(in) begin
    out = in / VAL; //violation
end

endmodule
```

Example 3

```
module DVDVAR(in,sel,out);

input      [31:0] in;
input      [2:0]  sel;
output reg  [31:0] out;

always @(in,sel) begin
    out = in / sel; //no violation
end
```

```
endmodule
```

Example 4

```
module DVD5( in , out );

input  wire [31:0] in;
output reg  [31:0] out;
parameter    VAL = 5;

always @(in) begin
    if( in / VAL == 0 ) //violation
        out = in ;
    else
        out = ~in;
end

endmodule
```

Example 5

```
module test( in , out );

input  wire [31:0] in;
output reg  [31:0] out;

always @(in) begin
    out = in % 2; // violation , tag will consider the %
    operator as well
end

endmodule
```

Default Severity Label

Warning

MoreLint Tags

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

NoOutputPort-ML

Module with no output

Description

The NoOutputPort-ML tag reports a violation for those modules that do not have any output ports.

Parameters

- *consider_inout*: Default value is `no`. If you set this parameter `yes`, the NoOutputPort-ML tag considers inout port as output port and ignores violation for the modules that have inout ports.
- *ignore_module*: By default, this parameter is not set. If you set this parameter to a comma separated list of PERL regular expressions, the NoOutputPort-ML tag ignores violations for the modules that match any of these specified regular expressions.

Message Details

The following message appears if a module does not have an output port:

[WARNING] Module '<module-name>' has no output port

Example and /or Schematic

Example 1

Consider the following example where the tag reports a violation because there is no output port for the module `mod`:

```
module mod(input in, in1); //Violation
...
endmodule
```

Example 2

Consider the following example where the tag does not report a violation because an output port is present for the module `test`:

```
module test(input in, output out); // No violation
endmodule
```

MoreLint Tags

Severity

Warning

FuncWithIntReturn-ML

Function with integer return value

Description

The FuncWithIntReturn-ML Tag reports a violation if a function returns integers because such function are not synthesizable.

Parameter(s)

- **consider_inout**: Default value is no. If you set this parameter yes, the FuncWithIntReturn-ML tag considers inout port as output port and ignores violation for the modules that have inout ports.
- **ignore_module**: By default, this parameter is not set. If you set this parameter to a comma-separated list of PERL regular expressions, the FuncWithIntReturn-ML tag ignores violations for the modules that match any of these specified regular expressions.

Message Details

The following message appears when a function returns integers:

Do not use function '<function-name>' with integer return value

Example and /or Schematic

Example 1

Consider the following example where the tag reports a violation because the function `adder` returns an integer:

```
function integer adder; // Violation
input a, b, c;
adder = a + b + c;
endfunction
```

Example 2

Consider the following example where the tag does not report violation:

```
function myfunc; // No violation
input a, b, c;
myfunc = a + b - c;
endfunction
```


MoreLint Tags

Severity

Warning

InoutPortConnectionSanity-ML

Checks if inout port is connected to right type of IO cell pin

Description

This tag ensures that inout ports are only connected to a valid IO PAD cell pin of type "Bi-Dir."

This tag reports primary inout ports that are topologically connected to either multiple PAD cell pins or no PAD cell pin. Primary inout port should be connected to only one inout pad cell pin.

This tag does a topological traversal to check PAD pin connectivity. During fan-in/fan-out traversal, only simple buffers/inverters (SpyGlass generated) are skipped.

Some of the invalid inout port connections are described below:

- Inout port connected to a cell input only
- Inout port connected to output pin of a cell only
- Inout port does not connect to any cell instance
- Inout port connected to IO PAD cell of incorrect type input and/or output

NOTE: *Cases are similar to PortIOConnectionSanity-ML tag, but the difference is with respect to the port direction.*

Parameters

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when inout ports are connected to either multiple PAD cell pins or no PAD cell pins:

[ERROR] Inout port '<port-name>' should be connected to single inout pad cell pin. But connections are: '<hier-pin-names>'

Where, <hier-pin-names> are the hierarchical pin names.

MoreLint Tags

Default Severity Label

Error

tag Group

None

Reports and Related Files

None

InstanceNameRequired-ML

Instance without name

Description

The InstanceNameRequired-ML tag reports violation for all such instantiation which do not have any name. This will Include instantiation from a module and interface.

Parameter(s)

None

Constraint(s)

None

Message Details

```
$msg = "<module/interface> '<module name/ interface name>'  
instantiated without name";
```

Tag Severity

Warning

HangingFlopOutput-ML

Report unused flip-flops in the design

Language

Verilog, VHDL

Description

The HangingFlopOutput-ML tag flags flip-flop output which is left unused in the design.

The HangingFlopOutput-ML tag reports a violation if the output of a flip-flop in a module is left hanging inside that module.

The HangingFlopOutput-ML tag also reports a violation if the output of a flip-flop connected to the output of a module is left hanging outside that module. A violation will also be reported if the output of a module is connected to an unloaded instance input of another module.

NOTE: *This tag ignores all the buffers and inverters while traversing.*

Message Details

The following message appears at the location where a net *<net-name>* connected to the output of a flip-flop in a module is left hanging inside that module:

Flop output '*<net-name>*' is unused

The following message appears at the location where a flip-flop output *<net-name>* is connected to the output of a module *<mod-name>* and is not used outside that module:

Flop output '*<net-name>*' connected to module output '*<mod-name>*' is unused outside module

Severity

Warning

LoopVarFreeAssign-ML

Loop variable independent assignments need optimization.

Description

The LoopVarFreeAssign-ML tag reports violations for assignments within generate for loops, if LHS of that assignment is independent from loop variables. If LHS of assignment is not dependent on loop variable in generate loop, that assignment needs optimization to avoid unintended functional behaviors in synthesis and simulation.

This tag checks for all types of assignments (continuous assignments, blocking assignments, non-blocking assignments). However, the tag does not consider loop variable usages in RHS of the assignment.

Parameter(s)

None

Constraint(s)

None

Message Details

Assignment to signal '<signal name>' is not depend on loop variable(s) '<loop variable name(s)>'

Example Code and/or Schematic

Case 1

```
module test( in1 , in2 , in3 , out1 , out2 , out3 );

parameter NUM = 3;
input in1;
input [NUM-1:0][NUM-1:0] in2;
input [NUM-1:0][NUM-1:0][NUM-1:0] in3;

output [NUM-1:0][NUM-1:0][NUM-1:0] out1;
output [NUM-1:0][NUM-1:0] out2;
```

MoreLint Tags

```

output [NUM-1:0][NUM-1:0]out3;

genvar i, j , k;
generate
for(i=0; i<NUM; i=i+1) begin:L1
for(j=0; j<NUM; j=j+1) begin:L2
for(k=0; k<NUM; k=k+1) begin:L3

assign out1[i][j][k] = in1; //no violation - all loop
variables used
assign out2[i][k]      = in2[i][k]; // violation
assign out3[i][k]      = in3[i][j][k]; // violation
end
end
end
endgenerate

endmodule

```

Example 2

```

module test( in1 , in2 , out1 , out2 );

parameter NUM = 3;
input in1;
input [NUM-1:0][NUM-1:0]in2;

output [NUM-1:0][NUM-1:0][NUM-1:0]out1;
output [NUM-1:0][NUM-1:0]out2;

genvar i, j , k;
generate
for(i=0; i<NUM; i=i+1) begin
for(j=0; j<NUM; j=j+1) begin
for(k=0; k<NUM; k=k+1) begin

    assign out1[i][j][k] = in1; //no violation

```

```

end
    end

    for(k=0; k<NUM; k=k+1) begin
        assign out2[i][k]    = in2[i][k]; //no violation
    end

    end
endgenerate
endmodule

```

Example 3

```

module test( in1 , in2 , in3 , out1 , out2 , out3 );

parameter NUM = 3;
input in1;
input [NUM-1:0][NUM-1:0]in2;
input [NUM-1:0][NUM-1:0][NUM-1:0]in3;

output reg [NUM-1:0][NUM-1:0][NUM-1:0]out1;
output reg [NUM-1:0][NUM-1:0]out2;
output reg [NUM-1:0][NUM-1:0]out3;

genvar i, j , k;
generate
for(i=0; i<NUM; i=i+1) begin:L1
for(j=0; j<NUM; j=j+1) begin:L2
for(k=0; k<NUM; k=k+1) begin:L3
always @(*) begin

out1[i][j][k] = in1; //no violation
out2[i][k]    = in2[i][k]; // violation
    if( in3[i][j][k] )
        out3[i][k]    <= in1; // violation
    end
end
end
end

```


MoreLint Tags

end
endgenerate

endmodule

Tag Severity

Warning

LoopVarInAlways-ML

For-Loop variable defined by using integer must be declared outside of always block

When to Use

Use this tag to detect for-loop variable that is defined using integer and declared inside always block.

Description

The LoopVarInAlways-ML tag reports violation when a for-loop variable is defined by using integer but is declared inside the always block.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] For-Loop variable '<var name>' is defined inside of an always block. [Hierarchy: 'hierarchy_to_the_var']

Potential Issues

The violation message is reported when a for-loop variable is defined inside on an always block.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

Click the violation message to highlight the instance where the integer variable is declared.

Example Code and/or Schematic

Example: Non-violating Scenario

```
integer i;
```

MoreLint Tags

```

always @ (posedge i_clk)
begin
    if (i_rst_n == 1'b0)
    begin
        o_data <= 10'b0;
    end
    else
    begin:OUTPUT_ASSIGN
        for (i=0;i<10;i=i+1)
        begin
            o_data[i] <= i_data[i];
        end
    end
end
end

```

Example :Violating Scenarios***Example 1***

```

always @ (posedge i_clk)
begin
    if (i_rst_n == 1'b0)
    begin
        o_data <= 10'b0;
    end
    else
    begin:OUTPUT_ASSIGN
        integer i; //violation
        for (i=0;i<10;i=i+1)
        begin
            o_data[i] <= i_data[i];
        end
    end
end
end

```

Example 2

```

module test(input clk, input rst);
    reg data [10:0], dt[10:0];
    always @ (*)

```

```
begin
    integer i; // violation
    for (i=0;i<10;i=i+1)
    begin
        data[i] <= dt[i];
    end
end
endmodule
```

Example 3

```
module test(input clk, input rst);
    reg data [10:0], dt[10:0];
    always_ff @ (posedge clk)
    begin
        integer i; // violation
        for (i=0;i<10;i=i+1)
        begin
            data[i] <= dt[i];
        end
    end
endmodule
```

Example 4

```
module test(input clk, input rst);
    reg data [10:0], dt[10:0];

    always_latch
    begin
        integer i; // violation
        for (i=0;i<10;i=i+1)
        begin
            data[i] <= dt[i];
        end
    end
endmodule
```

Example 5

```

module test(input clk, input rst);
  reg data [10:0], dt[10:0];
  always_comb
  begin
    integer i; // violation
    for (i=0;i<10;i=i+1)
    begin
      data[i] <= dt[i];
    end
  end
endmodule

```

Example 6

```

module test(input clk, input rst);
  reg data [10:0], dt[10:0];
  always_comb
  begin
    int i; // violation
    for (i=0;i<10;i=i+1)
    begin
      data[i] <= dt[i];
    end
  end
endmodule

```

Example 7

```

module test(input clk);
  always @ (posedge clk)
  begin
    shortint i; // violation
    for (i=0;i<10;i=i+1)
    begin;

```

```
end  
endmodule
```

Example 8

```
module test(input clk);  
always @ (posedge clk)  
begin  
    longint i;    // violation  
    for (i=0;i<10;i=i+1)  
    begin;  
        end  
    end  
endmodule
```

Example 9

```
module test(input clk);  
always @ (posedge clk)  
begin  
    int i;    // violation  
    for (i=0;i<10;i=i+1)  
    begin;  
        end  
    end  
endmodule
```

Example 10

```
module test(input clk);  
always @ (posedge clk)  
begin  
    integer i;    // violation  
    for (i=0;i<10;i=i+1)  
    begin;  
        end  
    end  
end
```

MoreLint Tags

```
endmodule
```

Example 11

```
typedef shortint sample_type; //or longint or integer or int
module test(input clk);
    always @ (posedge clk)
    begin
        sample_type i;    // violation
        for (i=0;i<10;i=i+1)
            begin ;
        end
    end
endmodule
```

Tag Severity

Warning

Reports and Related Files

None

NonReusableParametricModule-ML

This tag categorizes parameters based on the usage of the parameter

Description

The NonReusableParametricModule-ML tag identifies the flavors of parameters based on the usage of the parameter.

The tag reports violation with Error severity, if the parameter is used as constant or if conditions in the design (assignment, comparison, and so on). In addition, the tag reports violation with Info severity, if the parameter is affecting width of bus.

Parameter(s)

[report_for_submodules](#): Default value is no. Set the value of the parameter to yes to report violation for sub modules.

Constraint(s)

None

Message Details

Message 1

[Error] Parameter <parameter-name> of module <module-name> is used as constant/condition to distinguish between flavors

Message 2

[Info] Parameter <parameter-name> of module <module-name> is used as physical determinant to distinguish between flavors

Example Code and/or Schematic

Example 1

```
module design_top(...);  
    parameter NUM_PORTS = 3;    // violation for NUM_PORTS  
    with Error Severity  
    ...
```


MoreLint Tags

```
    logic [3:0] SigA;
    assign SigA = NUM_PORTS;
    ...
endmodule
```

Example 2

```
module design_top (...);
    parameter IS_MC_2LM = 0; // violation for IS_MC_2LM
    with Error Severity
    ...
    generate if (IS_MC_2LM) begin
        ...
    end else begin
        ...
    end
    endgenerate

    always_comb begin
        if (IS_MC_2LM)
            SigA = <something>;
        else
            SigA = <something else>;
        end
    end
    ...
endmodule
```

Example 3

```
module design_top (...);  
    parameter BUS_WIDTH = 0; // violation for BUS_WIDTH  
    with Info Severity  
    ...  
    logic [BUS_WIDTH -1:0] MyBus;  
    ...  
endmodule
```

Example 4

```
module design_top (...);  
    parameter PARAM1 = 0; // violation for PARAM1 with Info  
    Severity  
    ...  
    logic [PARAM1 -1:0] MyBus;  
    logic [3:0] SigA;  
    assign SigA = PARAM1;  
    ...
```

Tag Severity

Error

Related Reports and Files

None

PackedArrayInRange-ML

Packed arrays should not be used as range values in a range.

Description

Packed arrays in ranges often result in large combinational logics and synthesis stage and consequent analysis stage may take longer time.

Though it is allowed to do so, designer should be aware which type of packed arrays are used and what are their ranges. Sometimes the designer could have mistakenly applied a packed array as range value though he was supposed to index it correctly.

This tag detects such scenarios and reports a violation a packed array is used as a range value.

Note that the tag considers:

- Only parameter and localparam packed arrays
- Only ranges at signal declarations

Also, the tag ignores unpacked arrays as they result in syntax error.

Parameter(s)

None

Constraint(s)

None

Example Code and/or Schematic

Example 1

```
parameter [1:0][15:0] SIZE = '{8, 8};
parameter DESIGN_CONFIG = 0;
`define DNZ(x) x[DESIGN_CONFIG]

typedef struct packed{
    logic x;
    logic y;
```

```
        logic [(SIZE)-1:0] z; // violation
    } DES;

module top (a, b, c);
    input a;
    input b;

    output DES c [`DNZ(SIZE)-1:0]; // no violation

    always_comb begin
        for(int i=0; i<`DNZ(SIZE); i++) begin
            c[i].x = !(a & b);
        end
    end
endmodule
```

Example 2

```
typedef struct packed{
    logic x;
    logic y;
    logic [9:0] z;
} DES;

parameter [1:0][15:0] SIZE = '{8, 8};
parameter DESIGN_CONFIG = 0;
`define DNZ(x) x[DESIGN_CONFIG]
```

MoreLint Tags

```

module top (a, b, c);
    input a;
    input b;

    output DES c [(SIZE)-1:0]; // violation

    always_comb begin
        for(int i=0; i<`DNZ(SIZE); i++) begin
            c[i].x = !(a & b);
        end
    end
endmodule

```

Example 3

```

typedef struct packed{
    logic x;
    logic y;
    logic [9:0] z;
} DES;

parameter [1:0][15:0] SIZE = '{8, 8};
parameter DESIGN_CONFIG = 0;
`define DNZ(x) x[DESIGN_CONFIG]

module top (a, b, c);
    input a;
    input b;

```

```
output DES c [`DNZ(SIZE)-1:0]; // no violation

always_comb begin
    for(int i=0; i<`DNZ(SIZE); i++) begin
        c[i].x = !(a & b);
    end
end

endmodule
```

Example 4

```
parameter [1:0][15:0] SIZE = '{8, 8};
parameter DESIGN_CONFIG = 1;
`define DNZ(x) x[DESIGN_CONFIG]

module top (a, b, c);
    input a;
    input b;

    output reg c [(SIZE)-1:0]; // violation

    always_comb begin
        for(int i=0; i<`DNZ(SIZE); i++) begin
            c[i] = !(a & b);
        end
    end
end
```

```
endmodule
```

Example 4

```
module top (a, b, c);  
    parameter [1:0][15:0] SIZE = '{8, 8};  
    input a, b;  
    output c [SIZE-1:0]; // violation  
  
    assign c[0] = a & b;  
endmodule
```

Example 5

```
module top (a, b, c);  
    parameter [1:0][15:0] SIZE = '{8, 8};  
    input a, b;  
    output c [SIZE[0]-1:0]; // no violation  
  
    assign c[0] = a & b;  
endmodule
```

Example 6

```
module top (a, b, c);  
    parameter [1:0] SIZE [15:0] = '{8, 8};  
    input a, b;  
    output c [SIZE-1:0]; // results in STX error, no  
violation needed  
    assign c[0] = a & b;  
endmodule
```

Example 7

```
module top (a, b, c);  
    input a, b;  
    output c [8-1:0];  
  
    parameter [1:0][15:0] SIZE = '{8, 8};  
    wire w [SIZE-1:0]; // violation  
    assign c[0] = a & b;  
endmodule
```

Example 8

```
module top (a, b, c, w);  
    parameter [1:0][15:0] SIZE = '{8, 8};  
    input a, b;  
    output c [8-1:0];  
  
    output w [SIZE-1:0]; // violation  
    assign c[0] = a & b;  
endmodule
```

Example 9

```
module top (a, b, c);  
    input a, b;  
    output c [8-1:0];  
  
    parameter [1:0][15:0] SIZE = '{8, 8};  
    wire w [SIZE-1:0]; // violation
```


MoreLint Tags

```
        assign w[0] = a & b;  
    endmodule
```

Example 10

```
module top (a, b, c);  
    parameter [1:0][15:0] SIZE = '{8, 8};  
    input a [(SIZE)-1:0], b [(SIZE)-1:0]; // violation  
    output c;  
  
    assign c = a[0] & b[0];  
endmodule
```

Example 11

```
module top (a, b, c);  
    localparam [1:0][15:0] SIZE = '{8, 8};  
    input a, b;  
    output c [(SIZE)-1:0]; // violation  
  
    assign c[0] = a & b;  
endmodule
```

Tag Severity

Warning

Reports and Related Files

None

SameAssignment-ML

Report same assignment expression in different branches of a conditional block.

Description

This tag flags violations, if same assignment expression presents multiple times in different branches of a conditional block.

Parameter(s)

- [report_case_block](#): By default, the value of the parameter is set to `yes` and the tag reports violations for same assignment expressions in `case` blocks. Set the value of the parameter to `no` to not report violations for same assignments in `case` blocks.
- [report_default_branch](#): By default, the value of the parameter is set to `yes` and the tag reports violations for same assignment expressions in the default branch of `case` blocks. Set the value of the parameter to `no` to not report violations for same assignment expressions in the default branch of `case` blocks.
- [report_if_block](#): By default, the value of the parameter is set to `yes` and the tag reports violations for same assignment expressions in `if-else` blocks. Set the value of the parameter to `no` to not report violations for same assignment expressions in `if-else` blocks.
- [report_generate_block](#): By default, the value of the parameter is set to `no` and the tag does not report violations for same assignment expressions in `generate-case` and `if` blocks. Set the value of the parameter to `yes` to report violations for same assignment expressions in `generate-case` and `if` blocks.
- [use_detailed_message](#): By default, the parameter is set to `no` and the violation message reported by the tag does not include the information about the first occurrence of the assignment. Set the parameter to `yes` to enable the tag to include information about the first occurrence of the assignment and the corresponding file name in the violation message.

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Signal '<LHS>' is assigned with same value in different conditional branches [Hierarchy: <hierarchy>]

If the use_detailed_message parameter is set to yes, the tag reports the following:

[WARNING] Signal '<LHS>' is assigned with same value in different conditional branches
[<file_name_of_the_first_occurrence>:<line_number_of_the_first_occurrence >][Hierarchy: <hierarchy>]

Potential Issues

The violation message is reported when the LHS of assignment has the same value in different conditional branches.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

Click the violation message to highlight the RTL line where multiple assignments present in RTL code.

Example Code and/or Schematic

Example 1

```
module test(in1, in2, in3, out1, out2, out3);
input in1, in2, in3;
output reg out1, out2, out3;
reg out_r1, out_r2, out_r3;

always @(in1, in2 , in3) begin
    if (in1)
        out_r1 = in2;
    else
        out_r1 = in2; // violation
end
```

```
always @(in1, in2 , in3) begin
    case(in1)
        1'b0 :out_r2 = in2;
        1'b1 :out_r2 = in2; // violation
    endcase
end

assign out_r3 = in1 ? 1'b1 : 1'b1; //violation

assign out1 = out_r1;
assign out2 = out_r2;
assign out3 = out_r3;
endmodule
```

Example 2

```
module test(in1, in2, in3, out1, out2, out3);
input in1, in2, in3;
output reg out1, out2, out3;
reg out_r1, out_r2, out_r3;

always @(in1, in2 , in3) begin
    if (in1)
        out_r1 = in2;
    else if(in3)
        out_r1 = in2; // violation
    else
        out_r1 = in2; // violation
end

always @(in1, in2 , in3) begin
    case(in1)
        1'b0 :    out_r2 = in2 + in3;
        default : out_r2 = in3 + in2; // violation
    endcase
end
```

MoreLint Tags

```

assign out1 = out_r1;
assign out2 = out_r2;
endmodule

```

Example 3

```

module test(in1, in2, in3, out1, out2, out3);
input [1:0] in1, in2, in3;
output reg [1:0] out1, out2, out3;
reg [1:0] out_r1, out_r2, out_r3;

```

```

always @(in1, in2 , in3) begin
    if (in1)
        out_r1[0] = in2;
    else
        out_r1[1] = in2; // no violation
end

```

```

always @(in1, in2 , in3) begin
    case(in1)
        1'b0 :out_r2 = in2[0];
        1'b1 :out_r2 = in2[1]; // no violation
    endcase
end

```

```

assign out_r3 = in1[0] ? {in1[1],in1[0]}: {in1[0],in1[1]}; //no violation

```

```

assign out1 = out_r1;
assign out2 = out_r2;
assign out3 = out_r3;
endmodule

```

Example 4

```

module test(in1, in2, in3, out1, out2, out3);
input [1:0] in1, in2, in3;
output reg [1:0] out1, out2, out3;
reg [1:0] out_r1, out_r2, out_r3;

```

```
always @(in1, in2 , in3) begin
    if (in1) begin
        out_r1[0] = in2[0];
        out_r1[1] = in3[0];
    end
    else begin
        out_r1[1] = in2[1];
        out_r1[0] = in2[0]; //violation
    end
end

assign out1 = out_r1;
endmodule
```

Default Severity Label

Warning

Reports and Related Files

No related reports or files.

SupplyNet-ML

Signal is defined as Supply0/1.

When to Use

Use this tag to detect signals that are defined as Supply0/1.

Description

The SupplyNet-ML tag reports violation for signals, which are defined as Supply0/1.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[ERROR] Signal <Variable Name> is defined as Supply<0 or 1>

Potential Issues

The violation message is reported when the signal is defined as supply0/1.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

To debug the violation, click the violation message. This displays the

To fix the violation, ensure that the signal is not defined as supply0/1.

Example Code and/or Schematic

Example 1

```
supply0 ex1; // Violation
interface intf();
    supply0 ex2; // Violation
endinterface;
```

```
package myPkg;
    supply1 ex3; // Violation
    supply0 ex4; // Violation
endpackage;

module test_supply (input a, input b, output c);
    supply0 wire_a; // Violation
    supply1 wire_b; // Violation

import myPkg::*;
test_supply_inst inst (.a(wire_a), .b(wire_b), .c(c));
endmodule

module test_supply_inst (input a, input b, output c);
    assign c = a && b;
endmodule
```

Example 2

```
//inc.v
supply0 ex1; // Violation
supply1 ex2; // Violation
supply0 ex3; // Violation

//end inc.v

//file test.v
`include "inc.v"

module test_supply (input a, input b, output c);
    supply0 wire_a;
    supply1 wire_b;
    test_supply_inst inst (.a(wire_a),
                           .b(wire_b),
                           .c(c));
endmodule

module test_supply_inst (input a, input b, output c);
```

MoreLint Tags

```
assign c = a && b;  
endmodule  
//end of file test.v
```

Default Severity Label

Error

Reports and Related Files

None

SV09Construct-ML

System Verilog 2009 construct used

Description

The SV08Construct-ML tag reports violation wherever System Verilog 2009 construct is used.

Parameter(s)

None

Constraint(s)

None

Message Details

system verilog construct (<construct>) found

Example Code and/or Schematic

Currently Unavailable

Tag Severity

Info

Reports and Related Files

None

TimescaleUsed-ML

Timescale usage found in the design

Description

This tag will report violations for timescale directives in the design.

Parameter(s)

None

Constraint(s)

None

Message Details

Timescale directive with value '<value>' is used in the design

Example Code and/or Schematic

The following example shows a violating scenario for the TimescaleUsed-ML tag:

```
`timescale 100ps/10ps // violation
module test ();
    `timescale 100ns/10ps // violation
    time a;
    initial begin
        $monitor ("%g a = %t", $time, a);
    #1 a = 1ns;
    #1 a = 0.2ns;
    #1 a = 300ps;
    #1 $finish;
    end
endmodule
`timescale 10ns/10ps // violation
```

Tag Severity

Info

Reports and Related Files

None

PortTypeMismatch-ML

Port type mismatch between module declaration and net connection in the instantiation.

When to Use

Use this tag detect mismatch between port type of module declaration and type of net connected to the port in instantiation.

Description

This tag reports violations, if port type of module declaration and type of net connected to the port in instantiation do not match.

Parameter(s)

valid_conn_types: Default value is "reg:wire , wire:reg". Set the value of the parameter to specify valid type mismatches.

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Port '<port name>: <port type>' of module <module name>' is not matched with net '<net name>: <net type>' in the instantiation '<instance name>' [Hierarchy: '<hierarchy>']

Potential Issues

The violation message is reported when there is a mismatch between module declaration and net connection in the instantiation

Consequences of Not Fixing

Not fixing the violation may result in unexpected design behavior.

How to Debug and Fix

To debug the violation, click the violation message. This displays the terminal connection present in RTL in the HDL viewer

To fix the violation, ensure that the port type of module declaration and type of net connected to the port in instantiation match.

Example Code and/or Schematic

Example 1

```
module top();
  wire [7:0] b;
  test test(.a($signed(b))); //violation - signed net is
  connected to unsigned port
endmodule

module test(a);
  input [7:0] a;
endmodule
```

Example 2

```
module top();
  wire [31:0] b;
  test test(.a(b)); // violation - unsigned vector net is
  connected to signed integer port
endmodule

module test(a);
  input integer a;
endmodule
```

Example 3

```
typedef struct packed
{
  logic [15:0] opA;
  logic [15:0] opB;
} my_opcode_struct_t;

module top();
  wire signed [31:0] b;
  wire [31:0] d;
  test test(.a(b), //violation - signed vector net is
  connected to signed integer port
```

MoreLint Tags

```
.c(d) ); //violation - unsigned vector net is connected
to packed struct port
```

```
endmodule
```

```
module test(a,c);
  input integer a;
  input my_opcode_struct_t c;
endmodule
```

Example 4

```
typedef reg[7:0] myReg;
```

```
module top();
  wire a;
  reg [7:0]b;
  logic c;
  test test(.p(a), //violation - wire net is connected to
reg port
  .q(b), //violation - reg vector net is connected to
'myReg' port
  .r(c)); //violation - logic net is connected to reg
port
```

```
endmodule
```

```
module test(p,q,r);
  input reg p;
  input myReg q;
  input reg r;
endmodule
```

Tag Severity

Warning

Reports and Related Files

None

ConcatInUnpacked-ML

Unpacked array concatenation must be avoided

When to Use

Use this tag to identify assignments where RHS of the assignment is an unpacked array concatenation.

Description

As per IEEE 1800-2005(enableSV) assignment to unpacked type through concatenation is not allowed and VC SpyGlass reports STX message for it.

But, IEEE 1800-2009(enableSV09) added a new construct unpacked array concatenation and no VC SpyGlass STX message is reported.

Parameter(s)

None

Example

```
module top(input a);  
    bit sig3 [1:0];  
    assign sig3 = {1'b1, 1'b0}; // violation  
ConcatInUnpacked-ML  
endmodule
```

Messages and Suggested Fix

Unpacked array concatenations are avoided as these could be mixed-up with assignment patterns without preceding apostrophe which are deprecated

Tag Severity

Error

Reports and Related Files

None

MaxFanoutCount-ML

High fan-out detected for internal node.

Language

Verilog, VHDL

Description

The MaxFanoutCount-ML tag flags internal nodes that have a fan-out greater than the specified number.

By default, the maximum number of fan-outs allowed for internal nodes is 300. Use the [int_net_fanout](#) parameter to change the default value. You can specify a different integer value for the [int_net_fanout](#) parameter to change the default value.

The MaxFanoutCount-ML tag does not report nets directly connected to primary input ports.

Message Details

The following message appears at the location where an internal net `<name>` has a fanout `<num>` greater than 300:

Internal net '<name>' has fanout <num>

Severity

Warning

PortConnectedToMux-ML

MUX detected in fan-in cone of specified pin.

When to Use

Use this tag to detect the MUXes in the fan-in cone of a specified pin.

Description

The *PortConnectedToMux-ML* tag reports all the MUXES (RTL or Lib Cell) in the combinational fan-in cone of the pin specified using the [instance_based_pin_names](#) or [module_based_pin_names](#) parameters.

The tag will report violations for internal names of MUX instances as well.

Parameter(s)

- [instance_based_pin_names](#): Default value is " ". Specify a comma-separated hierarchical instance-based pin list to identify the pins which needs to be analyzed by the tag.
- [module_based_pin_names](#): Default value is " ". Specify a comma-separated module-based pin list to identify the ports of module which needs to be analyzed by the tag.

Constraint(s)

None

Messages and Suggested Fix

[WARNING] MUX instance '<instance name>' of cell '<cell type>' (Parent: '<module name>') detected in the fan-in cone of pin '<hierarchical pin name>'

Potential Issues

The violation message is reported when a MUX is detected in the fanin cone of a specified pin.

Consequences of Not Fixing

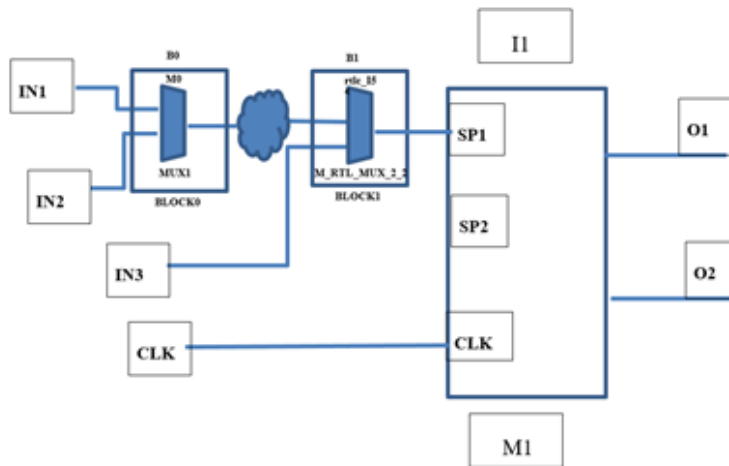
MUXES inferred from RTL, implemented as combinational logic can cause glitch in the design.

How to Debug and Fix

Click the violation message to highlight the violating scenario in the HDL viewer.

Example Code and/or Schematic

Consider the following example:



As shown in the above figure, at RTL stage there might be presence of certain modules having RTL (or lib cell) Mux instances, like above module BLOCK0 & BLOCK1, instances - M0 & rtlc_I5 MUXES.

The *PortConnectedToMux-ML* tag reports violation for the following:

- Each pin provided (Top.I1.SP1)
- Full hierarchical name to MUX instance - Top.B0.M0
- Module name of the MUX instance (user provided, tool inferred) - MUX1, M_RTL_MUX_2_2
- Parent Module of the MUX instance, like - BLOCK0, BLOCK1 in above case.

Default Severity Label

Warning

MoreLint Tags

Reports and Related Files

None

UnusedFlopOutput-ML

Unreachable sequential element must be removed from the design

Language

Verilog, VHDL

Description

The UnusedFlopOutput-ML tag checks usage of the output pins of the sequential elements.

The UnusedFlopOutput-ML tag identifies unreachable sequential elements when any of the following scenarios occur for each of the sequential element output pin:

- An unloaded sequential element output pin
- The output pin of the sequential element is only connected (via glue logic) to another sequential element that turns out to be unreachable
- The path is blocked. The tag behavior is sensitive to case-analysis settings
- The fan-out cone of the sequential element output pin contains neither a primary output port nor a sequential element that turns out to be "NOT unreachable", nor a black box

Violation will not be reported for block path due to 'design structure'.

If the fan-out cone of sequential element contains unused as well as block paths, then violation will be reported (either unused or blocked) according to last traversed fan-out path, although all paths will be highlighted in schematic (unused as well as blocked). Blocked paths will be highlighted in red color.

By default, the UnusedFlopOutput-ML tag checks all the sequential elements in the design. Use the [ignore_elements](#) parameter to specify the sequential elements that should be considered as NOT unreachable.

Use the [showReducedSchematic](#) parameter to control the extent of details displayed in the Schematic window of SpyGlass. Set this parameter to `max` if the reason (`set_case_analysis`) for blocked output pin needs to be reported.

Use the [REPORT_UNUSEDFLOP_COUNT](#) parameter to report number of unused D flip-flops in the design. By default, the value of the parameter is

no. In this case, the tag reports a violation for unreachable sequential elements. Set the value of the parameter to yes to report number of unused D flip-flops in the design, in addition to, the violation for unreachable sequential elements.

NOTE: *The UnusedFlopOutput-ML tag may be runtime intensive tag. Therefore, it is not recommended to be used as a general tag. You should run this tag only when needed.*

Message Details

Message 1

The following message appears when an unused output pin of a sequential element `<seq-element-name>` of type `<obj-type>` is found:

[WARNING] Unused output pin(s) of '`<seq-element-name>`' (type '`<obj-type>`') detected

where `<obj-type>` can be flip-flop, or latch, or library cell.

Message 2

The following message appears when a blocked (due to reasons other than `set_case_analysis`) output pin of a sequential element `<seq-element-name>` of type `<obj-type>` is found:

[WARNING] Blocked output pin(s) of '`<seq-element-name>`' (type '`<obj-type>`') detected

where `<obj-type>` can be flip-flop, or latch, or library cell.

Message 3

The following message appears when a `set_case_analysis` blocked output pin of a sequential element `<seq-element-name>` of type `<obj-type>` is found:

[WARNING] Blocked (due to `set_case_analysis`) output pin(s) of '`<seq-element-name>`' (type '`<obj-type>`') detected

where `<obj-type>` can be flip-flop, or latch, or library cell.

NOTE: *The **Message 3** is reported only when the [showReducedSchematic](#) parameter is set to max.*

Severity

Warning

Example (VHDL)

Consider the following example where unused output pin mydffob2 and mydffob1 are flagged:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
```

```
ENTITY test IS
  PORT (
    in0  : IN  STD_LOGIC;
    in1  : IN  STD_LOGIC;
    rst  : IN  STD_LOGIC;
    clk  : IN  STD_LOGIC;
    out0 : OUT STD_LOGIC
  );
```

```
END;
```

```
ARCHITECTURE rtl OF test IS
```

```
SIGNAL intsiguu : STD_LOGIC;
SIGNAL mydffob1 : STD_LOGIC;
SIGNAL mydffob2 : STD_LOGIC;
SIGNAL mydffob3 : STD_LOGIC;
```

```
BEGIN
```

```
PROCESS(rst, clk)
```

```
BEGIN
```

```
  IF rst = '0' THEN
    mydffob1 <= in0;
    mydffob2 <= in0;
    mydffob3 <= in0;
```


MoreLint Tags

```
ELSIF clk'EVENT AND clk = '1' THEN
    mydffob1 <= in1;
    mydffob2 <= in1;
    mydffob3 <= in1;
END IF;
END PROCESS;

intsiguu <= not (mydffob2);

out0      <= not (mydffob3);

END;
```

VC SpyGlass Lint generates the following messages for this example:

Unused output pin(s) of 'test.mydffob2' (type 'flop') detected

Unused output pin(s) of 'test.mydffob1' (type 'flop') detected

ResetPreventSRL-ML

Reset used in shift register will prevent SRL inference

Language

Verilog, VHDL

Description

The ResetPreventSRL-ML tag flags inferred shift register logic with reset sequence.

A shift register with reset sequence is inferred when a sequence of cascaded flip-flops (at least three in number), which have the same clock source and the same reset source, and each subsequent flip-flop after the first one has its d-pin connected to the q-output of the previous flip-flop, without any intervening logic.

NOTE: *The ResetPreventSRL-ML tag supports asynchronous reset only.*

By default, the ResetPreventSRL-ML tag flags shift register inferred from a sequence of 3 cascaded flip-flops. Use the [minShiftRegLength](#) parameter to set a different number.

The ResetPreventSRL-ML tag ignores shift registers logic without any reset logic.

Using the SRL primitive is recommended over inferring shift register logic in the design as this primitive is more efficient in terms of area and performance. The SRL primitive, however, does not have any reset function and thus, cannot be used for shift registers that have a reset sequence.

By default, the number of messages reported by the ResetPreventSRL-ML tag is 500. Set the [allviol](#) parameter to *yes* or *<tag-name>* to report all violation messages.

Message Detail

The following message appears at the first line of a design unit in which a shift register with reset sequence is inferred:

Reset used in shift register will prevent SRL inference

Severity

Warning

LINT_DETECT_EMPTY_MODULE

Reports violations for empty modules in RTL design.

When to Use

Use this tag to detect empty modules in the design.

Description

This tag reports violations for empty modules. It reports a violation, if a module definition is present in the RTL without any logic inside the module. Any assignment (procedural or continues) or instantiations (sub module, lib cell, and so on) will be considered as logic.

However, this tag does not reports library modules, whose functional view is missing in the specified .sglib files.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[ERROR] '<module name>' is an empty module

Potential Issues

The violation message is reported when an empty module is present in the design.

Consequences of Not Fixing

Not fixing the violation may result in unexpected tag behavior.

How to Debug and Fix

To debug the violation, click the violation message to highlight the empty module in the HDL viewer.

Example Code and/or Schematic

Example 1

```
module test(input din, output dout); //No volition
sub S1 ( .sin(din), .sout(dout));
assign dout = din;
endmodule

module sub (input sin, output sout); //Violation - 'sub'
is an empty module
endmodule
```

Example 2

```
module test(input din, output dout); //No violation
assign dout = din;
endmodule

module sub (input sin, output sout); //Violation - 'sub'
is an empty module
endmodule
```

Example 3

```
module test(input din, output dout); //No violation
sub S1 ( .sin(din), .sout(dout));
endmodule

module sub (input sin, output sout); //No violation
assign sin = 1'b1;
assign sout = 1'b0;
endmodule
```

Example 4

```
module test(input din, output dout); //No violation
```

MoreLint Tags

```
sub S1 ( .sin(din), .sout(dout));
endmodule
```

```
module sub (input sin, output sout); //Violation - 'sub'
is an empty module
parameter W = 2'b000;
localparam T = 2'b11;
endmodule
```

Example 5

```
module test(input din, output dout); //No violation
```

```
sub S1 ( .sin(din), .sout(dout));
endmodule
```

```
module sub (input sin, output sout); //No violation
```

```
reg [2:0] a;
assign a = 3'b101;
endmodule
```

Example 6

```
module test(input din, output dout); //No violation
```

```
sub S1 ( .sin(din), .sout(dout));
endmodule
```

```
module sub (input sin, output sout); //Violation - 'sub'
is an empty module.
```

```
reg [2:0] a;
endmodule
```

Default Severity Label

Warning

Reports and Related Files

None

LINT_IMPROPER_RANGE_INDEX

Possible discrepancy in the range index or slice of an array

Description

The LINT_IMPROPER_RANGE_INDEX tag reports a violation if the number of bits required to cover index does not match $\log_2 N$. For example, if the size of a signal is ten bits, so a four-bit signal is required to cover its range. Therefore, if any signal is used as bit-select/part-select and is not of four bits, a violation is reported. Or, a violation is reported if signal used in a bit-select is less than four bits or greater than four bits.

NOTE:

- *The tag checks the index at all possible places in the module*
- *Analysis is done for the non-static indexes only*

Language

Verilog

Parameters

- ***nocheckoverflow***: By default, this parameter is set to `yes` and the width is calculated according to the LRM and natural width is considered for constants. If you set the value of the `nocheckoverflow` parameter to `no` then the width is calculated considering the best fit width of an expression. That is, the width in which maximum value of an expression can be accommodated.
- ***report_max_val_overflow***: By default, the value of the parameter is `no`. Set the value of the parameter to `yes` to report violation when maximum value that can be represented by the index is larger than the maximum value of the range of the signal being indexed.

Message Details

The following message appears when there is a discrepancy in the range index or slice of an array:

[WARNING] Index '<index-string>' of width '<bit-number>' is <larger/fewer> than the width '<number>' required for the max value '<number>' of the signal '<signal-name>' [Hierarchy :

<hier-name>]

Index '<index-string>' of width '<bit-number>' can have max value '<number>' which is greater than the required max value '<value>' of the signal '<signal-name>' [Hierarchy: <hier-name>]

Example and /or Schematic

Example 1

Consider the following example where the tag does not report violation because b4 covers all the bits of a4 and width of b4 is equal to the width of 8:

```
wire [8:0] a4;  
wire [3:0] b4;  
assign c4 = a4[b4]; // No violation
```

Example 2

Consider the following example where the tag reports a violation:

```
wire [8:0] a4;  
wire [2:0] b4;  
assign c4 = a4[b4]; // Violation
```

Example 3

Consider the following example where the tag reports a violation:

```
wire [8:0] a4;  
wire [5:0] b4;  
assign c4 = a4[b4]; // Violation
```

Example 4

Consider the following example where the tag does not report violation:

```
wire [7:0] a4;  
wire [2:0] b4;  
assign c4 = a4[b4]; // No violation
```

Example 5

Consider the following example where the tag does not report violation because the LRM width of b4+b5 is "3" to honor self-determined expressions:

```
wire [7:0] a4;
```



```
wire [2:0] b4; wire [2:0] b5;
assign c4 = a4[b4+b5]; // No violation
```

Example 6

Consider the following example where the tag does not report violation for the first index 'b4', whereas reports a violation for the second index 'b4':

```
wire [7:0] [3:0]a4;
wire [2:0] b4;
assign c4 = a4[b4][b4];
```

Example 7

Consider the following example:

```
assign a [0][1:0]= b [0][c:0];
```

In this case, a syntax gets flashed because non-static c cannot be present in slice.

Example 8

Consider the following example where the tag does not report violation for the first index 'b4.' The tag also does not report violation for the second index 'b4 [b3]' as it is a bit select and 1 bit and parent range of a4 is [0:0], which is also 1 bit. The tag reports a violation for b3.

```
wire [7:0] [0:0]a4;
wire [2:0] b4;
wire [2:0] b3;
assign c4 = a4[b4][b4[b3]];
```

Example 9

Consider the following example where the tag does not report violation because b4 covers the a4 bits:

```
wire [50001:50000]a4;
wire [15:0] b4;
assign c4 = a4[b4]; // No violation
```

Example 10

Consider the following example where the tag reports a violation because

b4 bits does not cover any of a4 bits:

```
wire [50001:50000]a4;  
wire [1:0] b4;  
assign c4 = a4[b4]; // Violation
```

Example 11

Consider the following example where the tag reports a violation. Though the width of the range of a4 (the width of -8 is 4) matches the width of b4, but violation is still reported, because b4 doesn't cover all a4 bits:

```
wire [-8: 7]a4;  
wire [3:0] b4;  
assign c4 = a4[b4];
```

Example 12

Consider the following example:

```
wire [-8: 7]a4;  
wire signed [3:0] b4;  
assign c4 = a4[b4]; // No violation
```

Example 13

Consider the following example where the tag reports a violation because b4 does not cover all the ranges of a4, maximum value of b4 is 7 and it does not cover 8:

```
wire [-1: 8]a4;  
wire signed [3:0] b4;  
assign c4 = a4[b4]; // Violation
```

Example 14

Consider the following example where the tag reports a violation because b4 does not cover all the ranges of a4, minimum value of b4 is 0 and it does not cover -1:

```
wire [-1: 8]a4;  
wire [3:0] b4;  
assign c4 = a4[b4]; // Violation
```

Example 15

Consider the following example where the tag does not report violation because the width of 8 is 5 for signed ranges (-16 to +15 has width 5, -8 to

MoreLint Tags

+7 has width 4):

```
wire [-1: 8]a4; //signed range as one of the index is -ve
wire signed [4:0] b4;
assign c4 = a4[b4]; // No violation
```

Example 16

Consider the following example where the tag reports a violation because the LRM width of $b4+c4$ is 3 only for self determined expressions while width of 8 in $a4$'s range is 4:

```
wire [8: 0]a4;
wire [2:0] b4;
wire [2:0] c4;
assign c4 = a4[b4 + c4]; // Violation
```

Example 17

Consider the following example where the tag reports a violation because the width of $b4 [2 : 0]$ is 3 and it does not match with the width of numeric 8 (width: 4) in the range $a4$:

```
wire [8: 0]a4;
wire [8:0] b4;
assign c4 = a4[b4[2:0]]; // Violation
```

Default Severity Label

Warning

LINT_INSTANCE_WITHOUT_NAME

Instance without name

Description

The LINT_INSTANCE_WITHOUT_NAME tag reports violation for instantiations that do not have any name. This includes instantiation from a module, LIB cell, UDP, or an interface.

Language

Verilog

Message Details

The following message appears when an instance is instantiated without name:

```
[WARNING] <cell-type> '<cell-inst-name>' instantiated without name
```

Example and /or Schematic

Example 1

Consider the following example where the tag reports a violation because module mod is instantiated without name:

```
module TOP();  
    mod (); // Violation  
endmodule
```

```
module mod(input in, output out);  
    assign out = in;  
endmodule
```

Example 2

Consider the following example where the tag reports a violation because LIB cell is instantiated without name:

```
module TOP();  
    CTG4_DHY (.CLK(CP)); //Violation
```

```
endmodule
```

Example 3

Consider the following example where the tag reports a violation because UDP module is instantiated without name:

```
module TOP();
    test_udp_mod(); //Violation
endmodule

primitive test_udp_mod(sum, cin, a, b);
    output sum;
    input cin,a,b;
    table
    0 0 0 : 0;
    0 0 1 : 1;
    endtable
endprimitive
```

Example 4

Consider the following example where the tag reports a violation because interface is instantiated without name:

```
module TOP();
    BusInterface (); //Violation
endmodule

interface BusInterface;
    logic [7:0] Addr, Data;
    logic RWn;
endinterface
```

Default Severity Label

Warning

LINT_LOOP_THRU_FLOP_WITHOUT_SETRESET

Avoid synchronous loops that contain flip-flops without any set/reset

Description

The LINT_LOOP_THRU_FLOP_WITHOUT_SETRESET tag reports violations for synchronous loops that contain flip-flops without any set/resets.

The feedback loops that contain flip-flops create paths whose timing is difficult to analyze. In fact, even if it is assumed that they were false paths and merely feedback, a timing analysis tool would not understand them. Such circuits may be acceptable on behavior but create problems in timing analysis and should not be used.

You can use the [overlappingLoops](#) parameter to get violations for all the overlapping loops.

In case the violation messages contain more than ten nets, the messages display only first ten nets of the loop, otherwise complete message is displayed.

The tag analyzes combinational blocking and non-blocking paths when some constant value is reached to the terminal of combinational instance, for example, (and/or gate) through set_case_analysis or supply net.

NOTE: *The tag does not analyze blocking and non-blocking paths for sequential logic.*

For a complete list of nets in the loop, see the LINT_LOOP_THRU_FLOP_WITHOUT_SETRESET.rpt report. This report is generated only for the violating cases.

The following is a sample of this report:

```
===== Sequential Loop Message report=====
SNO      FILE                LINE      Message
=====
1        <file-name>         <Line-Num>      m1_1.out1-m1_1.D1
```

Language

Verilog, VHDL

Parameters

- ***allviol***: Default value is `no` and the tag reports maximum of 500 violations. Set this parameter to `yes` or tag name to report all the violations.
- ***overlappingLoops***: Default value is set to `yes` and the tag detects all of the overlapping combinational loops. Set this parameter to `no` to not report overlapping combinational loops.

Message Details

Message 1

The following message appears if synchronous loops are found that contain flip-flops without any set/resets:

[WARNING] Synchronous loop from register without set/reset exists on '<signal-name>'

Message 2

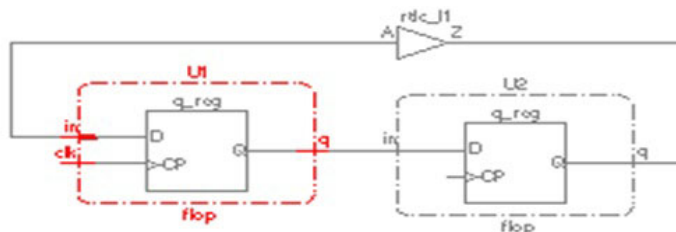
The following message appears when the LINT_LOOP_THRU_FLOP_WITHOUT_SETRESET.rpt report is generated:

[INFO] Please refer to 'LINT_LOOP_THRU_FLOP_WITHOUT_SETRESET.rpt' for complete synchronous loops

Example and/or Schematic

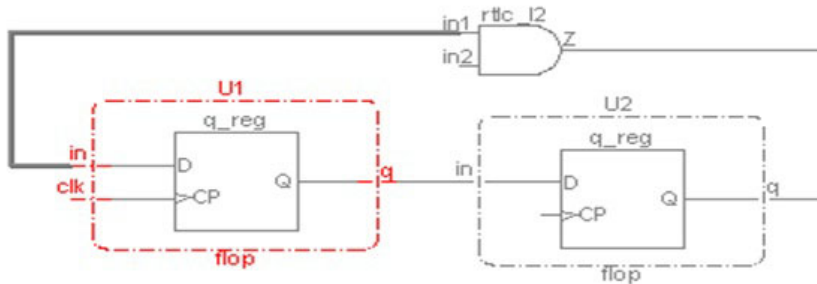
Example 1

Consider the following example where the tag reports violation because a sequential loop exists and flip-flop does not have any set/reset:



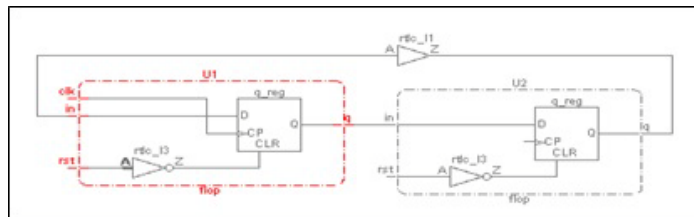
Example 2

Consider the following example where the tag reports violation based on the blocking-path analysis as the path is found to be unblocked:



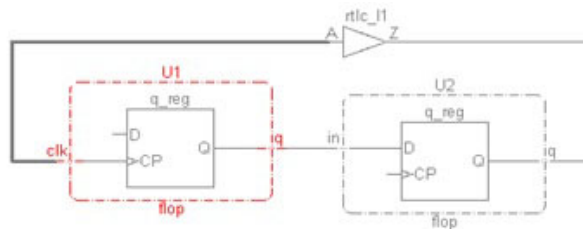
Example 3

Consider the following example where the tag does not report any violation for sequential loop because the flip-flop contains set/reset:



Example 4

Consider the following example where the tag reports violation because the sequential loop is through clock pin of the first flip-flop:



MoreLint Tags

Default Severity Label

Warning

LINT_LOGICAL_EXPR_USED_IN_SENS_LIST

Logic expression used in sensitivity list

Description

The *LINT_LOGICAL_EXPR_USED_IN_SENS_LIST* tag reports a violation if a logic expression is used in the sensitivity list of an always block.

Language

Verilog

Message Details

The following message appears when a logic expression is used in the sensitivity list of an always block:

```
[WARNING] logic expression <expression> used in sensitivity list
```

Example and /or Schematic

Consider the following example:

```
module test (count, clk1, clk2, rst_p);  
input clk1, clk2, rst_p;  
output [7:0] count;  
reg [7:0] count;  
always @(posedge (clk1 | clk2) or //warning  
posedge rst_p)  
if (rst_p)  
count = 0;  
else  
count = count + 1;  
endmodule
```

MoreLint Tags

Default Severity Label

Warning

LINT_ALL_BITS_SHIFT_OUT

All bits shifted out

Description

The *LINT_ALL_BITS_SHIFT_OUT* tag reports a violation if shift operation are used, which shift out all bits of a signal.

NOTE: *The tag considers both left and right shift operation to check if all bits are shifted out or not.*

Language

Verilog

Message Details

The following message appears if all bits of a signal are shifted out:

[WARNING] All bits of signal '<signal-name>' are shifted out. [Hierarchy: '<hier-name>']

Example and/or Schematic

Example 1

Consider the following example where the tag reports a violation because all bits are shifted out:

```
module test (q, sig);
    input [7:0] sig;
    output [7:0] q;
    reg [7:0] q;

    always @(sig)
        q = sig << 8; //Violation
endmodule
```

Example 2

Consider the following example:

```
reg [7:0] q, q2, sig;
q = sig << 8;           //Violation
q2 = sig >> 8;          //Violation
q2[2] = sig >> 8;      //Violation
```

```

q2 = sig >> 7;           //No violation
q2 = sig[2:0] >> 7;      //Violation
q2 = sig[2:0] << 8;      //Violation
q2 = sig[2:0] << 6;      //No violation
q2 = sig[2:0] << 5;      //No violation
q2[3] = sig[3] << 1;    //Violation

```

Example 3

Consider the following example:

```

reg [3:0]out;
wire [3:0]a, b, c;
out = a + b - (c[3:0] << 4);    //Violation

```

Example 4

Consider the following example where the tag does not report a violation because the LHS of shift is non-static:

```

reg [3:0]out;
wire [3:0]a, b, c;
out = c << a; //No violation

```

Example 5

Consider the following example:

```

reg result = (4'b1111 << 4);    //Violation

```

Example 6

Consider the following example:

```

if ( ( 1'b1 << 1 ) > 1'b1 ) //Violation, all bits shifted out
    q <= 1;
if ( ( 1'b1 << 1 ) > 2'b01 ) //No warning, all bits shifted out but within maximum
    q <= 0;

```

Example 7

Consider the following example:

```

wire [4:0] a1;
assign a1 = ( (1'b1 << 4) | 2'b11 ) | 4'b0000;    //No violation
assign a1 = ( (1'b1 << 5) | 2'b11 ) | 4'b0000;    //Violation

```

Example 8

Consider the following example:

```
wire [3:0] a2;
assign a2 = ( (1'b1 << 4) | 2'b11 ) >> 1; //Violation
```

Example 9

Consider the following example:

```
wire [3:0] a3;
assign a3 = (( (1'b1 << 4) | 2'b11) | 5'b00011) >> 1; //No violation
assign a3 = (( (1'b1 << 5) | 2'b11) | 5'b00011) >> 1; //Violation
```

Example 10

Consider the following example where shifting of signed numbers is also considered:

```
if ( ( 1'sb1 <<< 1 ) > 1'sb1 ) //Violation
    q <= 1;
if ( ( 1'sb1 <<< 1 ) > 2'sb01 ) //No violation
    q <= 0;
```

Example 11

Consider the following example:

```
reg signed [3:0] start, result1, result2, result3;
reg [3:0] result4;
start = 4'b1111;
result1 = (start <<< 4); //Violation
result2 = (start <<< 3); //No violation, all bits not shifted out
result3 = (start <<< 5); //Violation
result4 = (start >>> 4); //Violation
```

Example 12

Consider the following example where the tag reports a violation because the maximum width of reduction operator will be 1:

```
wire [3:0] a4;
assign a4 = (| (1'b1 << 3)) & 4'b1111; //Violation
```

Example 13

Consider the following example where the tag reports a violation:

```
wire [2:0] a63;
assign a63 = 4'b1111 ? (1'b1 << 3 ) : 3'b000; //Violation
```

MoreLint Tags

Default Severity Label

Warning

LINT_MACRO_MODULE_USED

Macromodules are not synthesizable

Description

The *LINT_MACRO_MODULE_USED* tag reports a violation if a macromodule is used in a design because macromodules are not synthesizable.

Language

Verilog

Message Details

The following message appears when a macromodule is found in a design:

[WARNING] Macromodule '<macromodule-name>' is un-synthesizable

Example and /or Schematic

Consider the following example where the tag reports a violation because a macromodule, *test*, is used:

```
macromodule test(in1,out1);  
input in1;  
output out1;  
assign out1 = in1;  
endmodule  
  
module top();  
wire a,b;  
assign b=a;  
test T1();      // Violation  
endmodule
```

Default Severity Label

Warning

LINT_DETECT_UNSYNTHESIZABLE_NET

Net type not synthesizable

Description

The *LINT_DETECT_UNSYNTHESIZABLE_NET* tag reports violations if non-synthesizable net types are used, for example, tri0, tri1, trireg, supply0, supply1, triand and trior.

Language

Verilog

Message Details

The following message appears when a function returns integers:

[WARNING] '<net-type>' net types may not be synthesizable

Example and /or Schematic

Consider the following example where the tag reports violations because non-synthesizable net types have been used:

```
module test ();  
    tri0      n1; // Violation  
    tri1      n2; // Violation  
    supply0   n3; // Violation  
    supply1   n4; // Violation  
    trireg    n5; // Violation  
    triand    n6; // Violation  
    trior     n7; // Violation  
endmodule
```

Default Severity Label

Warning

LINT_SV_STRING_USED_IN_DESIGN

Avoid using SystemVerilog string in the design

Description

The *LINT_SV_STRING_USED_IN_DESIGN* tag reports a violation if a SystemVerilog string is used in the design. If the [check_relational_operators](#) parameter is set to *yes*, tag reports violation only if a string is used in the relational operation.

The tag does not report violation for parameter declarations with strings irrespective of used or not used and string parameters operated with another string inside any if condition. However, the tag reports violation for string parameters that are operated with numerical operators and in assign statement irrespective of any condition.

The following relational operators are checked:

- Logical equality (==)
- Logical inequality (!=)
- Less than (<)
- Less than or equal to (<=)
- Greater than (>)
- Greater than or equal to (>=)

Language

Verilog

Parameters

[check_relational_operators](#): Default value is *no*. If you set to this parameter to *yes*, the *LINT_SV_STRING_USED_IN_DESIGN* tag reports violation only if a string is used in a relational operation.

Message Details

The following message appears when a SystemVerilog string is used in the design:

[WARNING] String '<string>' is used in the design

MoreLint Tags

Default Severity Label

Warning

LINT_SNPS_DIRECTIVE_USED

Synopsys directive `translate_off/translate_on` used

Description

The *LINT_SNPS_DIRECTIVE_USED* tag reports violation on the use of `translate_off/translate_on` nested constructs. The use of `translate_off/translate_on` may cause mismatch between pre-synthesis and post-synthesis.

Language

Verilog

Message Details

The following message appears if `translate_off/translate_on` nested constructs are used:

[WARNING] Statement or expression used in `translate_off/translate_on` may cause mismatch between pre-synthesis and post-synthesis

Example and /or Schematic

Example 1

Consider the following example where the tag reports a violation because nested constructs are used:

```
module TOP(input in, clk, a, output q);
    reg q;
    wire w1, w2;
    //synopsys translate_off // Violation
    assign w1 = in;
    assign w2 = w1;
    //synopsys translate_on
    always @(posedge clk)
        q <= in;
endmodule
```

Example 2

MoreLint Tags

Consider the following example where the tag reports violations because nested constructs are used:

```
module TOP(input in);  
wire w1, w2;  
//synopsys translate_off //Violation, nested construct  
  //synopsys translate_off //Violation, nested construct  
    assign w1 = in;  
    assign w2 = w1;  
  //synopsys translate_on  
//synopsys translate_on  
endmodule
```

Default Severity Label

Warning

LINT_FSM_WITHOUT_EXIT_STATE

Do not have states without exit in FSMs

Description

The *LINT_FSM_WITHOUT_EXIT_STATE* tag reports a violation if there are states without `exit` statement in an FSM.

States without the `exit` statement are the states that do not have a transition to another state. Therefore, when control reaches this state, it gets trapped there and there are no inputs that could force it out of that state. States without `exit` could be single or a set of states together. All such states are reported by this tag.

Language

Verilog

Parameters

states_to_check: Default value is `CHECK_ALL` and the tag reports a violation if the FSM is stuck in any states. If you set this parameter to `IGNORE_UNREACHABLE_STATE`, the tag does not report any violation when the FSM is stuck in any unreachable states.

Message Details

The following message appears when a state is found to be without `exit` statement in an FSM:

[WARNING] State '<state-name>' is without exit in FSM

Suggested Fix

Reset the FSM again to make the FSM functional.

Example and /or Schematic

Consider the following example where the tag reports a violation because the `ST_S3` state does not have the `exit` statement in the FSM:

```
module FSM_693 (clk, rst_n, enable);
```

MoreLint Tags

```
input clk;
input rst_n;
input enable;
parameter [2:0]
ST_IN = 3'b000,
ST_S1 = 3'b001,
ST_S2 = 3'b010,
ST_S3 = 3'b011;

reg [2:0] rw_cs;
reg [2:0] rw_ns;
reg control;
reg y;

always @ (rw_cs or control)
begin
case (rw_cs)
ST_IN:
begin
if (control == 1'b1)
rw_ns = ST_S1;
else
rw_ns = ST_S3;
end
ST_S1:
begin
rw_ns = ST_S2;
end
ST_S2:
begin
rw_ns = ST_IN;
end
ST_S3: //violation
begin
y = 1;
end
endcase
end
```

```
always @ (negedge rst_n or posedge clk)
begin
  if (!rst_n)
    rw_cs <= ST_IN;
  else
    rw_cs <= rw_ns;
  end

endmodule
```

Default Severity Label

Warning

LINT_FSM_UNREACHABLE_STATES

Do not have unreachable states in FSMs

Description

The *LINT_FSM_UNREACHABLE_STATES* tag reports a violation if there are unreachable states in FSM.

An FSM description may contain states that are neither the initial states (starting states for the FSM) nor reachable through any other states. Such states are called unreachable states. A group of states can also be unreachable, when a state is reachable through some other state that in turn is unreachable. The tag reports such states.

Language

Verilog

Parameters

check_default_value: Default value is `IGNORE_DEFAULT` and the tag ignores the default state. If you set this parameter to `CHECK_DEFAULT`, the tag checks and reports the default state, which cannot be reached from the initial state.

Message Details

The following message appears when a state is unreachable in the FSM:

[WARNING] State '<state-name>' is unreachable

Suggested Fix

Reset the FSM again to make the FSM functional.

Example and /or Schematic

Consider the following example where the tag reports violations because the `ST_S4` and `ST_S5` states are unreachable:

```
module FSM_695 (clk, rst_n, enable, y);  
  input clk;  
  input rst_n;  
  input enable;
```

```
output [1:0] y;
parameter [2:0]
ST_IN = 3'b000,
ST_S1 = 3'b001,
ST_S2 = 3'b010,
ST_S3 = 3'b011,
ST_S4 = 3'b100,
ST_S5 = 3'b101;
reg [2:0] rw_cs;
reg [2:0] rw_ns;
reg [1:0] y;
reg control;
always @ (rw_cs or control)
begin
case (rw_cs)
ST_IN:
begin
y = 2'b00;
if (control == 1'b1)
rw_ns = ST_S1;
else
rw_ns = ST_IN;
end
ST_S1:
begin
y = 2'b01;
rw_ns = ST_S2;
end
ST_S2:
begin
y = 2'b10;
rw_ns = ST_S3;
end
ST_S3:
begin
y = 2'b11;
rw_ns= ST_IN;
end
```

MoreLint Tags

```
ST_S4:
begin
y = 2'b10;
rw_ns= ST_S5;
end
ST_S5: //violation
begin
rw_ns = ST_IN;
end
default:
begin
y = 2'b00;
rw_ns = ST_IN;
end
endcase
end
always @ (negedge rst_n or posedge clk) // sequential logic
begin
if (!rst_n)
rw_cs <= ST_IN;
else
rw_cs <= rw_ns;
end
endmodule
```

Default Severity Label

Warning

LINT_FSM_WITHOUT_INITIAL_STATE

Do not have FSM without the initial state

Description

The *LINT_FSM_WITHOUT_INITIAL_STATE* tag reports a violation if there are any FSMs without an initial state. If an FSM hasn't the capability to reset its state a constant value, this tag is violated.

The tag checks the presence of default case statement and reset in the sequential block.

Language

Verilog

Message Details

The following message appears when an FSM is found without an internal state:

[WARNING] FSM without an initial state is detected

Example and /or Schematic

Example 1

Consider the following example where the tag reports a violation because FSM cannot attain a constant initial state from the case-block:

```
module FSM_697 (clk, rst_n, enable);
  input clk;
  input rst_n;
  input enable;
  parameter [2:0] // symbolic state_vector
  ST_IN = 3'b000,
  ST_S1 = 3'b001,
  ST_S2 = 3'b010,
  ST_S3 = 3'b011;
  reg [2:0] rw_cs; // current state
  reg [2:0] rw_ns; // next state
  reg control;
```

MoreLint Tags

```
reg y;
always @ (rw_cs or control) // combinational logic
begin
  case (rw_cs)
  ST_IN:
  begin
    if (control == 1'b1)
      rw_ns = ST_S1;
    else
      rw_ns = ST_S3;
    end
  ST_S1:
  begin
    rw_ns = ST_S2;
    end
  ST_S2:
  begin
    rw_ns = ST_S3;
    end
  ST_S3:
  begin
    rw_ns = ST_IN;
    y = 1;
    end
  default:
  begin
    //rw_ns = ST_IN; //violation
    end
  endcase
  end
always @ (posedge clk) // sequential logic
begin
  rw_cs <= rw_ns; //violation
  end
always @ (negedge rst_n or posedge clk)
begin
  if (!rst_n)
    control <= 1'b0;
```

```
else
control <= 1'b1;
end

endmodule
```

Example 2

Consider the following example where the tag report does not report violation because the FSM has no initial state, but it attains the initial state from the default node:

```
module FSM_697 (clk, rst_n, enable);
input clk;
input rst_n;
input enable;
parameter [2:0] // symbolic state_vector
ST_IN = 3'b000,
ST_S1 = 3'b001,
ST_S2 = 3'b010,
ST_S3 = 3'b011;
reg [2:0] rw_cs; // current state
reg [2:0] rw_ns; // next state
reg control;

reg y;
always @ (rw_cs or control) // combinational logic
begin
case (rw_cs)
ST_IN:
begin
if (control == 1'b1)
rw_ns = ST_S1;
else
rw_ns = ST_S3;
end
ST_S1:
begin
rw_ns = ST_S2;
end
end
end
```

MoreLint Tags

```
ST_S2:
begin
rw_ns = ST_S3;
end
ST_S3:
begin
rw_ns = ST_IN;
y = 1;
end
default:
begin
rw_ns = ST_IN;
end
endcase
end
always @ (posedge clk) // sequential logic
begin
rw_cs <= rw_ns;
end
always @ (negedge rst_n or posedge clk)
begin
if (!rst_n)
control <= 1'b0;
else
control <= 1'b1;
end
endmodule
```

Default Severity Label

Warning

LINT_GTECH_CELL_USED

GTECH cell is used

Description

The *LINT_GTECH_CELL_USED* tag reports a violation if a GTECH_* cell is used in the design.

Language

Verilog

Message Details

The following message appears when a GTECH_* cell is used in the design:

[WARNING] GTECH cell '<cell-name>' is used

Example and/or Schematic

Consider the following example where the tag reports a violation because GTECH_* cell is used in the design:

```
module TOP(input in, in1, output o1, o2);  
    GTECH_AND U1 (); //Violation, GTECH_ cell used  
    GTECH_OR U2 (); //Violation, GTECH_ cell used  
endmodule
```

Default Severity Label

Warning

LINT_SUB_MODULE_NAMING_STYLE

Checks the naming convention of sub-modules

Description

The *LINT_SUB_MODULE_NAMING_STYLE* tag reports a violation if the recommended naming convention for sub-modules is not followed. If a module has `_top` postfix as `<keyword>_top`, then its sub-module name should contain the word `keyword` prefix. That is, when a sub-module name has `_top` postfix, it will update `<keyword>` and should append on its further sub-modules.

Language

Verilog

Message Details

The following message appears if module does not follow the recommended naming convention:

[WARNING] Module '`<module-name>`' doesn't follow recommended naming convention

Example and /or Schematic

Consider the following example where the tag reports violations where the module naming convention is not followed, however, violation is not reported where the naming convention is applied:

```
module MODEM_top();
    MODEM_MEMORY U1();    //No violation
    MOD_MEMORY  U2();     //Violation
    MOD_MEMORY_1_1 U0();  //Violation
endmodule

module MODEM_MEMORY(input in, output out);
    assign out = in;
endmodule
```

```
module MOD_MEMORY(input in, output out);  
    MOD_MEMORY_1 U3(); //Violation  
endmodule  
  
module MOD_MEMORY_1(input in, output out);  
    MOD_MEMORY_1_top U4(); //No violation  
endmodule  
  
module MOD_MEMORY_1_top(input in, output out);  
    MOD_MEMORY_1_1 U5(); //No violation  
    MOD_MEM U6(); //Violation  
endmodule  
  
module MOD_MEMORY_1_1(input in, output out);  
endmodule  
  
module MOD_MEM();  
endmodule
```

Default Severity Label

Warning

LINT_INVALID_NET_TYPE_IN_MODULE_INSTANTIATION

REG output port is assigned to a WIRE port connection

Description

This tag reports violation for output signals that are declared as reg and used in port connection, which are wire.

Language

Verilog

Parameters

None

Messages

Output reg net <net_name> is assigned to a wire port connection in instance <instance_name>: [Hierarchy < Hierarchy path>]

Examples

Example 1

```
module bm(in, out);
    input in;
    output out;
    assign out=in;
endmodule

module test (in1, out1);
    input in1;
    output reg out1; //Violation
    bm u_bm ( .in(in1), .out(out1));
endmodule
```

Example 2

```
module bm(in, in2, out, out2);
```

```
    input reg in2;
    output reg out2;
    input in;
    output out;
    assign out=in;
    assign out2=in2;
endmodule

module test (in1, in2, out1, out2);
    input in1;
    input in2;
    output reg out1; //Violation
    output reg out2; // No Violation
    bm u_bm ( .in(in1), .in2(in2), .out(out1), .out2(out2));
endmodule
```

Example 3

```
module bm(in, out);
    input in;
    output wor out;
    assign out=in;
endmodule

module test (in1, out1);
    input in1;
    output reg out1; // No Violation
    bm u_bm ( in1, out1);
endmodule
```

Example 4

```
module bm(in, out);
    input in;
```

MoreLint Tags

```

        output wire [3:0] out;
        assign out=in;
    endmodule
    module test (in1, out1);
        input in1;
        output reg [3:0] out1; //Violation
        bm u_bm ( in1, out1);
    endmodule

```

Example 5

```

interface mem_if (we);
    input        we;
    wire         reset;
    wire         ce;
    wire [7:0] datai;
    logic [7:0] datao;
    wire [7:0] addr;
endinterface
module test (in1, out1);
    input in1;
    output reg out1; //Violation
    reg x;
    mem_if intf(out1);
endmodule

```

Example 6

```

module bm(in, out);
    input in;
    output out;

```

```
        assign out=in;
    endmodule
    module test (in1, out1);
        input in1;
        output out1;
        assign out1=in1;
        reg x; // No Violation
        bm u_bm ( in1, x);
```

Default Severity Label

Error

LINT_UNSPECIFIED_MACRO_NAME

Macro name is unspecified in `ifdef, `ifndef, `elsif usage

Description

This tag reports violation, when any of `ifdef, `ifndef, `elsif compiler directives are used without specifying a macro name. These compiler directives are used to check whether a specified macro is defined or not. So usage of these compiler directives, without specifying a macro name can possibly be a mistake.

Language

Verilog

Parameters

None

Message Details

The *LINT_UNSPECIFIED_MACRO_NAME* tag generates the following violation message:

Directive <`ifdef/`ifndef/`elsif> must have an identifier name after it

The violation message allows you to cross-probe to the HDL description. Click the violation message to point to the line where the violated compiler directive is defined.

Example

Example 1

```
module and_op (a, b, c);
output a;
input b, c;
`ifdef behavioral // No violation, `ifdef is used with specifying macro name
    wire a = b & c;
`else
    and a1 (a,b,c);
```

```
`endif  
endmodule
```

Example 2

```
module and_op (a, b, c);  
output a;  
input b, c;  
//violation for below `ifdef, a macro name is not  
specified  
`ifdef  
    wire a = b & c;  
`else  
    and a1 (a,b,c);  
`endif  
endmodule
```

Example 3

```
module and_op (a, b, c);  
output a;  
input b, c;  
//violation for below `ifndef, a macro name is not  
specified  
`ifndef  
    wire assign a = b & c;  
else  
    and a1 (a,b,c);  
endif  
endmodule
```

Example 4

```
module and_op (a, b, c);
```


MoreLint Tags

```
output a;  
input b, c;  
  `ifdef behavioral  
wire a = b & c;  
  //violation for below `elsif, a macro name is not  
  specified  
  `elsif  
    and a1 (a,b,c);  
  `endif  
endmodule
```

Default Severity Label

Error

LINT_UNDEFINED_MACRO_USED

Macro is defined after it is used

Description

A macro should always be defined before they are used in `ifdef`. Sometimes unintentionally macros get define after they are used, which is a wrong practice. This tag checks if any macro is defined after it is used in an `ifdef` check.

This tag reports violations for those macros which are used in `ifdef/ifndef` but defined in a file which does not match with the regular expression specified through the [define_file_name](#) parameter.

However, the tag does not report violation for the following coding pattern.

```
`ifdef A
`else
`define A
`endif
```

That pattern refers to use of a macro before defining, but is intentionally written, so that violations are ignored for that pattern.

Language

Verilog

Parameters

[define_file_name](#): By default, this parameter is not set. You can set this parameter to a PERL regular expression to specify the file names. This tag reports violations for those macros which are used in `ifdef/ifndef` but defined in a file which does not match with the regular expression specified through this parameter.

Message Details

Message 1

The following message appears when a macro is defined after it is used:

[WARNING] Macro '<macro-name>' is defined after it is used
(Declared in file: '<declared-file-name>' at line '<declared-

line-number>')

Message 2

The following message appears when a macro is defined in a file but the file name does not match the regular expression specified in the [define_file_name](#) parameter:

[WARNING] Macro '<macro-name>' is defined in file('<declared-file-name>' at line '<declared-line-number>'), but file name '<declared-file-name>' does not match regex '<expected-regex>'

Message 3

The following message appears when a violated macro is defined in an included file:

[WARNING] Macro '<macro-name>' is defined in file('<declared-file-name>' at line '<declared-line-number>', included in '<parent-file-name>'), but file name '<parent-file-name>' does not match regex '<expected-regex>'

Example

Consider the following example:

```
module top (clk,rst,d,q,q2) ;
    input clk,rst,d ;
    output q,q2 ;
    reg q,q2 ;
    always @(posedge clk or negedge rst)
        if (rst == 0)
            `ifdef SET //Violation, SET is defined after it is
used in ifdef
                q <= 1 ;
            `else
                q <= 0 ;
            `endif
        else
```

```
        q <= d ;
`define SET

always @(posedge clk or negedge rst)
    if (rst == 0)
        `ifdef SET //No Violation, set is already defined
            q2 <= 1 ;
        `else
            q2 <= 0 ;
        `endif
    else
        q2 <= d ;
endmodule
```

For the above example, the tag reports the following violation message because macro `SET` is not defined before it is used in `ifdef`:

Macro 'SET' is defined after it is used (Declared in file: 'top.v' at line '15')

Default Severity Label

Warning

LINT_INVALID_NET_TYPE_IN_DU_INSTANTIATION

REG output port is assigned to a WIRE port connection

Description

This tag reports violation for output signals that are declared as reg and used in port connection, which are wire.

Language

Verilog

Parameters

None

Messages

Output reg net <net_name> is assigned to a wire port connection in instance <instance_name>: [Hierarchy < Hierarchy path>]

Examples

Example 1

```
module bm(in, out);
    input in;
    output out;
    assign out=in;
endmodule

module test (in1, out1);
    input in1;
    output reg out1; //Violation
    bm u_bm ( .in(in1), .out(out1));
endmodule
```

Example 2

```
module bm(in, in2, out, out2);
```

```
    input reg in2;
    output reg out2;
    input in;
    output out;
    assign out=in;
    assign out2=in2;
endmodule

module test (in1, in2, out1, out2);
    input in1;
    input in2;
    output reg out1; //Violation
    output reg out2; // No Violation
    bm u_bm ( .in(in1), .in2(in2), .out(out1), .out2(out2));
endmodule
```

Example 3

```
module bm(in, out);
    input in;
    output wor out;
    assign out=in;
endmodule

module test (in1, out1);
    input in1;
    output reg out1; // No Violation
    bm u_bm ( in1, out1);
endmodule
```

Example 4

```
module bm(in, out);
    input in;
```

MoreLint Tags

```

        output wire [3:0] out;
        assign out=in;
    endmodule
    module test (in1, out1);
        input in1;
        output reg [3:0] out1; //Violation
        bm u_bm ( in1, out1);
    endmodule

```

Example 5

```

interface mem_if (we);
    input        we;
    wire         reset;
    wire         ce;
    wire [7:0] datai;
    logic [7:0] datao;
    wire [7:0] addr;
endinterface
module test (in1, out1);
    input in1;
    output reg out1; //Violation
    reg x;
    mem_if intf(out1);
endmodule

```

Example 6

```

module bm(in, out);
    input in;
    output out;

```

```
        assign out=in;
    endmodule
    module test (in1, out1);
        input in1;
        output out1;
        assign out1=in1;
        reg x; // No Violation
        bm u_bm ( in1, x);
```

Default Severity Label

Error

LINT_CLKGEN_AVOID_CONTASSIGN

Use of continuous assign statement in clock generator module

When to Use

Use this tag to check for the presence of continuous assign statement in clock generator module.

Description

This tag reports violation when a continuous assign statement is inferred in user-specified clock-generator-module.

Parameter(s)

clk_gen_du: Specify the *clock_gen_du* parameter to specify the list of clock-generator-modules.

Constraint(s)

None

Messages and Suggested Fix

[WARNING] The clock module '<MODULE_NAME>' has got continuous assignment statement inferred

Potential Issues

The violation message is reported when a continuous assignment statement is inferred in the clock generator module.

Consequences of Not Fixing

Not fixing the violation may lead to unexpected results.

How to Debug and Fix

To fix the violation, ensure that the continuous assignment statement is inferred in the clock generator module.

Example Code and/or Schematic

```
module CLKGEN (clock1, clock2, clock_sel, clk_en, reset,
               clk_out1, clk_out2);
  input clock1, clock2, clock_sel, clk_en, reset;
```

```
output clk_out1, clk_out2;  
...  
assign clk_sel_en = clock_sel && clk_en; // report  
violation if CLKGEN module in the clock module list
```

Reports and Related Files

None

Default Severity Label

Warning

LINT_CLKGEN_AVOID_SEQSTMT

Use of sequential process statement in clock generator module

When to Use

Use this tag to check for the presence of sequential process statement in clock generator module.

Description

This tag reports violation when a sequential process statement is inferred in user-specified clock-generator-module.

Parameter(s)

clk_gen_du: Specify the *clock_gen_du* parameter to specify the list of clock-generator-modules.

Constraint(s)

None

Messages and Suggested Fix

[WARNING] The clock module '<MODULE_NAME>' has got sequential process inferred

Potential Issues

The violation message is reported when a sequential process statement is inferred in the clock generator module.

Consequences of Not Fixing

Not fixing the violation may lead to unexpected results.

How to Debug and Fix

To fix the violation, ensure that the sequential process statement is not inferred in the clock generator module.

Example Code and/or Schematic

```
module cLKGEN (clock1, clock2, clock_sel, reset, clk_out1,
clk_out2);
input clock1, clock2, clock_sel, reset;
output clk_out1, clk_out2;
...
    always @(posedge clock_selout or negedge reset)//
report violation if cLKGEN module in the clock module list
```

Reports and Related Files

None

Default Severity Label

Warning

LINT_RSTGEN_AVOID_SEQSTMT

Use of sequential process statement in reset generator module

When to Use

Use this tag when a sequential process statement is inferred in the user-specified reset-generator-module.

Description

This tag will report violation when a sequential process statement is inferred in the user-specified reset-generator-module.

Parameter(s)

reset_gen_du: The *reset_gen_du* parameter is used to specify the list of reset-generator-modules.

Constraint(s)

None

Messages and Suggested Fix

The reset module '<MODULE_NAME>' has got sequential process inferred

Potential Issues

The violation message is reported when a sequential process is inferred in a reset module.

Consequences of Not Fixing

Not fixing the violation may result in unexpected results.

How to Debug and Fix

To fix the violation, ensure that a sequential process is not inferred in a reset module.

Example Code and/or Schematic

```
module resetGEN (clk_in, reset_in1, reset_in2, reset_in3,
reset_sel, reset_out);
```

```
input clk_in, reset_in1, reset_in2, reset_in3, reset_sel;  
output reset_out;  
  
...
```

```
always @(posedge clk_in or negedge reset_muxout) // report  
violation if resetGEN module in the reset module list
```

Reports and Related Files

None

Default Severity Label

Warning

LINT_NONSTANDARD_LIB_USED

Report direct use of library cell in the RTL

When to Use

Use this tag to detect library cell that is directly used in the RTL.

Description

This tag reports violation when a library cell is directly used in RTL code. While checking the library cell, the tag ignores modules whose name matches with the ones specifies using the `allowed_module` list or start with "PD".

Parameter(s)

[`allow\_module\_list`](#): Use this parameter to specify the list of modules for which the direct use of library cell is allowed.

Constraint(s)

None

Messages and Suggested Fix

The instance '<INSTANCE_NAME>' should be created using standard cells not '<LIB_CELL_NAME>'

Potential Issues

The violation message is reported when a library cell is directly used in the RTL.

Consequences of Not Fixing

Not fixing the violation may result in unexpected results.

How to Debug and Fix

To fix the violation, ensure that a library cell is directly not used in the RTL.

Example Code and/or Schematic

```
module test (out, in, clock1, clock2, clk_sel, reset);
output out;
MXI2X8 clk_mux1 (.A(clock1), .B(clock2), .S0(clk_sel),
```

```
.Y(clock)); // report violation because use standard cell in
RTL code directly
CLKINX16 U_clkinv1 (.A(clock), .Y(clock_temp)); // report
violation because use standard cell in RTL code directly

gli_clkinv U_clkinv2 (.A(clock_temp), .Y(clock_out)); //No
violation here based on instantiated submodule.

endmodule

module gli_clkinv (A,Y);
input A;
output Y;
CLKINX16 U_clkinv (.A(A), .Y(Y)); //No violation here if
gli_clkinv in the allow list ( gli_.* is passed as parameter
)

endmodule
```

Reports and Related Files

None

Default Severity Label

Warning

OpenMore Tags

This section explains the OpenMore Tags in VC SpyGlass Lint.

ActLowName

Active-low signal name does not follow the naming convention

Language

Verilog, VHDL

Description

The ActLowName Tag flags those negative logic signals/active low signals that do not follow the recommended naming convention.

All scalar signals that are connected to the enable pin of a latch, set/reset pin of a flip-flop, or the enable pin of a tristate signal are considered candidate active low signals.

Then, the ActLowName Tag ignores those candidate signals that have been used in the following situations:

- In LHS of procedural assignments
- In RHS of procedural assignments
- In the RHS of a continuous assignment or a wire declaration unless it is as a simple scalar expression
- As a task call argument

Remaining candidate signals are Tag-checked.

Following the convention checked by this Tag ensures that such signals are clearly identifiable from other signals.

By default, the ActLowName Tag flags the active low signal names that do not end with either `_n` or `_b`. Use the [actlowname](#) Tag parameter to define your own regular expression pattern for naming the active low-signal.

By default, the ActLowName Tag does not report active low signals containing constants in the else part of the if-else construct. Set the [report_all_actlow](#) parameter to yes to report such cases. For example, a violation will be reported in the following case if the `report_all_actlow` parameter is set to yes:

```
if(res==1) q = din;  
else q = 1'b0;
```

By default, the ActLowName Tag does not report a violation for an active

low reset branch having non-static assignment in the RHS of an assign statement. Set the [report_nonstatic_actlow](#) parameter to yes to report violations for such cases.

For example, a violation is reported if the `report_nonstatic_actlow` parameter is set to yes in the following expression:

```
if(reset == 0)
    q = din;
```

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the location where an active low signal `<signal-name>` is declared whose name does not end with `_n` or `_b` or the regular expression specified with the [actlowname](#) Tag parameter:

Active Low signal '`<signal-name>`' does not follow naming convention

Tag Severity

Guideline

Suggested Fix

Modify all references to this name to a form that meets the convention.

Examples (Verilog)

Consider the following example that infers a MUX:

```
...
always @(test_high)
    if(test_high == 1'b0)
        b <= 1'b1;
    else
        b <= 1'b0;
```

Here, the ActLowName Tag does not consider the signal `test_high` (used as MUX select signal) as an active low signal since it is not being used as

the enable pin of a latch, set/reset pin of a flip-flop, or the enable pin of a tristate signal.

In the following example, the signal `test_high` is connected to the reset pin of flip-flop `q` and hence is flagged by the `ActLowName` Tag (assuming it is not used elsewhere in an invalid situation):

```
...
always @(posedge clock or negedge test_high)
    if(test_high == 1'b0)
        q <= 1'b0;
    else
        q <= in;
...
```

Similarly, the signal `test_high` is connected to the set pin of flip-flop `q` and hence is flagged by the `ActLowName` Tag (assuming it is not used elsewhere in an invalid situation):

```
...
always @(posedge clock or negedge test_high)
    if(test_high == 1'b1)
        q <= data;
    else
        b <= 1'b1;
...
```

In the following example, the signal `enable` is connected to the enable pin of latch `q` and hence is flagged by the `ActLowName` Tag (assuming it is not used elsewhere in an invalid situation):

```
...
if(!enable)
    q <= data; // or q <= 1'b0;
// No else part
...
```

In the following example, the signal `enable` is connected to the enable pin of tristate signal `b` and hence is flagged by the `ActLowName` Tag (assuming it is not used elsewhere in an invalid situation):

```
...
always @(enable or data)
```

```

    if (!enable) b <= data;
    else b <= 1'bZ;
...

```

In the following example, the ActLowName Tag does not flag signal `test_high` as it is used in the LHS of a procedural assignment statement:

```

...
always @(posedge clk_sys or negedge rst_sys_n)
begin
    if(test_high == 1'b0)
        b1 <= 1'b1;
    else
        b1 <= data;
    if ( !rst_sys_n )
        begin
            b <= 1'b0 ;
            test_high <= 1'b0 ;
        end
    end
end
...

```

Also, the signal `rst_syn_n` is identified as an active low signal but is not flagged as it follows the (default) naming convention.

AlwaysComment

Always block is not commented

Language

Verilog

Description

The AlwaysComment Tag flags all always blocks that do not have a corresponding comment in the same or immediately preceding line.

A comment explains clearly the purpose of the block (combinational or sequential), thus increasing understanding for future reuse.

Related Standard

Section 5.2.5 of RMM

Parameter(s)

- [*ignore_below_line*](#): By default, the parameter is set to `no`. Set the value of the parameter to `yes` to ignore the comment following (next line) the RTL construct.
- [*check_all_always*](#): By default, the parameter is set to `no`. Set the value of the parameter to `yes` to enable the tag to check all `always` (`always_ff`, `always_comb` and `always_latch`) blocks and report violations if they do not have a corresponding comment in the same or preceding lines.
- [*ignore_empty_lines*](#): By default, the parameter is set to `-1` and the tag ignores any number of white spaces/blank lines before the construct to check the comment. Set the value of the parameter to 0 or any positive integer to restrict the number of blank lines before the construct to match the comment. That is, by setting the value to 0 or any positive integer, the tag ignores that number of empty lines as specified in the value.

Message Details

The following message appears at the location where an always block is without a corresponding comment:

[OpenMore Tags](#)

Always block is not commented

Tag Severity

Tag

Suggested Fix

Add a short comment, explaining the purpose of this block.

ConstantComment

Constant declaration does not have a comment

Language

VHDL

Description

The ConstantComment Tag flags all constant declarations that do not have a corresponding comment in the same or immediately preceding line.

By default, the ConstantComment Tag flags only 30 messages per file and 600 messages per design. Set the [archname](#) Tag parameter to `yes` or `<Tag-name>` to report all violation messages.

A comment explains clearly the purpose of the constant, thus increasing understanding for future reuse.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the location where a constant is declared without a corresponding comment:

```
Constant declaration does not have comments
```

Tag Severity

Guideline

Suggested Fix

Add a comment to each constant declaration, explaining the purpose of that constant.

ArrayIndex

Bus signals are declared with low-order bit first

When to Use

Use this Tag to detect multi-bit signals that are declared with the low-order bit first.

Description

The *ArrayIndex* Tag reports violation for the bus signals that are declared with the low-order bit first, that is, it reports those multi-bit signals that do not follow the specified bit-order specification convention.

Arithmetic operations on one-dimensional arrays work on the assumption that the arrays are defined in the [MSB:LSB] format where LSB is less than MSB. Such operations on reverse arrays require complicated conversions.

NOTE: *The ArrayIndex Tag does not require the low order bit to be 0.*

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

- *strict*: The default value of the strict parameter is no. Set the value of the parameter to yes or *ArrayIndex* to enable the Tag to check for only std_logic / std_logic_vector type signals for VHDL. Also, for Verilog, if the parameter is set to yes, the Tag does not check for integer and user-defined type signals.
- *checkLowToHigh*: The default value of the checkLowToHigh parameter is no. Set the value of the parameter to yes to specify the [LSB:MSB] format as the default ordering of bits of multi-bit signals.
- *checkalldimension*: The default value of the checkalldimension parameter is no and the *ArrayIndex* Tag checks only the vector size (right-most dimension of packed array) of a variable. For Verilog:

- ? If this parameter is set to *yes*, the *ArrayIndex* Tag checks for all the dimensions (both packed and unpacked) and reports all the violations.
- ? If the parameter is set to *packed*, the Tag checks for packed dimensions and reports violations.
- ? If the parameter is set to *unpacked*, the Tag check for unpacked dimensions and reports violations.

For VHDL, the *checkalldimension* parameter takes only two values, *yes* or *no*. If set to *yes*, the *ArrayIndex* Tag checks for both packed and unpacked. Otherwise it checks only for the vector size (right-most dimension of packed array) of a variable.

- *checkTopLevel*: The default value is *no*. Set the value of the parameter to *yes* to report violation only on module/entity boundaries.

Messages and Suggested Fix

Message 1

The following message appears at the location where a multi-bit signal *<sig-name>* does not follow the recommended order of [MSB:LSB] format:

[Tag] Use '*<MSB> <rec_dir> <LSB>*' instead of '*<LSB> <dir> <MSB>*' for multibit signal '*<sig-name>*' [Hierarchy: '*<hier-path>*']

Where,

- *<rec_dir>* is recommended direction. For Verilog, the value of the *<rec_dir>* parameter is *':'*. For VHDL, the value of the parameter is either *to* or *downto*.
- *<dir>* is the specified direction. For Verilog, the value of the *<dir>* parameter is *':'*. For VHDL, the value of the parameter is either *to* or *downto*.
- *<hier-path>* is the complete hierarchical path of the signal.

Potential Issues

For more information on potential issues, click [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

The following message appears at the location where a multi-bit signal `<sig-name>` does not follow the recommended order of [MSB:LSB] format and the following **constants**/parameters/generics are used to declare MSB and LSB:

[Tag] Use (`<MSB> <rec_dir> <LSB>`) (Value: (`<msb_value> <rec_dir> <lsb_value>`)) instead of (`<LSB> <dir> <MSB>`) for multibit signal '`<sig-name>`' [Hierarchy: '`<hier-path>`']

Where,

- `<msb_value>` and `<lsb_value>` are the values of the specified msb and lsb.
- `<dir>` is the specified direction. For Verilog, the value of the `<dir>` parameter is `'<dir>'`. For VHDL, the value of the parameter is either `to` or `downto`.
- `<rec_dir>` is recommended direction. For Verilog, the value of the `<rec_dir>` parameter is `'<rec_dir>'`. For VHDL, the value of the parameter is either `to` or `downto`.
- `<hier-path>` is the complete hierarchical path of the signal.

Potential Issues

The violation message is reported when the multi-bit signal does not follow the recommended order of [MSB:LSB] format and the LSB is declared before MSB.

Consequences of Not Fixing

Not using a consistent convention increases the possibility of connecting buses in reverse order.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the instance where the offending signal is declared whose bit specification does

not follow convention.

NOTE: *By default this Tag reports violation if low-order bit is declared first. If checkLowToHigh parameter is set, then Tag will flag violation for signals with high-order bit declared first.*

To fix the violation message, define all buses with a consistent convention.

It is recommended to define the high-order bit first as this reduces the possibility of confusion and unintentionally reversed connections.

Example Code and/or Schematic

Example 1

Consider the following example for Verilog:

```
module ArrayIndex_mod1(out, in1, in2, en1, en2);
  input [3:0] in1, in2;
  input [2:0] en1, en2;
  output [3:0] out;
  reg [3:0] out;
  always @ (en1 or en2 or in1 or in2)
  begin
    if (en1)
      out = func_call(en1, en2);
    else
      task_call(out, in1, in2);
    end
function func_call;
  input i1, i2;
  reg [0:3] sig_func;
  begin
    if (&(i1 & i2) == 1'b1)
      func_call = sig_func;
  else;
  end
endfunction
```

In the above example, the *ArrayIndex* Tag reports a violation as 0:x is used instead of x:0 for multibit signal, sig_func.

Example 2

Consider the following example for VHDL:

```
library ieee;
use ieee.std_logic_1164.all

entity char_type1 is
    port (in1: bit_vector(0 to 3);
          in2: bit_vector(3 downto 0);
          in3: bit_vector(0 to 3);
          out1: out bit_vector(0 to 3)
        );
    signal signal_a : bit_vector(0 to 3);
    procedure proc
    (variable data_v: bit;
     signal s_data: bit_vector(0 to 3)) is
    begin
    end proc;
end char_type1;
```

In the above example, the *ArrayIndex* Tag reports a violation as bus signals, in1 and in3, are declared in the [LSB:MSB] format.

Example 3

The following table describes the default Tag behavior with the help of some examples. It also describes the Tag behavior with respect to the [checkalldimension](#) parameter.

Example	Packed Dimensions	Unpacked Dimensions	Violation (Default)	Violation (checkalldimension=yes)	Violation (checkalldimension=unpacked)	Violation (checkalldimension=packed)
[0:1][0:2] a [0:3][0:4]	[0:1][0:2]	[0:3][0:4]	Yes ([0:2])	Yes([0:3])	Yes([0:3])	Yes([0:1])
[1:0] a [2:0][0:3]	[1:0]	[2:0][0:3]	No	Yes([0:3])	Yes([0:3])	No

Example	Packed Dimensions	Unpacked Dimensions	Violation (Default)	Violation (checkalldimension=yes)	Violation (checkalldimension=unpacked)	Violation (checkalldimension=packed)
[0:2][1:0] a [2:0]	[0:2][1:0]	[2:0]	No	Yes([0:2])	No	Yes([0:2])
[0:2][0:1] a [2:0]	[0:2][0:1]	[2:0]	Yes ([0:1])	Yes ([0:2])	No	Yes([0:2])

Example 4

Consider following VHDL examples:

```
library ieee;
use ieee.std_logic_1164.all

type VHDLOUT_TYPE_1 is array (1 downto 0) of std_logic; -- 1D
type VHDLOUT_TYPE_2 is array (0 to 7) of VHDLOUT_TYPE_2;--2D
type VHDLOUT_TYPE_3 is array (0 to 1) of
std_logic ;--1D
type VHDLOUT_TYPE_4 is array (0 to 3) of VHDLOUT_TYPE_3;--2D
```

The following table describes the default Tag behavior and using the [checkalldimension](#) parameter for the above examples:

Example	Violation (Default)	Violation (checkalldimension=yes)
signal a: VHDLOUT_TYPE_1	No	No
signal a: VHDLOUT_TYPE_2	No	Yes (0 to 7)
signal a: VHDLOUT_TYPE_3	Yes(0 to 1)	Yes(0 to 1)
signal a: VHDLOUT_TYPE_4	Yes(0 to 1)	Yes(0 to 3)

[OpenMore Tags](#)

Default Severity Label

Tag

Tag Group

General Naming Conventions

Reports and Related Files

No related reports or files.

Related Standard

Section 5.2.1 of RMM

CaseOverIf

Use case statements instead of if/else, where feasible, if performance is important. (Verilog)
Use case statements instead of if/else, where feasible, if performance is important. (VHDL)

Language

Verilog, VHDL

Description

Verilog

The CaseOverIf Tag flags `if-else` constructs in the design that should be changed to `case` constructs to improve performance.

If the condition expression of an `if-else` construct tests the same signal more than n times, the CaseOverIf Tag assumes the nesting to be done n times. A signal is assumed tested if the signal and only that signal appears on the LHS of a comparison operation (operation using the `==`, `<=`, `>=`, and `!=` operators).

The CaseOverIf Tag flags an `if-else` construct only if there are more than 3 branches of the `if-else` construct and the same signal is being tested in each branch.

A chain of `if-else` statements on the same test signal is inferred as a priority encoder. The same logic build using `case` statement is inferred as a multiplexer, which is likely to be faster than a priority encoder.

Using a `case` construct instead of `if-else` construct only meaningful only if the same value is being tested in each branch of the `if-else` construct.

VHDL

The CaseOverIf Tag flags `IF-THEN-ELSE` constructs in the design that should be changed to `CASE` constructs to improve performance.

A chain of `IF-THEN-ELSE` statements on the same test signal is inferred as a priority encoder. The same logic build using `CASE` statement is inferred as a multiplexer, which is likely to be faster than a priority encoder.

Using a `CASE` construct instead of `IF-THEN-ELSE` construct only meaningful

[OpenMore Tags](#)

only if the same value is being tested in each branch of the IF-THEN-ELSE construct.

The CaseOverIf Tag flags an IF-THEN-ELSE construct only if there are 4 or more branches of the IF-THEN-ELSE construct and the same value is being tested in each branch.

Set the [limit_if_branch](#) parameter to a positive integer value (VHDL and Verilog) to enable this Tag to report violations when `if` branches are more than the specified value.

Related Standard

Section 5.5.8 of RMM

Message Details

Verilog

The following message appears at the first line of an `if-else` construct that has more than 3 branches and the same value is being tested in the all branches:

Case statement is recommended over deeply nested 'if' statement

VHDL

The following message appears at the first line of an IF-THEN-ELSE construct for condition clause `<sig-name>` if it has `<num>` branches and `<num>` is more than or equal to 4:

Case Statement preferred instead of if-then-else. Condition Clause variable '`<sig-name>`' is being used more than `<num>` times

Tag Severity

Guideline

Examples (Verilog)

Consider the following example where IF-THEN-ELSE construct has 4 branches and case construct is suggested:

```
module test(in1, in2, in3, sel, out);
    input in1, in2, in3;
```

```

input [3:0] sel;
output out;

reg out;

always @(in1 or in2 or in3 or sel)
begin
    if (sel == 4'h0 )
        out = in1;
    else if (sel == 4'h1 )
        out = in2;
    else if (sel == 4'h2 )
        out = in3;
    else if (sel == 4'h3 )
        out = in2 + in3;
    else
        out = in1 + in2 + in3;
end
endmodule

```

Examples (VHDL)

Consider the following example IF-THEN-ELSE construct that has 4 branches and the same value (x) is being tested in each branch:

```

library ieee;
use ieee.std_logic_1164.all

if ( x = "000" ) then y <= a and b;
elsif ( x = "001" ) then y <= a or b;
elsif ( x = "010" ) then y <= a nand b ;
elsif ( x = "011" ) then y <= a xor b;
else y <= '0';
end if;

```

For this example, VC SpyGlass Lint generates the following message:

Case Statement preferred instead of if-then-else. Condition Clause variable 'x' is being used more than 4 times

To fix this message, you can rewrite the above example using the CASE

[OpenMore Tags](#)

construct as follows:

```
case (x) is
  when "000"    => y <= a and b;
  when "001"    => y <= a or b;
  when "010"    => y <= a nand b;
  when "011"    => y <= a xor b;
  when others => y <= '0';
end case;
```

Suggested Fix

Generally it is best to recode deeply nested `if`'s as case statements unless you actually need a priority encoder or the logic is not in a performance-critical part of the circuit.

ConsCase

Name not used consistently with same case

Language

Verilog, VHDL

Description

The ConsCase Tag flags names that are case variants of a name already encountered in the same scope.

The ConsCase Tag checks for name case variants within a specific scope, that is, the boundaries of a module, a UDP, a function, a task, or a named block. If a scope is contained in another scope (say, a named always block in a module), the names in the containing scope (module) are also checked against the names in the contained scope (named always block).

Names are not checked across parallel scopes.

As Verilog is case-sensitive, it is legal to have two different constructs with the same name but with a different case (for example, macro `FOO` and signal `f00`). However, such names hinder possible future translation of the design to VHDL as VHDL is case-insensitive and thus considers objects named `FOO` and `f00` as the same object.

Also, overuse of case variants can cause significant readability problems and can increase the possibility of outright error.

NOTE: *This Tag does not flag function/procedure arguments if any case-inconsistency has occurred.*

Related Standard

Section 5.2.1 of RMM

Parameters

waiver_compat: Default value is no. If you set the value of this parameter to yes or <Tag-name>, it ensures that the Tag does not generate the line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.

Message Details

Verilog

The following message appears at the location where a name *<name-case2>* is encountered that is a case variant of the name *<name-case1>* already found in same scope and in same or different design source file *<file-name>* at line *<num>*:

Name '*<name_case2>*' used with different case at different places (prior usage '*<name_case1>*' at file: *<file-name>*, line: *<num>*)

VHDL

The following message appears at the location where a name *<name-case2>* is encountered that is a case variant of the name *<name-case1>* already found in same scope and in same or different design source file *<file-name>* at line *<num>*:

Name '*<name_case2>*' [previously '*<name_case1>*' (*<file-name>* line#*<num>*)] not used with consistent case

Tag Severity

Guideline

Suggested Fix

Fix reported names so they are different, independent of case.

Examples (Verilog)

Consider the following example:

```
module ConsCase(a_in, B_in, rst, clk, q_out);
    input rst, clk;
    input a_in, B_in ;
    output q_out;
    reg q_out;
    reg Q_bar;

    always@(posedge clk)
        begin
```

```
        if(rst)
            begin
                q_out = 1'b0;
                Q_bar = 1'b1;
            end
        else
            q_out = a_in | B_in ;
        end

        always@(q_out)
            begin :q_bar
                Q_bar = !q_out;
            end
    endmodule
```

Examples (VHDL)

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

entity andgate is
    port(a,b : in bit; c :out bit);
end andgate;

architecture arc of andgate is
begin
    process(a,b)
    begin
        c <= a and b;
    end process;
end arc;

entity top1 is
    port(a,b: in bit; out1:out bit);
end top1;

architecture arc of top1 is
```

[OpenMore Tags](#)

```
component andgate
  port(a,b : in bit; c :out bit);
end component;

begin
  INST1: andgate port map(a=>a,b=>b,c=>out1);
end arc;

configuration conf of top1 is
  for arc
    for INST1:andgate
      use entity work.ANDGATE(arc);
    end for;
  end for;
end configuration conf;
```

ConstName

Constant does not follow recommended naming convention

Language

Verilog, VHDL

Description

The ConstName Tag flags the constant names (macros and parameters) that do not follow the specified constant naming convention.

By default, the ConstName Tag flags the following types of constant names:

- Names that start with a non-alphabetical character
- Names that contain a lowercase alphabet (naming convention requires only uppercase alphabets)
- Names that contain characters other than alphanumeric characters and underscores
- Names that contain more than 8 characters (reduced readability)
- Names that contain two successive underscores (unsupported in some hardware accelerators and emulators)

Use the [constname](#) Tag parameter to define your own regular expression pattern for naming the constants.

Following the convention checked by the ConstName Tag ensures your design will encounter a minimum of macro- and parameter-naming problems across a wide range of EDA tools.

By default, the [report_localparam](#) parameter is set to `no`, and this Tag does not report violations for local parameters. Set the `report_localparam` parameter to `yes` to enable the Tag to report violation for local parameters, which do not follow the recommended naming convention.

By default, the [print_recommended_regexp](#) parameter is set to `no`. If you set this parameter to `yes` or comma-separated Tag list, the violation messages of the Tag include the recommended regular expression pattern that is used.

By default, the [check_globals](#) parameter is set to `no`. Set the value of the

parameter to `yes` to enable the tag to report a violation for parameters defined in the global scope.

By default, the `ignore_macro_names` parameter is set to `no`. Set this parameter to `yes` or the tag name (`ConstName`) to not report any violations for macro names.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the location where a Tag-violating constant name `<const-name>` is first encountered:

Constant '`<const-name>`' does not follow recommended naming convention

If you set the `print_recommended_regexp` parameter, the following message is reported that includes the recommended regular expression pattern:

Constant '`<const-name>`' does not follow recommended naming convention. Recommended pattern is '`<regular-expression-pattern>`'

Tag Severity

Guideline

Suggested Fix

Modify all references to this name to a form that meets the convention.

Examples (Verilog)

Consider the following example:

```
module test ( clk, en, out0, out1);
  input  clk;
  input [3:0] en;
  output out0, out1;
  parameter state0 = 2'b00;
  parameter STATE__1 = 2'b01;
```

```

`define STATE1      2'b10
`define MACRO_STATE3  11
reg [1:0] temp;
assign out0 = temp[0];
assign out1 = temp[1];

always @ (posedge clk)
begin
    case (temp)
        state0:
            begin
                temp <= STATE__1;
            end
        STATE__1:
            begin
                temp <= `STATE1;
            end
        `STATE1:
            begin
                if (en) begin
                    temp <= `MACRO_STATE3;
                end
            end
        `MACRO_STATE3:
            begin
                end
            endcase
    end
endmodule

```

This example generates a message for the following constant names:

- state0: constant name is in lower case
- STATE__1: constant name contains two successive underscores
- MACRO_STATE3: constant name is longer than 8 characters

Examples (VHDL)

Consider the following example:

```

library ieee;
use ieee.std_logic_1164.all

entity char_type1 is
  port ( in1: bit_vector(3 downto 0);
         in2: bit_vector(3 downto 0);
         in3: bit_vector(3 downto 0);
         out1: out bit_vector(3 downto 0)
        );
  constant SIGNa1A : bit := '0';
  constant SIGNALB : bit := '0';
  constant signalc : bit := '0';

  procedure proc (
    variable data_v: bit;
    constant S_Data: bit ) is
  constant InProc: character := '0';
  constant INPRO: character := '0';
  begin
  end proc;
end char_type1;

architecture rtl of char_type1 is
  signal signalchecked : bit;
  constant signalcheckedA : bit := '0';
  function func (
    constant In1: character;
    constant IN11: character;
    signal in2,in3: bit_vector( 3 downto 0)) return
  bit_vector is
    constant InFunc: character := 'a';
  begin
    case in1 is
      when 'a' => return in2 or in3;
      when 'b' => return in2 and in3;
      when 'c' => return in2 nand in3;
      when 'd' => return in2 xor in3;
      when others => return in2 nor in3;
    end case;
  end func;
end architecture rtl;

```

```
        end case;
    end;

begin
    test_PROC: process(in1,IN2,in3)
        variable var : character ;
        constant var1 : character := 'b';
        procedure procl (
            constant data_v: bit;
            constant DATA_V1: bit;
            signal    s_data: bit ) is
            constant InProc: character := 'c';
        begin
            end procl;
        begin
            case in1 is
                when "0000" => var:='a';
                when "0010" => var:='b';
                when "0100" => var:='c';
                when "0110" => var:='d';
                when "1000" => var:='e';
                when "1010" => var:='f';
                when "1100" => var:='g';
                when others => var:= 'h';
            end case;
            out1<= func(VAR,VAR, IN2,in3);
        end process;
    end;
```

[OpenMore Tags](#)

DefaultState

Finite-state machine does not have a defined default state

Language

Verilog, VHDL

Description

The DefaultState Tag flags the finite-state machine constructs that do not have a defined default state or have a defined default state branch with no state variable assignment.

If a finite-state machine does not have a defined default state (default clause in case statement), initialization and behavior in unusual circumstances can be unpredictable. Even if the default behavior is controlled, the logic may be difficult to understand by other users if the default state is not explicitly defined.

Related Standard

Section 5.5.9 of RMM

Message Details

Verilog

The following message appears for an finite-state machine construct with no defined default state or with a defined default state having no state variable assignment:

State Machine should have a default state

VHDL

The following message appears for an finite-state machine construct that has no defined default state:

State Machine has no default state

The following message appears for an finite-state machine construct that has a defined default state but without state variable assignment:

State Machine has default-state with no state variable assignment

Tag Severity

Guideline

Suggested Fix

Verilog

Add a default case to the case statement, in which you should set the state to some reasonable default, such as reset state or idle state. For example:

```
...
case (state) begin
    ...
    default:
        state = IDLE;
endcase
...
```

VHDL

Add either an others clause to the case construct or add a signal assignment preceding the finite-state machine description in which you should set the state to some reasonable default, such as reset state or idle state. For example:

```
library ieee;
use ieee.std_logic_1164.all

...
case state is
    ...
    when others => state <= IDLE;
end case
...
or
...
next_state <= idle;
case state is
    ...
    when idle => next_state <= bit3;
end case
```

...

Examples (VHDL)

No Default State

Consider the following example where the case construct has no others clause:

```
library ieee;
use ieee.std_logic_1164.all

architecture rtl of test is
  <signal definitions>
begin
  fsm_seq : process (<port_list>)
    <FSM's sequential logic>
  end process fsm_seq;

  fsm_com: process (c_state, c_data, data)
  begin
    case c_state is
      when i_state => c_data <= data;
                        c_state <= state_1;
      when state_1 => c_data <= c_data or data;
                        c_state <= state_2;
      when state_2 => c_data <= c_data and data;
                        c_state <= state_3;
      when state_3 => c_data <= c_data xor data;
                        c_state <= i_state;
    end case;
  end process fsm_com;

end rtl;
```

VC SpyGlass Lint generates the following message for the case construct:

State Machine has no default state

To fix this message, define the default state as follows:

```
case c_state is
```

```

        when i_state => c_data <= data;
            c_state <= state_1;
        when state_1 => c_data <= c_data or data;
            c_state <= state_2;
        when state_2 => c_data <= c_data and data;
            c_state <= state_3;
        when state_3 => c_data <= c_data xor data;
            c_state <= i_state;
        when others => c_data <= (others => '0');
            c_state <= i_state;
    end case;

```

Default State without State Variable Assignment

Consider the following example where the default state has been defined but no state variable is assigned in the default branch:

```

architecture rtl of test is
    <signal definitions>
begin
    fsm_seq : process (<port_list>)
        <FSM's sequential logic>
    end process fsm_seq;

    fsm_com: process (c_state, c_data, data)
    begin
        case c_state is
            when i_state => c_data <= data;
                c_state <= state_1;
            when state_1 => c_data <= c_data or data;
                c_state <= state_2;
            when state_2 => c_data <= c_data and data;
                c_state <= state_3;
            when state_3 => c_data <= c_data xor data;
                c_state <= i_state;
            when others <= NULL;
        end case;
    end process fsm_com;
end rtl;

```

[OpenMore Tags](#)

VC SpyGlass Lint generates the following message for the case construct:

State Machine has default state with no state variable assignment

To fix this message, rewrite the default state as follows:

```
when others => c_data <= (others => '0');  
               c_state <= i_state;
```

ExprParen

Parenthesize complex expressions to force evaluation precedence in the order you intend.

Language

Verilog, VHDL

Description

The ExprParen Tag flags all operators that follow (right to left) another operator with the same or different precedence without parenthesis to resolve the precedence conflicts.

Relying on language definitions of operator precedence is not completely reliable if you need to support multiple analysis and synthesis platforms.

Further, it may be difficult for other users to determine the order you are depending on if these users are not as familiar with operator precedence.

The ExprParen Tag will report a message in the following cases:

- Operators of different precedence level appear in an expression. For example:

```
sum = a + b * c;
```

- Different operators from same precedence level appear in an expression. For example:

```
sum = a + b - c;
```

No message is reported if same operator is used in whole expression. All expressions will be evaluated after macro expansion, if any used in the expression.

Parameters

allowedOpPrecedence: The default value is " ". Specify a list of unary and binary operators for which the ExprParen Tag should not report a violation.

Related Standard

Section 5.6.5 of RMM

Message Details

The following message appears at the location of an expression where an operator follows (right to left) another operator with the same precedence without correct parenthesis:

Use parentheses in complex expressions <opr>

Where <opr> is the first offending operator found.

Tag Severity

Guideline

Suggested Fix

Parenthesize reported cases as necessary.

Examples (Verilog)

Consider the following example:

```
module test(in1, in2, in3, sel, out);
  input      in1, in2, in3;
  input      [3:0] sel;
  output     out;
  reg out;
  always @(in1 or in2 or in3 or sel)
  begin
    case(sel)
      4'h0 : out = in1 + in1 - in3;
      4'h1 : out = in2 * in2 - in3;
      4'h2 : out = (in3 + in1) - in2 / in3;
      4'h3 : out = (in2 - in1) + in3;
      default : out = in2;
    endcase
  end
endmodule
```

Examples (VHDL)

Consider the following example:

```

library ieee;
use ieee.std_logic_1164.all

architecture rtl of test is
  signal s1, s2, s3, s4, s5, s6, s7, s8: integer;
  signal a, b, c, d, e: integer;
begin
  s1 <= a+b+c+d+e;
  s2 <= a-b+c-d-e;
  s3 <= (a-b)+(c-d)-e;
  s4 <= ((a-b)+(c-d))-e;
  s5 <= a-(b-c)-(d+e);
  s6 <= a+b+c*d*e;
  s7 <= a+b+c*d/e;
  s8 <= a*b/c*d/e;
end rtl;

```

The expression for signal `s1` uses only one operator. While the expression has not parenthesis, there are no precedence conflicts.

The expression for signal `s2` uses two operator types with same precedence and also does not have parenthesis. SpyGlass generates two `ExprParen` messages for the plus (+) operator between `b` and `c` and the minus (-) operator between `a` and `b`.

The expression for signal `s3` uses two operator types with same precedence and also has parenthesis. However, use of the parenthesis does not resolve the conflict between $(c-d) - e$ and $(a-b) + (c+d)$. SpyGlass generates an `ExprParen` message for the plus (+) operator between $(a-b)$ and $(c+d)$.

The expressions for signal `s4` and `s5` use two operator types with same precedence and also have correct parenthesis to resolve precedence conflicts.

The expression for signal `s6` uses two operator types with different precedence. While the expression has not parenthesis, there are no precedence conflicts.

The expression for signal `s6` uses three operator types with two operator types with same precedence. Also there are no parenthesis to resolve the

[OpenMore Tags](#)

precedence conflicts between $c * d$ and d / e . SpyGlass generates an ExprParen message for the multiply (*) operator between c and d .

The expression for signal `s7` uses two operator types with same precedence and also does not have parenthesis. SpyGlass generates three ExprParen messages for the multiply (*) operator between c and d , the divide (/) operator between b and c , and the multiply (*) operator between a and b .

FileHdr

Standard header not found at the top of source file

Language

Verilog, VHDL

Description

The FileHdr Tag flags lines in the file header that do not conform to the recommended structure.

Verilog

By default for Verilog designs, VC SpyGlass Lint checks for the file header in the following format:

```

////////////////////////////////////////
// Copyright (c) <year>, <company_name>
// program: <program_name>
// file: <file_name>
// Creator: <author_name>
// Date: <date>
// Description: <your_comments>
// Modification History: <your_comments>
// VERSION      Date      AUTHOR      DESCRIPTION
// <version> <date>      <name>      <comments>
// <version> <date>      <name>      <comments>
////////////////////////////////////////

```

Set the [generate_header_check_report](#) parameter to enable the *FileHdr* Tag to generate a report containing differences of actual file headers and expected file headers of all violated file headers having basic header requirements met. For this parameter to work, the [header_check_file](#) parameter should be set to an updated header check file.

Use the [header_check_file](#) Tag parameter to adjust the format of the file header to your own company standards.

By default, the *FileHdr* tag reports violation for all files, which do not have a proper file header. Use the [file_hdr_ignore_path](#) parameter to specify a list of file paths for which you want to ignore the violations

[OpenMore Tags](#)

VHDL

By default for VHDL designs, VC SpyGlass Lint checks for the file header in the following format:

```
library ieee;
use ieee.std_logic_1164.all

-----
-- Copyright (c) <c-year>, <company-name>
-- program: <program-name>
-- file: <file-name>
-- Creator: <author-name>
-- Date: <date>
-- Description: <your comments>
-- Modification History:
-- VERSION      Date      AUTHOR      DESCRIPTION
-- <version> <date>      <name>      <comments>
-- <version> <date>      <name>      <comments>
-----
```

Use the [header_check_file_vh](#) Tag parameter to adjust the format of the file header to your own company standards.

Related Standard

Section 5.2.4 of RMM

Message Details

The following message appears for each file header line that does not confirm to the default file header format:

Non-standard file header: <error-type>

Where <error-type> is the type of error in the line.

Tag Severity

Tag

Suggested Fix

Determine and follow the local approved standard.

[OpenMore Tags](#)

FuncName

Function/procedure does not follow recommended naming convention

Language

Verilog, VHDL

Description

The FuncName Tag flags the sub-program names that do not follow the specified naming convention.

By default, the FuncName Tag flags the following types of function or procedure names:

- Names that start with a non-alphabetical character
- Names that contain an uppercase alphabet (naming convention requires only lowercase alphabets)
- Names that contain characters other than alphanumeric characters and underscores
- Names that contain more than 16 characters (reduced readability)
- Names that contain two successive underscores (unsupported in some hardware accelerators and emulators)

Use the [funcname](#) Tag parameter to define your own regular expression pattern for naming the functions and procedures.

Following the convention checked by this Tag ensures your design will encounter a minimum of function/procedure-naming problems across a wide range of EDA tools.

By default, the [check_task](#) parameter is set to `no`, and the Tag does not check for task name. Set this parameter to `yes` to enable the Tag checking for task names that do not follow the naming convention.

Related Standard

Section 5.2.1 of RMM

Message Details

Verilog

The following message appears at the location where a Tag-violating function name *<func-name>* is first encountered:

Function name *<func-name>* does not follow naming convention

VHDL

The following message appears at the location where a Tag-violating function/procedure name *<name>* is first encountered:

Function/procedure name '*<name>*' does not follow the naming convention

Tag Severity

Guideline

Suggested Fix

Modify all references to this name to a form that meets the convention.

Examples (Verilog)

Consider the following example where the function name exceeds the limit of 16 characters:

```
module FuncName_mod3(in1, in2, sum);
    input [3:0] in1;
    input [3:0] in2;
    output [3:0] sum;
    reg [4:0] sum;
    reg sel;

    always @(in1 or in2)
    begin
        sum = function_adder_is(in1,in2,sel);
    end

    function [4:0] function_adder_is;
    input [3:0] first;
```

```

    input [3:0] second;
    input select;

    begin
        case(select)
            1'b1 : function_adder_is = first + second;
            1'b0 : function_adder_is = 5'b00000;
        endcase
    end
endfunction

endmodule

```

Examples (VHDL)

Consider the following example:

```

library ieee;
use ieee.std_logic_1164.all

entity FuncName_Test1_ent is
end entity;

architecture FuncName_Test1_arc of FuncName_Test1_ent
is
function funcfuncfuncfunc1(ARG1: bit) return integer;
signal sig1 : bit; -- comment
constant C1 : integer := 0;

function funcfuncfuncfunc1(ARG1: bit) return integer
is
begin
    return 1;
end funcfuncfuncfunc1;

procedure procprocprocproc1(ARG1 : bit) is
begin
end procprocprocproc1;
begin

```

```
end architecture FuncName_Test1_arc;

package FuncName_Test1_pack is
function funcfuncfuncfunc1 (ARG1: bit) return integer;
procedure procprocprocprocproc1 (ARG1 : bit);
end FuncName_Test1_pack;

package body FuncName_Test1_pack is
function funcfuncfuncfunc1 (ARG1: bit) return integer
is -- function comment
variable fvar1, fvar2 : bit;
begin
    return 1;
end funcfuncfuncfunc1;

procedure procprocprocprocproc1 (ARG1 : bit) is
variable pvar1 : bit;
begin
end procprocprocprocproc1;
end FuncName_Test1_pack;
```

[OpenMore Tags](#)

FunctionComment

Function is not commented

Language

Verilog, VHDL

Description

The FunctionComment Tag flags all function declarations that do not have a preceding comment line.

A comment explains clearly the purpose and output of the function, thus increasing understanding for future reuse.

Related Standard

Section 5.2.5 of RMM

Message Details

The following message appears at the location where a subprogram is defined without a corresponding comment in the preceding line:

```
Function does not have an introductory comment
```

Tag Severity

Tag

Suggested Fix

```
Add a comment to briefly explain the purpose of this  
function.
```

HardConst

Do not use hard-coded constants other than 0 or 1

Language

Verilog, VHDL

Description

The HardConst Tag flags use of constants other than 0 and 1.

For Verilog designs, the HardConst Tag flags all constants used in expressions except the following.

- Binary constants 1'b0 and 1'b1
- Values for 0, 1, X, Z, and ?
- Constants used as named block labels
- Constants in part-select or bit-select of a signal
- Case labels
- Vector constants having all 0 or 1 in case of binary (for example, 4'b1111 or 4'b0000). Similarly all 0 and 7 in octal based numbers and all 0 and F in case of hexadecimal based numbers.

Hard-coding constants may make it difficult to adapt a design for other applications. Use of hard-coded constants can decrease readability.

By default, the maximum number of messages reported by the HardConst Tag is restricted to 20 per design unit (Verilog) / 10 per design unit (VHDL) and 500 per design. Set the [archname](#) Tag parameter to `yes` or `<Tag-name>` to report all violation messages.

NOTE: *The HardConst Tag checking is not done for system function calls.*

Related Standard

Section 5.3.2 of RMM

Message Details

The following message appears at the location where a hard constant `<const>` other than 0 or 1 is encountered:

[OpenMore Tags](#)

Verilog

Do not use hard-coded constants <const>

VHDL

Do not use hard constant '<const>' - use named constants or generics

Tag Severity

Guideline

Suggested Fix

Declare all such constants as macros or parameters (in Verilog designs) or as named VHDL constraints or generics (in VHDL designs). This makes them easier to understand and change in a reuse application.

Examples (Verilog)

Consider the following example where hard constant 3'b101 is used:

```
module Top(clk, regA, regB, sig0, sig1);
    input clk;
    input regA, regB;
    output sig0, sig1;
    reg sig0, sig1;

    always @(clk)
    begin
        if (regA)
            sig0 = 1'b0;
        else
            sig0 = 1;
        end

    always @(clk)
    begin
        if (regB)
            sig1 = 3'b111;
        else
```

```
        sig1 = 3'b101;  
    end  
endmodule
```

Examples (VHDL)

Consider the following example that uses a hard constant 10101010:

```
library ieee;  
use ieee.std_logic_1164.all  
  
architecture rtl of test is  
    constant five: integer:= 5;  
    signal temp: string(1 to five):= "aaaaa";  
    begin  
        sig3 <= (sig1 and sig2) xor "10101010";  
    end rtl;
```

SpyGlass generates the following message:

Do not use hard constant '10101010' - use VHDL constants instead

Indent

Code is not correctly indented

Language

Verilog, VHDL

Description

The Indent Tag flags statements in the design that do not have proper indentation with respect to surrounding statements.

By default, the Indent Tag checks for indent of 4 spaces per indent level. For example, if statements in a block are to be indented by one level means they should be shifted by 4 spaces. All ports, signals, and variable declarations should be indented by one level (or 4 spaces). `Begin` should be in the same line where the construct starts or if it is on the next line then it should be indented by 4 spaces.

If a statement is broken in more than one line, then it should be indented by four spaces. This is checked in all HDL constructs.

Use the [indent_level](#) tag parameter to specify the number of spaces for each indentation level.

For exact specifications, refer to the documentation of standard Tag-primitives `velexCheckIndent` and `vhlexCheckIndent` available in the `$SPYGLASS_HOME/doc/c_primitives/libLexTags` directory.

By default, the Indent Tag flags only 20 messages per file and 500 messages per design. Set the [archname](#) Tag parameter to `yes` or `<Tag-name>` to report all violation messages.

Following this guideline enhances readability of the design.

Related Standard

Section 5.2.8 of RMM

Tag Arguments

You can specify the statement categories to be checked and corresponding indentation levels in the Perl Tagdeck file using the following syntax:

```
[ checkgivencat {<cat-id> <level-num>} ]
```

```
[ ignore {<cat-id>} ]
```

Where `checkgivencat` keyword specifies that the specified statement categories should be Tag-checked, the keyword `ignore` indicates that the specified statement categories should be ignored during Tag-checking, `<cat-id>` indicates the statement category, and `<level-num>` indicates the recommended indentation level of the statement category.

TABLE 3 Statement `cat_ids` and their purpose

Statement <code>cat_id</code>	Purpose	Default <code>level_num</code>
<code>-indent_define</code>	Check indent level of compiler directives. The statements should be at the specified indent level.	0
<code>-indent_mod_udp</code>	Checks the indent level of module and UDP statements. The statements should be at the specified indent level.	0
<code>-indent_decl</code>	Checks the indent level of parameter, port, and net declarations. The statements should be at an indent level higher than previous statement by the specified level number.	1
<code>-indent_elsif</code>	Checks the indent level of second IF statement in an <code>if-else-if</code> construct. The second <code>if</code> statement should be either in the same line as the preceding <code>else</code> statement or should be in the next line at indent level higher than preceding <code>else</code> statement by the specified indent level.	0

TABLE 3 Statement cat_ids and their purpose (Continued)

Statement cat_id	Purpose	Default level_num
-indent_table	Checks the indent level of table statement in a UDP definition. The table statement should be at indent level higher than corresponding port declarations by the specified indent level.	1
-indent_keyword {<Verilog_keyword>}	Checks the indent level of Reserved keywords always, if, repeat, forever, for, named_block, while, task, and function. All these constructs must be at the highest level in case of nesting.	1
-indent_begin	Checks the indent level of begin statements with respect to the block construct	0
-indent_case_stmt	Checks the indent level of case branch statements with respect to the case statement	1
-indent_blines	Checks the number of blank lines before and after every always, initial, task, and function blocks.	1 line
-indent_fold_line	Checks the indent level of folded lines with respect to the first (unfolded) line for all types of statements.	Same

TABLE 3 Statement cat_ids and their purpose (Continued)

Statement cat_id	Purpose	Default level_num
-indent_max_level	Specifies the maximum indent level allowed in the design source file. If indent level reaches the maximum level number, no further indentation checking will be performed unless the indent level returns to a level lower than this maximum level number. All further statements will be checked for proper indentation.	10
-indent_comment	Checks the indent level of a comment line with respect to the previous line. A comment line following another comment line should be at the same indent level.	0

Parameter(s)

[tag_arguments](#): By default, the parameter is not set. The tag uses the default configuration for indentation check. The tag_arguments parameter can specify the statement categories to be checked and the corresponding indentation levels in the parameter using the following syntax:

```
[checkgivencat {<cat-id> <level-num>}}][ ignore {<cat-id>}]
```

Message Details

The following message appears at the line *<num>* where a statement does not have the expected indentation:
Indentation mismatch, <num>

Tag Severity

Tag

Suggested Fix

Indent the code according to the above specification.

[OpenMore Tags](#)

InstName

Instance name does not follow the naming convention

Language

Verilog, VHDL

Description

The InstName Tag flags the instance names that do not follow the naming convention.

By default, following is the recommended naming convention for instance names:

- Name should start with U_
- Name length should be between 1 to 16 characters
- Name should not have two successive underscores (Use of two successive underscores may not be supported by some hardware accelerators and emulators)

Use the [instname](#) Tag parameter to define your own regular expression pattern for naming the instances.

Following the same naming convention for instance names makes it easier to identify instances and enhances readability of the design.

Set the [ignore_modules](#) parameter to a comma-separated list of PERL regular expressions specifying the names of modules for which violations should be filtered.

Set the [ignore_libcells](#) parameter to `yes` to ignore violations for lib cell instances in the module.

Related Standard

Section 5.2.15 of RMM

Message Details

The following message appears at the location where an instance `<inst-name>` is defined whose name does not start with U_:

Instance name '`<inst-name>`' does not follow naming convention

Tag Severity

Guideline

Suggested Fix

Modify all references to this name to a form that meets the convention.

Examples (Verilog)

Consider the following example where instance name U__FF1 does not follow naming convention:

```

module InstName_mod2(in1, in2, q1, q2, clk);
    input in1, in2, clk;
    output q1, q2;
    wire q1, q2;
    wire reset;

    Flop_one U__FF1
        (.clk(clk), .din(in1), .res_X(reset), .qout(q1));
    Flop_two U__FF2
        (.clk(clk), .din(in2), .res_X(reset), .qout(q2));
endmodule

```

Examples (VHDL)

Consider the following example where instance name t does not follow naming convention:

```

library ieee;
use ieee.std_logic_1164.all

architecture sim of char_type2 is
    signal signalchecked : bit_vector(3 downto 0);
    signal signalchecked1 : bit_vector(3 downto 0);
    component test1
        port (in1: bit_vector( 3 downto 0);
              in2: bit_vector( 3 downto 0);
              in3: bit_vector( 3 downto 0);
              out1: out bit_vector( 3 downto 0)

```

[OpenMore Tags](#)

```
    );  
end component;  
  
for all: test1 use entity work.char_type1(rtl);  
begin  
    t: test1 port map(in1, in2, in3, out1);  
    U_t1: test1 port map(signalchecked, signalchecked,  
signalchecked, signalchecked1);  
    tU_t1: test1 port map(signalchecked, signalchecked,  
signalchecked, signalchecked1);  
end;
```

InferFF

Flip-flop inferred

Description

The InferFF tag flags the inferred flip-flops in the design.

The InferFF tag helps you ensure that the design has been inferred correctly, that is, flip-flops have been inferred where intended. Conversely, you can ensure that flip-flops have not been inferred where you intended to infer latches.

Parameter(s)

REPORT_D_FF_COUNT: Default value is no. Set the value of the parameter to yes to report only the number of D flip-flops in the design. Set the value of the parameter to all to report violations for all inferred flip-flops as well as the count of D flip-flops.

Message Details

The following message appears at the location where a flip-flop has been inferred for the signal <sig-name>:

Flip-flop inferred for signal '<sig-name>'

Severity

Guideline

Suggested Fix

Nothing, unless you expect you design to be purely combinational or purely latch-based.

Examples (Verilog)

Consider the following example where a flip-flop for signal out is inferred:

```
module dff5 (clk, in, set, reset, out);  
input clk, in, set, reset;  
output out;  
reg out;  
always @ (posedge clk)
```


[OpenMore Tags](#)

```

case (reset)
  1'b1 : out = 1'b0;
  1'b0 : case (set)
    1'b1 : out = 1'b1;
    1'b0 : out = in;
  endcase
endmodule

```

Examples (VHDL)

Consider the following example where a flip-flop for signal q is inferred:

```

library ieee;
use ieee.std_logic_1164.all

process (rst, clk)
begin
  if (rst = '0') then
    q <= '0';
  elsif clk'event and clk = '1' then
    q <= d;
  end if;
end process;

```

For this example, VC SpyGlass Lint generates the following message:

Flip-flop inferred for signal 'q'

Since you had intended to infer a flip-flop, the message confirms that your design is correct.

InstNameLength

Instance name length does not follow the convention

Language

Verilog

Description

The InstNameLength Tag flags module instance names that are too short or too long.

By default, the InstNameLength Tag flags module instance names that are not between 4 and 8 characters. Use the [inst_min_length](#) and [inst_max_length](#) Tag parameters to specify your own limits.

Short names are easier to comprehend.

Following the same naming convention for instance names enhances readability of the design.

Related Standard

Section 5.2.15 of RMM

Message Details

The following message appears at the location where an instance `<inst-name>` is defined whose name is not between `<min-value>` and `<max-value>` characters:

Instance '`<inst-name>`' length is not between `<min-value>` and `<max-value>` characters

Where `<min-value>` and `<max-value>` is the minimum and maximum number of characters specified by the `inst_min_length` and `inst_max_length` parameters, respectively.

Tag Severity

Tag

[OpenMore Tags](#)

Suggested Fix

Modify all references to this name to a form that meets the convention.

Examples (Verilog)

Consider the following example where the instance name `FF1length` is not between 4 and 8 characters:

```
module test(in1, in2, q1, q2, clk);
    input in1, in2, clk;
    output q1, q2;

    wire q1, q2;
    wire reset;

    Flop_one FF1
        (.clk(clk), .din(in1), .res_X(reset), .qout(q1));
    Flop_one U_instname_8_chars
        (.clk(clk), .din(in1), .res_X(reset), .qout(q1));
    Flop_one U_FF1
        (.clk(clk), .din(in1), .res_X(reset), .qout(q1));
endmodule
```

InvSigType

Signal type is type bit or bit_vector

Language

VHDL

Description

The InvSigType Tag flags those signals that are of `bit` or `bit_vector` type.

Many simulators do not provide built-in arithmetic functions for `bit` and `bit_vector` types. Hence, it is recommended to use signals of type `std_logic` or `std_logic_vector` only.

Related Standard

Section 5.3.1 of RMM

Message Details

The following message appears at the location of the declaration of a signal `<signal-name>` that is of `bit` or `bit_vector` type:

Type of signal '`<signal-name>`' is `bit` or `bit_vector`. Use IEEE Standard Types

Tag Severity

Guideline

Suggested Fix

Convert bit types to standard logic types wherever possible.

Examples (VHDL)

Consider the following example where signal type is bit type:

```
library ieee;  
use ieee.std_logic_1164.all
```

[OpenMore Tags](#)

```
entity andgate is
  port ( in1, in2 : in bit; output : out bit );
end andgate;

architecture behav of andgate is
  signal x : bit; --signal type is bit
begin
  x <= in1 or in2;
  output <= in1 and in2 and x;
end behav;
```

LineLength

Line-length exceeds defined limit

Language

Verilog, VHDL

Description

The LineLength Tag flags all lines that are longer than the specified size.

Longer lines can be difficult to read, especially on smaller terminals.

By default, the maximum length of any line allowed is 132. Use the [line_max_length](#) Tag parameter to change the maximum allowed length of the line.

Use the [comment_max_length](#) Tag parameter to change the maximum allowed length of the pure comment line.

Use the [comment_acc_max_length](#) parameter to specify the maximum value for the accumulated comment length in a line where both RTL and comments are present.

Use the [rtl_acc_max_length](#) parameter to specify the maximum value for the accumulated RTL length in a line where both RTL and comments are present.

Use the [comment_string](#) tag parameter to check the length of pure comment line containing a particular string, against the value specified in the [comment_max_length](#) Tag parameter.

Related Standard

Section 5.2.7 of RMM

Message Details

The following message appears at the location of line `<line-num>` that is longer than the specified size `<size>`:

Line length `<line-num>` is more than `<size>` characters

Tag Severity

Guideline

[OpenMore Tags](#)

Suggested Fix

Fold longer lines so they become shorter.

NamedAssoc

Use named-association rather than positional association to connect to an instance

Language

Verilog, VHDL

Description

The NamedAssoc Tag flags instances with positional port mapping.

Using positional association significantly increases the possibility of misconnects, some of which may prove very subtle and difficult to find. Using named association completely eliminates this possibility. Also, other users do not have to cross-reference to the module definition to understand signal connections. Port mapping by position or order is difficult to maintain in case the type, number, or name of ports change.

Related Standard

Section 5.2.11 of RMM

Message Details

The following message appears at the location of an instance that uses positional port mapping:

Verilog

Recommend making instance connections by name rather than by position

VHDL

Explicit named association must be used in instance port/generic maps

Tag Severity

Tag

Suggested Fix

Rewrite all instances using named-association connections.

Examples

Consider the following example where the port connection is by port order or position:

```
clkGen cg(IN1, CLK1);
```

The same example can be rewritten for port mapping by name as follows:

```
clkGen cg(.IN(IN1), .CLK(CLK1));
```

Examples (Verilog)

Consider the following example:

```
module clkGen(in,out);
  input [5:0] in;
  output [2:0] out;
  reg    temp1,temp2;
  wire   net1;
  reg    net2,net3;
  assign out[0] = temp1;
  assign out[1] = temp2;

  always@(posedge in[5])
  begin
    temp2 = net3;
    net3  = ~net3;// Sequential
  end
  gatedClk gc(in[4:1],out[2]); // Gated Clock
endmodule
```

Examples (VHDL)

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

architecture arc_top of top is
  component mid is
    generic ( sizein : integer := 2 ;
              sizeout : integer := 4 );
```

```
end component;  
begin  
    mid_inst: mid generic map (3, 6);  
end arc_top;
```

[OpenMore Tags](#)

NoDup

Signal, variable, entity or other named object names are duplicated

Language

VHDL

Description

The NoDup Tag flags duplicate object names in the design.

By default, the NoDup Tag does not report violations for duplicate names in port and function input argument.

Set the [ignore_function_port](#) parameter to `no` to report violations for duplicate names in port and function input argument.

NOTE: Duplicate names in package body and architecture are not reported.

Following the guideline of unique names for all objects enhances readability of the design. It also removes the possibility of variable shadowing.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the location where an object is found with the same name as that of an object `<obj-name>` already found at line `<num>` in the same library:

Use unique names. Object named '`<obj-name>`' exists at line `<num>` ('`<file-name>`')

The following message appears at the location where an object is found with the same name as that of an object `<obj-name>` already found in some other library `<lib-name>`:

Use unique names. Object named '`<obj-name>`' exists in library '`<lib-name>`'

Tag Severity

Tag

Suggested Fix

Modify one or other of the reported names to avoid duplication.

Examples (VHDL)

Consider the following example where object name myname is duplicate:

```
library ieee;  
use ieee.std_logic_1164.all
```

```
ENTITY myname IS  
  PORT (  
    clk :in std_logic;  
    myname : IN std_logic;  
    sig2 : IN std_logic;  
    SIG3 : OUT std_logic);  
END myname;
```

NotInSens

A signal is read inside a combinational always block but is not included in the sensitivity list (Verilog)
A signal is read inside a combinational process block but is not included in the sensitivity list (VHDL)

Language

Verilog, VHDL

Description

The *NotInSens* Tag reports signals that are read in a combinational `always` (`process`) construct but are not present in the construct's sensitivity list.

By default, the Tag (Verilog) does not report a signal that is missing from the sensitivity list when it is set in a blocking assignment inside the `always` construct before it is read. Set the [simplsense](#) parameter to `yes` to report such signals also.

The Tag (Verilog) does not check the for-loop indices because they (typically) do not represent real signals.

By default, the Tag does not report any violation for delay variable used inside the `always` blocks, which are not read inside sensitivity list. Set the value of the [strict](#) parameter to `yes` to report such variables.

Signals missing from the sensitivity list can lead to a mismatch between pre- and post-synthesis simulation. Before synthesis, the simulation only evaluates changes in the combinational logic when the signals in the sensitivity list change. However, the synthesis process generates logic that reads all required values, whether or not they are in the sensitivity list. In effect, the missing signals are added to the sensitivity list. Therefore, the post-synthesis simulation results are different from those of the pre-synthesis simulation.

To ensure that the pre-synthesis simulation matches the post-synthesis simulation results, you must add the missing signals to the sensitivity list of the combinational `always` (`process`) construct.

By default, the Tag does not report a violation for memory accesses. Use the [strict](#) parameter to report memory accesses also.

Related Standard

Section 5.5.5 of RMM

Message Details

Verilog

The following message appears at the sensitivity list of an `always` construct where a signal or variable `<name>` is read in the construct but is not present in the construct's sensitivity list:

Not all bits of signal/variable '`<name>`', that are read in this block, are present in sensitivity list [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the hierarchical path starting at the root of the design hierarchy and descending to the object, including the names of the instantiated design entities.

VHDL

The following message appears at the sensitivity list of a `process` block where the signal `<sig-name>` of type `<sig-type>` in the instance hierarchy `<hier-path>` is read in the process but is not present in the sensitivity list of the process:

The `<sig-type>` '`<sig-name>`' (or some of its bits) read in the process is not in the process sensitivity list. [Hierarchy: '`<hier-path>`']

Where, `<sig-type>` can be `signal`, `in port`, or `inout port` and the `<hier-path>` is the complete hierarchical path of the signal.

Tag Severity

Tag

Suggested Fix

Add the reported signals to the sensitivity list.

Examples (Verilog)

Consider the following example where all the signals that are read in the

block are not present in the sensitivity list:

```
module test(in1, in2, out1);
  input in1, in2;
  output out1;

  reg out1;

  always @(in1)
    out1 = in1 & in2;
endmodule
```

Examples (VHDL)

Consider the following example where the signals `in2` and `in3` are read in the process `s_PROC` but are not present in the sensitivity list:

```
library ieee;
use ieee.std_logic_1164.all

entity test is
  port(d1, d2, d3: in std_logic; q: out std_logic);
end test;

architecture rtl of test is
begin
  s_PROC: process(d1)
  begin
    case d1 is
      when '0' => q <= d3;
      when '1' => q <= d2;
      when others => null;
    end case;
  end process s_PROC;
end rtl;
```

For this example, SpyGlass generates the following messages:

The in port 'd2' (or some of its bits) read in the process is not in the process sensitivity list [Hierarchy: ':test(rtl):s_PROC:']

The in port 'd3' (or some of its bits) read in the process is not in the process sensitivity list [Hierarchy: 'test(rtl):s_PROC:']

To fix these messages, rewrite the design to include the missing signals in the sensitivity list as follows:

```
s_PROC: process(d1, d2, d3)
```


NotReqSens

**A signal is included in the sensitivity list of a combinational always block but not all (or none) of its bits are read in that block (Verilog)
A signal is included in the sensitivity list of a process block but only some or none of its bits are read in that block (VHDL)**

Description

Verilog

The NotReqSens Tag flags signals that are in the sensitivity list of a combinational `always` construct but are not read in the construct.

While such signals do not affect the functionality, they slow down the design simulation. Also, the logic related to a signal in the sensitivity list may have been accidentally removed or not added, which results in wrong functionality.

By default, the NotReqSens Tag reports a message if at least one bit of a vector signal present in the sensitivity list of an `always` construct, is not read in the construct. However, if you set the [dcompat](#) Tag parameter, the NotReqSens Tag reports a message only when none of the bits of a vector signal present in the sensitivity list of an `always` construct, are read in the construct.

VHDL

The NotReqSens Tag flags signals that are in the sensitivity list of a process but are not completely read in the process.

In combinational processes, any signal (as a whole or some of its bits) is not read in the process but included in the sensitivity list, is flagged.

In sequential processes, signals other than clock and reset signals are not required in the sensitivity list even if they are read in that process block and hence, are flagged.

By default, the NotReqSens Tag reports a message if not all bits of a vector signal are read in the process but the whole bus is present in the sensitivity list. Set the [dcompat](#) Tag parameter to report a message only when none of the bits of a vector signal are read in the process but the whole bus is present in the sensitivity list.

While such signals do not affect the functionality, they slow down the design simulation. Also, the logic related to a signal in the sensitivity list may have been accidentally removed or not added, which results in wrong

functionality.

NOTE: For VHDL designs, the *NotReqSens* Tag does not handle records.

Related Standard

Section 5.5.5 of RMM

Message Details

Verilog

The following message appears at the location of the sensitivity list of a combinational always construct where a signal or variable `<name>` is present in the sensitivity list of the construct but is not subsequently read in the construct:

The signal/variable '`<name>`' may not be required in the sensitivity list as not all (or none) of its bits are read in always block [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the signal.

VHDL

The following message appears at the location of the sensitivity list of a combinational process where signal `<sig-name>` of type `<sig-type>` of instance hierarchy `<hier-path>` is present in the sensitivity list of the process but is not subsequently read in the process:

The `<sig-type>` '`<sig-name>`' is not required in the sensitivity list as only some or none of its bits are read in process [Hierarchy: '`<hier-path>`']

Where, `<sig-type>` can be signal, in port, or inout port and the `<hier-path>` is the hierarchical path starting at the root of the design hierarchy and descending to the object, including the names of the instantiated design entities.

The following message appears at the location of the sensitivity list of a sequential process where signal `<sig-name>` of type `<sig-type>` of instance hierarchy `<hier-path>` is present in the sensitivity list of the process but is not required:

The `<sig-type>` '`<sig-name>`' is not required in the sequential process [Hierarchy: '`<hier-path>`']

Where, *<sig-type>* can be signal, in port, or inout port and the *<hier-path>* is the hierarchical path starting at the root of the design hierarchy and descending to the object, including the names of the instantiated design entities.

Tag Severity

Guideline

Suggested Fix

In general, it is recommended that you set the `dccompat` parameter, then remove any signals reported by this Tag from the sensitivity list.

Examples (Verilog)

Consider the following example where the signal `in1` is in the sensitivity list but is not read in the `always` construct:

```
...
always @(in1 or in2)
    out1 = &in2;
...
```

Examples (VHDL)

Consider the following example where the input port `in2` is not required in the sensitivity list as it is not read in the `process` construct and the signal `in1` is also not required in the sensitivity list as only some of its bits are read in the `process` construct:

```
library ieee;
use ieee.std_logic_1164.all

entity top is
    port (in1  : in std_logic_vector (2 downto 0);
          in2  : in std_logic;
          en   : in std_logic;
          out1 : out std_logic_vector (2 downto 0);
          out2 : out std_logic);
```

```
end top;

architecture top of top is
begin
  PROC_1: process (en, in1, in2)
  begin
    if (en = '0') then
      out1(1 downto 0) <= in1 (1 downto 0);
    end if;
  end process PROC_1;
end top;
```

[OpenMore Tags](#)

RegOutName

Register output signal does not follow naming convention.

Language

Verilog, VHDL

Description

The RegOutName Tag flags those register output signal names that do not follow the recommended naming convention.

By default, the RegOutName Tag flags those register output signal names that do not end in `_r`. Use the [regoutname](#) Tag parameter to define your own regular expression pattern for naming the register output signal.

Following this Tag makes it easy to identify those signals which are directly registered and therefore have well defined timing.

Related Standard

Section 5.2.1 of RMM

Message Details

With default name check

The following message appears at the location where a Tag-violating output name `<name>` is encountered:

Register output name '`<name>`' should end with `_r`

With user-defined name check

If you have specified your own regular expression using the [regoutname](#) Tag parameter, then the following message appears at the location where a Tag-violating output name `<name>` is encountered:

Register output name '`<name>`' should end with regular expression `$regoutname`

Tag Severity

Guideline

Suggested Fix

Adjust signal names to follow the convention. Note that input and output signal names must be fixed together, otherwise this tag reports violation.

Examples (Verilog)

Consider the following example where register output name w1_out should end with _r:

```
module RegOutName(w1_in, w2_in, clk, w1_out, w2_out);
    input w1_in, w2_in, clk;
    output w1_out, w2_out;
    reg w1_out, w2_out;

    always @(posedge clk)
    begin
        w1_out <= 1;
        w2_out <= 0;
    end
    always @(negedge clk)
    begin
        w1_out <= ~w1_in;
        w2_out <= w2_in;
    end
endmodule
```

Examples (VHDL)

Consider the following example where register output name q should end with _r:

```
library ieee;
use ieee.std_logic_1164.all

entity dff is
    port (d,clk,reset : in std_logic ;
          q : out std_logic ) ;
end dff ;
```

[OpenMore Tags](#)

```
architecture behav of dff is
begin
  process(clk,reset)
begin
  if reset = '0' then q <= '0' ;
  elsif clk'event and clk = '0' then
    q <= d ;
  end if ;
end process ;
end behav;
```

NoTab

Tab detected in design

Language

Verilog, VHDL

Description

The NoTab flags tab characters found in the design file.

The NoTab Tag flags only one message per file to reduce the number of messages.

Following this guideline enhances readability of the design across different text editors and editor settings.

Related Standard

Section 5.2.8 of RMM

Message Details

The following message appears at the location where a tab character is found:

Use spaces rather than tabs for indentation

Tag Severity

Guideline

Suggested Fix

Search for and replace all tabs by spaces.

[OpenMore Tags](#)

NoDefine

Macro definitions should not appear in files containing module definitions

Language

Verilog

Description

The NoDefine Tag flags macro defined and used in the same design file.

Macros should be defined in separate files which should be included in the main design files. Following this guideline improves reuse of macro definitions and minimizes the chance of mismatches between design files.

Related Standard

Section 5.3.4 of RMM

Message Details

The following message appears at the location where a macro *<macro-name>* is used that is also defined in the same design file:

Macro '*<macro-name>*' should be defined in a separate file

Tag Severity

Guideline

Suggested Fix

Consolidate all macro definitions in a set of standard header files.

Examples

Consider the following example where macro `delay_lower` is defined and used in the same file:

```
module test(out, in, enable);  
    output out;  
    input in, enable;  
    lower lower1 (out, in, enable);
```

```
endmodule

module lower (out_lower, in_lower, en_lower);
    output out_lower;
    input in_lower, en_lower;
    `define delay_lower 20
    reg out_lower;

    always @ (en_lower or in_lower)
        out_lower = #`delay_lower in_lower;
endmodule
```

[OpenMore Tags](#)

NoGates

Directly instantiated gate detected

Language

Verilog, VHDL

Description

The NoGates Tag flags the gates that are directly instantiated in an architecture.

The NoGates Tag flags any HDL primitive or black box module.

By default, the NoGates Tag flags all directly instantiated primitives or black box modules. Use the [gates](#) Tag parameter to specify the names of primitives or black box modules whose instants are to be allowed. Then, the NoGates Tag flags instantiations of all other primitives or black box modules.

By default, the NoGates Tag reports a maximum of 10 messages per design unit and a maximum of 500 messages per design. Set the [archname](#) Tag parameter to *yes* or *<Tag-name>* to report all violation messages.

Directly instantiating gates undermine design portability.

NOTE: *The NoGates Tag does not flag DesignWare instantiations.*

Related Standard

Section 5.3.6 of RMM

Message Details

The following message appears at the location of a directly instantiated gate *<gate-name>*:

Do not instantiate gate '*<gate-name>*' directly in design

Tag Severity

Guideline

Suggested Fix

Avoid instantiating gates directly in the design. If you must use that gate,

hide it in a 'wrapper' design unit so that the technology decision is localized to one place.

Examples (Verilog)

Consider the following example where HDL primitives and black boxes are instantiated directly:

```
module top(Q, QB, signal, rst, clk);
    input signal;
    input rst, clk;
    output Q, QB;

    wire QN, QBN;

    and and1 (Q, clk, rst);
    or or1 (QB, clk, rst);
    not not1 (QN, clk);
    bufif1 (QBN, rst, signal);
    blackbox t1 (Q, QB, signal, rst, clk);
endmodule
```

All such instantiations are flagged.

Examples (VHDL)

Consider the following example where components one and two are directly instantiated in architecture rtl:

```
library ieee;
use ieee.std_logic_1164.all

entity one is
    port(in1 : integer; out1 : out integer);
end;

architecture a of one is
    begin
end;

entity two is
    port(in1 : integer; out1 : out integer);
```

```
end;
architecture a of two is
  begin
end;
entity top is
end;
architecture a of top is
  component one
    port(in1 : integer; out1 : out integer);
  end component;
  component two
    port(in1 : integer; out1 : out integer);
  end component;
  signal in1, in2, in3 : integer;
  begin
    inst_one : one
      port map(in1 => in1, out1 => in2);
    inst_two : two
      port map(in2, out1 => in3);
  end;
```

For this example, VC SpyGlass Lint generates the following messages:

Do not instantiate gate 'one' directly in design

Do not instantiate gate 'two' directly in design

NoTopGates

Gates instantiated at the top-level of the design

Language

Verilog, VHDL

Description

The NoTopGates Tag flags all gates instantiated at the top-level of the design hierarchy.

By default, the NoTopGates Tag flags all gates, black boxes, and ASIC cells directly instantiated at the top-level of the design. Use the [topgates](#) Tag parameter to specify the names of the gates, black boxes, and ASIC cells to be allowed at the top-level.

During synthesis, logic instantiated directly at the top-level of the design hierarchy cannot easily be optimized with the rest of the logic. Thus, it is recommended to include any such logic at lower levels of the design hierarchy.

NOTE: *For Verilog designs, the NoTopGates Tag does not flag DesignWare instantiations.*

By default, the NoTopGates Tag reports a maximum of 10 messages per design unit and a maximum of 500 messages per design. Set the [archname](#) Tag parameter to *yes* or *<Tag-name>* to report all violation messages.

Related Standard

Section 5.6.8 of RMM

Message Details

The following message appears at the location where a gate *<gate-name>* is instantiated directly at the top-level of the design hierarchy:

Gate '*<gate-name>*' instantiated at top of design

Tag Severity

Guideline

Suggested Fix

If the gates you are instantiating are allowed at the top-level, such as IO pads, modify the `topgates` parameter to include these cells. If the cells you are instantiating are not allowed, find other way to implement the same function.

Examples (Verilog)

Consider the following example where `and`, `or`, `not`, and `buf1` gates are instantiated at the top-level:

```
module test(Q, QB, signal, rst, clk);
    input signal;
    input rst, clk;
    output Q, QB;

    wire QN, QBN;

    and and1 (Q, clk, rst);
    or or1   (QB, clk, rst);
    not not1 (QN, clk);
    buf1 (QBN, rst, signal);
endmodule
```

Examples (VHDL)

Consider the following example where components one and two have been directly instantiated (as `inst_one` and `inst_two`) at the top-level of the design hierarchy:

```
library ieee;
use ieee.std_logic_1164.all

architecture rtl of test is
    signal s1, s2, s3: bit;
    component one
        port(in1: integer; out1: out integer);
    end component;
    component two
        port(in1: integer; out1: out integer);
```

```
end component;  
signal in1, in2, in3: integer;  
begin  
    inst_one: one  
        port map(in1 => in1, out1 => in2);  
    inst_two: two  
        port map(in2, out1 => in3);  
end rtl;
```

For this example, SpyGlass generates the following messages:

Gate 'one' instantiated at top of design

Gate 'two' instantiated at top of design

NoTopLogic

Logic inferred at the top level of the design

Language

Verilog, VHDL

Description

The *NoTopLogic* Tag reports all logic inferred at the top-level of the design hierarchy.

The Tag does not report a message if the design has no hierarchy.

For Verilog designs, the gate and UDP instances are not treated as hierarchy.

During synthesis, logic inferred directly at the top-level of the design hierarchy cannot easily be optimized with the rest of the logic. Therefore, it is recommended to include any such logic at lower levels of the design hierarchy.

NOTE: *It is recommended to use this Tag for checking a netlist design. Do not use this Tag to check an RTL design.*

The Tag does not report violations for simple assignments. For example, no violation will be reported for the following assignments:

```
assign Out = in; //no violation
assign Out = in[1]; //no violation
assign Out = in1 & in2; //violation
```

Parameter(s)

- *ignore_conv_function*: Default value is no and *NoTopLogic* does not ignore any function calls. Set the value of *ignore_conv_function* Tag parameter to yes to ignore simple IEEE Verilog and VHDL conversion functions. This parameter also accepts a comma separated list of functions. In this case, all the IEEE simple conversion functions and user-defined functions specified through this argument are ignored by the *NoTopLogic* Tag.

Related Standard

Section 5.6.8 of RMM

Message Details

The following message appears at the first line of the top-level module where logic has been inferred:

```
Logic inferred at the top level
```

Tag Severity

Guideline

Suggested Fix

Push inferred logic down inside a lower level instance or create a new instance for this logic.

Examples (Verilog)

Consider the following example where gate black box is instantiated at the top of the design:

```
module top(Q, QB, signal, rst, clk);  
    input signal;  
    input rst, clk;  
    output Q, QB;  
  
    blackbox t1 (Q, QB, signal, rst, clk);  
endmodule
```

Examples (VHDL)

Consider the following example where logic is inferred at the top-level:

```
library ieee;  
use ieee.std_logic_1164.all  
  
architecture top_arc of top is  
    component flop is  
        port(d, clk, rst: in bit; q: out bit);  
    end component;  
  
    signal q : bit;  
begin
```

[OpenMore Tags](#)

```
qb <= not q;  
INST: flop port map(d, clk, rst, q);  
end top_arc;
```

OnePortLine

No more than one port should be declared per line

Language

Verilog, VHDL

Description

The OnePortLine Tag flags multiple port declarations in one line.

Declaring ports in separate lines improves readability and provides space for individual port comments.

Related Standard

Section 5.2.10 of RMM

Message Details

The following message appears at the location where multiple ports are declared in one line:

```
Multiple ports declared in one line
```

Tag Severity

Guideline

Suggested Fix

Split multi-port declaration lines into one port per line.

Examples (Verilog)

Consider the following example where multiple ports are declared in one line:

```
module test(inp1, inp2, outp, sel);  
    input inp1, inp2, sel;  
    ...  
endmodule
```

[OpenMore Tags](#)

Example (VHDL)

Consider the following example:

```
library ieee;  
use ieee.std_logic_1164.all  
  
entity test is  
  port( in1, in2 : in integer;  
        outtt : out integer );  
end;
```

OneStmtLine

More than one statement detected on a line

Language

Verilog, VHDL

Description

The OneStmtLine Tag flags the lines that contain more than one HDL statement.

The second statement on a line can easily be overlooked and can create confusion.

By default the OneStmtLine Tag will not check for signal/variable declarations for VHDL designs and input, output, reg, wire, parameter, and integer declarations for Verilog designs. Set the value of the [strict](#) Tag parameter to `yes` or `OneStmtLine` to have the OneStmtLine Tag perform strict checking and report all cases.

You can set the value of the [ignore_stmt](#) parameter to `yes` to enable the *OneStmtLine* Tag to ignore violations for the following scenarios:

- "begin" is allowed on the line creating the block (if/always/generate block)
- "end" is allowed with "else" and "else" with "begin", in any combination
- Single-assignments are allowed with the generating block statement (if/always), but "begin <command> end" is reported as a violator
- For the struct/union/enum type declaration, the first element declaration is allowed on the line creating the struct/union/enum structure

NOTE: *The ignore_stmt parameter is applicable for Verilog only.*

Related Standard

Section 5.2.6 of RMM

Message Details

The following message appears at the location where more than one HDL statement appears on the same line:

[OpenMore Tags](#)

Use a separate line for each HDL statement

Tag Severity

Tag

Suggested Fix

Break the statements up into multiple lines, one statement per line.

PortComment

Port is not commented

Language

Verilog, VHDL

Description

The PortComment Tag flags the port declarations that do not have a comment on the same line.

A comment explains clearly the purpose of the port, thus increasing understanding for future reuse.

For Verilog, the PortComment Tag checks the ports of all the constructs like module, interface, task, and functions.

By default, the PortComment Tag flags only 30 messages per file and 600 messages per design. Set the [archname](#) Tag parameter to *yes* or *<Tag-name>* to report all violation messages.

Related Standard

Section 5.2.5 of RMM

Parameter(s)

- [ignore_below_line](#): By default, the parameter is set to *no*. Set the value of the parameter to *yes* to ignore the comment following (next line) the RTL construct.
- [ignore_empty_lines](#): By default, the parameter is set to *-1* and the tag ignores any number of white spaces/blank lines before the construct to check the comment. Set the value of the parameter to 0 or any positive integer to restrict the number of blank lines before the construct to match the comment. That is, by setting the value to 0 or any positive integer, the tag ignores that number of empty lines as specified in the value.

Message Details

The following message appears at the location where a port is declared without a corresponding comment on the same line:

[OpenMore Tags](#)

Port declaration is not commented

Tag Severity

Guideline

Suggested Fix

Add a comment to each port declaration.

PortGroups

No blank line between connections to input ports and output ports

Language

Verilog, VHDL

Description

The PortGroups Tag flags those groups of port declarations that do not have blank line(s) between input port declarations and output port declarations.

For Verilog designs, the PortGroups Tag reports a message if there is no blank or comment line between connections to ports of different I/O type in an instantiation.

For VHDL designs, the PortGroups Tag reports a message if there is no blank or comment line between input and output ports in:

- entity port declaration
- port map in instantiation
- block statement
- component port declaration

By default, SpyGlass checks for at least one blank line between port groups.

Following this guideline enhances readability of the design.

Related Standard

Section 5.2.11 of RMM

Message Details

The following message appears at the location where input port declarations and output port declarations are not separated by the specified number of lines *<num>*:

At least 1 blank line(s) required between input and output ports

[OpenMore Tags](#)

Tag Severity

Guideline

Suggested Fix

Whenever you switch port direction in an instance connection, add a blank line.

Examples (Verilog)

Consider the following example where no blank lines between the input and output port:

```

module test(in1, in2, q1, q2, clk);
    input in1, in2, clk;
    output q1, q2;
    wire q1, q2;
    wire reset;

    Flop_one FF1 (.clk(clk),
                  .din(in1),
                  .res_X(reset),
                  .qout(q1));

    Flop_one FF2
        (.clk(clk), .din(in1), .res_X(reset), .qout(q1));

    Flop_one FF3 (.clk(clk),
                  .din(in1),
                  .res_X(reset),
                  .qout(q1));

endmodule

```

Examples (VHDL)

Consider the following example:

```

library ieee;
use ieee.std_logic_1164.all

entity andgate is
    port(a,b : in bit;

```

```
        c :out bit);
end andgate;
architecture arc of andgate is
begin
    process(a,b)
    begin
        c <= a and b;
    end process;
end arc;

entity topl is
    port(a,b,c,d: in bit;
        out1,out2:out bit);
end topl;

architecture arc of topl is
    component andgate
    port(a,b : in bit;
        c :out bit);
    end component;

begin
    INST1: andgate port map(a=>a,b=>b,
                            c=>out1);
end arc;
```

PortOrder_A

Ports not declared in recommended order

Language

Verilog, VHDL

Description

Verilog

The PortOrder_A Tag flags module port declarations that are not in the recommended order.

By default, the recommended order for port declaration is as follows:

1. Input ports
2. Output ports
3. Inout ports

Within each port type, the ports should be specified in the following order:

1. Clock
2. Resets
3. Other types

Under the recommended port definition order a message is generated for the following cases:

- Every input port generated after an output or inout port.
- Every output port generated after an inout port

Within each port a message is generated for the following cases:

- Every clock type port specified after a reset type port or a port that is neither a clock or a reset
- Every reset type port specified after a port that is neither a clock nor a reset.

Use the [portorder](#) Tag parameter to define your own order of port declaration.

VHDL

The PortOrder_A Tag flags the port declarations that are not in the recommended order.

By default, the recommended order for port declaration is as follows:

1. Input ports
2. Output ports
3. Inout ports
4. Buffer ports
5. Linkage ports

Within each port type, the ports should be specified in the following order:

1. Clock
2. Reset
3. Other types

Under the recommended port definition order, a message is generated for the following cases:

- Every input port specified after an output, inout, buffer or linkage port.
- Every output port specified after an inout, buffer or linkage port
- Every inout port specified after a buffer or linkage port
- Every buffer port specified after a linkage port

Within each port type, a message is generated for the following cases:

- Every clock type port specified after a reset type port or a port that is neither a clock or a reset
- Every reset type port specified after a port that is neither a clock nor a reset

Use the [portorder](#) Tag parameter to define your own order of port declaration.

Declaring ports in a consistent order aids understanding and reduces the likelihood of incorrect connections.

Related Standard

Section 5.2.10 of RMM

Message Details

The following message appears at the location of the port declaration of the port `<port-name>` that is not in the recommended order:

Port '<port-name>' not defined in recommended order

Tag Severity

Warning

Suggested Fix

Reorder port definitions to comply with this convention.

Examples (Verilog)

Consider the following example where the output port `q_output` is declared before the input ports:

```
module test(q_output, a_in, b_in, rst, clk);
    input rst, clk;
    input a_in, b_in ;
    output q_output;
    ...
endmodule
```

Examples (VHDL)

Consider the following example where the ports are not declared in recommended order:

```
library ieee;
use ieee.std_logic_1164.all

entity test is
    port( inout1: inout std_logic;
          t1: linkage std_logic;
          in1: in std_logic ;
          out1: out std_logic;
          in2: in std_logic;
          out2: out std_logic;
          t2: buffer std_logic;
          inout2: inout std_logic
        );
end test;
```


SigHierName

Non-clock signal does not preserve a similar name through the hierarchy

Language

Verilog, VHDL

Description

Verilog

The SigHierName Tag flags those non-clock signals that do not maintain same or similar name through the hierarchy.

The SigHierName Tag flags those cases where a signal connected to a non-clock pin on a macro instance does not have the same or similar name as the pin to which it is attached. Thus, the two names should be exactly same or the signal name should contain the pin name.

By default, the SigHierName Tag checks all non-clock signals in the design. Use the [sighiernamelist](#) Tag parameter to specify your own list of signals to be checked.

By default, the SigHierName Tag reports only 500 messages per design. Set the [archname](#) Tag parameter to `yes` or `<Tag-name>` to report all violation messages.

Tracing signals through a design becomes significantly easier if the signals maintain similar names through the hierarchy.

VHDL

The SigHierName Tag flags those non-clock signals that do not maintain same or similar name through the hierarchy.

By default, the SigHierName Tag flags those cases where the name of a signal connected to a non-clock pin on a macro instance does not contain the name of the pin to which it is attached. Set the [SigHierName_option](#) Tag parameter to the same value to flag cases where the signal name is not the exact same name as the name of the pin to which it is connected.

By default, the SigHierName Tag checks all non-clock signals in the design. Use the [sighiernamelist](#) Tag parameter to specify your own list of signals to be checked.

By default, the SigHierName Tag reports only 20 messages per architecture

and 500 messages per design. Use the [archname](#) Tag parameter to report all the messages.

Tracing signals through a design becomes significantly easier if the signals maintain similar names through the hierarchy.

Related Standard

Section 5.2.1 of RMM

Message Details

Verilog

The following message appears at the location where the actual name *<actual-name>* and the formal name *<formal-name>* of a non-clock pin of an instance are not same or similar:

Names for actual '*<actual-name>*' and formal '*<formal-name>*' are not same or similar

VHDL

The following message appears at the location where the names of the connected signal *<sig-name>* and its pin driver *<pin-driver-name>* of a non-clock pin in an instance are not exactly same or the signal name does not contain the pin name (with the `SigHierName_option` Tag parameter not set or set to `similar`):

Names for signal '*<signal-name>*' and pin driver '*<pin-driver-name>*' are not same or similar

The following message appears at the location where the names of the signal *<sig-name>* and the pin driver *<pin-driver-name>* of a non-clock signal in an instance are not exactly same (with the `SigHierName_option` Tag parameter set to `same`):

Names for signal '*<signal-name>*' and pin driver '*<pin-driver-name>*' are not same

Tag Severity

Guideline

Suggested Fix

Modify all references to this name to a form that meets the convention.

Examples (Verilog)

Consider the following example:

```
...
gatedClk gc(.in(in[4:1]), .rst(rst1),
            .out(o_u_t[2]));
...
```

Here, the actual and formal names (`in`) are exactly same for the first connection, the actual name contains the formal name in the second connection, but the actual (`o_u_t`) and formal name (`out`) are not exactly same nor does the actual name (`o_u_t`) contains the formal name (`out`) in the third connection.

Thus, the third connection is flagged.

Examples (VHDL)

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

...
U1: dfflop port map (
    clk => clk, reset => reset, in => input1,
    q => output);
...
```

Here, the first connection relates to a clock pin and is hence ignored, the actual and formal names (`reset`) are exactly same for the second connection, the actual name (`input1`) contains the formal name (`in`) in the third connection, but the actual (`output`) and formal name (`q`) are not exactly same nor does the actual name (`output`) contains the formal name (`q`) in the fourth connection.

Thus, the SigHierName Tag flags the fourth connection if the SigHierName_option Tag parameter is not set or is set to similar

and flags both the third and fourth connections if the `SigHierName_option` Tag parameter is set to same.

[OpenMore Tags](#)

SignalComment

Signal declaration does not have a comment

Language

Verilog, VHDL

Description

Verilog

The SignalComment Tag flags signal declarations that do not have comments in the same or immediately preceding line.

The SignalComment Tag checks signal declarations of the following types:

reg	wire	tri	tri1	supply0	wand	triand
tri0	supply1	wor	trior			

A comment explains clearly the purpose and output of the signal, thus increasing understanding for future reuse.

By default, the SignalComment Tag flags only 30 messages per file and 600 messages per design. Set the *archname* Tag parameter to *yes* or *<TagName>* to report all violation messages.

VHDL

The SignalComment Tag flags all signal declarations that do not have a corresponding comment in the same or immediately preceding line.

A comment explains clearly the purpose and output of the signal, thus increasing understanding for future reuse.

By default, the SignalComment Tag flags only 30 messages per file and 600 messages per design. Set the *archname* Tag parameter to *yes* or *<TagName>* to report all violation messages.

Related Standard

Section 5.2.5 of RMM

Parameter(s)

- [ignore_below_line](#): By default, the parameter is set to `no`. Set the value of the parameter to `yes` to ignore the comment following (next line) the RTL construct.
- [ignore_empty_lines](#): By default, the parameter is set to `-1` and the tag ignores any number of white spaces/blank lines before the construct to check the comment. Set the value of the parameter to 0 or any positive integer to restrict the number of blank lines before the construct to match the comment. That is, by setting the value to 0 or any positive integer, the tag ignores that number of empty lines as specified in the value.

Message Details

The following message appears at the location where a signal is declared without a corresponding comment in the same line:

```
signal declaration does not have comments
```

Tag Severity

Guideline

Suggested Fix

Add a comment to each signal declaration, explaining the purpose of that signal.

[OpenMore Tags](#)

SigType

Signal type is not a type or subtype of std_logic or std_logic_vector

Language

VHDL

Description

The SigType Tag flags signal declarations that are not of type or subtype of `std_logic` or `std_logic_vector`.

Using `std_logic` or `std_logic_vector` has the following advantages:

1. These types are better supported by EDA tools.
2. These types provide resolution functions for tristate signals.

Following this guideline maintains the same standard across the design and helps in resolving multiple drivers.

Message Details

The following message appears at the location of the declaration of a signal `<signal-name>` that is of type other than `std_logic` or `std_logic_vector`:

```
Type of signal '<signal-name>' is not std_logic or  
std_logic_vector
```

Tag Severity

Guideline

Suggested Fix

Use standard IEEE types wherever possible.

Examples (VHDL)

Consider the following example where the signal type is not `std_logic` or `std_logic_vector`:

```
library ieee;
```

```
use ieee.std_logic_1164.all

entity andgate is
    port (in1 : in bit; --signal type is not std_logic
          in2 : in bit;  --signal type is not std_logic
          output : out bit --signal type is not std_logic
          );
end andgate;

architecture behav of andgate is
    begin
        output <= in1 and in2;
    end behav;
```


[OpenMore Tags](#)

TriStateName

Tristate signal does not follow naming convention

Language

Verilog, VHDL

Description

The TriStateName tag flags those tristate signal names that do not follow the recommended naming convention.

Following this tag makes it easy to identify tristate signals which should be treated with special care.

Parameter(s)

- **triname**: The default value is `"/_z$/"`. Set the value of the parameter to a perl regular expression pattern for a tristate name to define your own regular expression pattern for naming the tristate signals.

Message Details

With default name check

The following message appears at the first location where a violating tristate signal `<signal-name>` is assigned:

Tristate signal name '`<signal-name>`' should end with `_z`

With user-defined name check

If you have specified your own regular expression using the **triname** parameter, then the following message appears at the first location where a tag-violating tristate signal `<signal-name>` is assigned:

Tristate signal name '`<signal-name>`' should conform to the regular expression `$triname`

Suggested Fix

Modify all references to this name to a form that meets the convention.

Severity

Guideline

Examples (Verilog)

Consider the following example where the tristate name does not follow the recommended naming convention:

```
module m (in1, in2, en1, en2, out);
    input [7:0] in1, in2;
    input en1, en2;
    output [3:0] out;
    reg [3:0] out;
    always @ (en1 or en2 or in1 or in2)
        begin
            out = en1?in1[3:0]:4'bz;
            out = en2?in2[6:3]:4'bz;
        end
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Tristate signal name 'out[3:0]' should end with _z

Examples (VHDL)

Consider the following example where the signal `outp` is a tristate signal but it is not named as per the defined naming convention (names ending with `_z`):

```
library ieee;
use ieee.std_logic_1164.all

process (in1, en)
begin
    if (en) then outp <= 'Z';
        else outp <= in1;
    end if;
end process;
```

For this example, VC SpyGlass Lint generates the following message:

Tristate signal name 'outp' should end with _z

To fix this message, change the signal name as per the defined naming convention.

[OpenMore Tags](#)

TypeComment

Type/Sub-type is not commented

Language

VHDL

Description

The TypeComment Tag flags all type or subtype declarations that do not have a corresponding comment in the same or immediately preceding line.

By default, the TypeComment Tag flags only 30 messages per file and 600 messages per design. Set the [archname](#) Tag parameter to *yes* or *<TagName>* to report all violation messages.

A comment explains clearly the purpose of the type or subtype, thus increasing understanding for future reuse.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the location where a type or subtype is defined without a corresponding comment:

Type definition is not commented

Tag Severity

Tag

Suggested Fix

Add a comment to briefly explain the purpose of this type.

TypeCount

Excessive number of subtypes created

Language

VHDL

Description

The TypeCount flags those types that have more than recommended number of subtypes defined.

Using a large number of subtypes makes the design difficult to understand.

Related Standard

Section 5.3.1 of RMM

Tag Arguments

By default, the TypeCount Tag flags those types that have more than two subtypes defined. You can modify the recommended number of the subtypes in the PERL control script.

Message Details

The following message appears at the first line of a design unit `<du-name>` that has `<num1>` subtype definitions which exceeds the recommended number (`<num2>`) of subtypes:

Design Unit '`<du-name>`' has `<num1>` subtypes. Approved max is `<num2>`

Tag Severity

Guideline

Suggested Fix

Eliminate seldom-used sub-types if possible.

Examples (VHDL)

Consider the following example where excessive number of subtypes

[OpenMore Tags](#)

defined:

```
library ieee;  
use ieee.std_logic_1164.all
```

package pack is

```
    subtype type1 is integer range 0 to 5;  
    subtype type2 is integer range 5 to integer'high;  
    subtype type3 is integer range 0 to integer'high;  
end pack;
```

```
use work.pack.all;  
entity ent is  
    port( in1 : in type1;  
          in2 : in type3;  
          in3 : out type2);  
end ent;
```

VariableComment

Variable declaration does not have a comment

Language

Verilog, VHDL

Description

The VariableComment Tag flags variable declarations that do not have a comment in the same or immediately preceding line.

For Verilog designs, the VariableComment Tag flags un-commented variable declarations of `real`, `integer`, `realtime`, and `time` types.

A comment explains clearly the purpose and output of the variable, thus increasing understanding for future reuse.

By default, the VariableComment Tag flags only 30 messages per file and 600 messages per design. Set the [archname](#) Tag parameter to `yes` or `<Tag-name>` to report all violation messages.

Related Standard

Section 5.2.5 of RMM

Parameter(s)

- [ignore_below_line](#): By default, the parameter is set to `no`. Set the value of the parameter to `yes` to ignore the comment following (next line) the RTL construct.
- [ignore_empty_lines](#): By default, the parameter is set to `-1` and the tag ignores any number of white spaces/blank lines before the construct to check the comment. Set the value of the parameter to 0 or any positive integer to restrict the number of blank lines before the construct to match the comment. That is, by setting the value to 0 or any positive integer, the tag ignores that number of empty lines as specified in the value.

Message Details

The following message appears at the location where a variable is defined without a corresponding comment in the same line:

[OpenMore Tags](#)

variable declaration does not have comments

Tag Severity

Guideline

Suggested Fix

Add a comment to each variable declaration, explaining the purpose of that variable.

ParamName

Parameter/generic name does not follow the naming convention

Language

Verilog, VHDL

Description

The ParamName Tag flags the parameter/generic names that do not follow the specified naming convention.

By default, the ParamName Tag flags the following types of parameter/generic names:

- Names that start with a non-alphabetical character
- Names that contain a lowercase alphabet (naming convention requires only uppercase alphabets)
- Names that contain characters other than alphanumeric characters and underscores
- Names that contain more than 8 characters (reduced readability)
- Names that contain two successive underscores (unsupported in some hardware accelerators and emulators)

In Verilog, the ParamName Tag flags violation for local parameter names that do not follow the specified naming convention.

Use the [paramname](#) Tag parameter to define your own regular expression pattern for naming the parameters.

Following the convention checked by this Tag ensures your design will encounter minimum parameter/generic-naming problems across a wide range of EDA tools.

Use the [check_globals](#) parameter to report a violation for parameters defined in the global scope.

Related Standard

Section 5.2.1 of RMM

Message Details

Verilog

The following message appears at the location where a Tag-violating parameter name `<param-name>` is first encountered:

Parameter name `<param-name>` does not follow naming convention

VHDL

The following message appears at the location where a Tag-violating generic name `<gen-name>` is first encountered:

Generic name '`<gen-name>`' does not follow naming convention

Tag Severity

Guideline

Suggested Fix

Modify all references to this name to a form that meets the convention.

Examples (Verilog)

Consider the following example:

```
module ParamName_mod2(q, qb, data, clk);
    input data, clk;
    output q, qb;
    reg q, qb;
    parameter P__DELAY = 1; //default delay parameter
    always @(posedge clk)
    begin
        # P__DELAY q = data;
        # P__DELAY qb = ~data;
    end
endmodule
```

Examples (VHDL)

Consider the following example where the generic name abc does not follow naming convention:

```
library ieee;
use ieee.std_logic_1164.all

entity test is
    generic ( abc : integer := 1);
    port ( SI : in std_logic_vector(abc downto 0);
          SO : out std_logic_vector(abc downto 0)
          );
end entity;

architecture logic of test is
begin
    SO <= SI;
end logic;
```

[OpenMore Tags](#)

PortGrpComment

Port group is not commented

Language

VHDL

Description

The PortGrpComment Tag flags port group declarations that do not have a preceding comment line.

By default, the PortGrpComment Tag flags only 30 messages per file and 600 messages per design. Set the [archname](#) Tag parameter to `yes` or `<Tag-name>` to report all violation messages.

It is recommended to group port declarations by type (input, output, inout, linkage, and buffer) and then provide comments for each group.

A comment can explain port grouping, thus increasing understanding for future reuse.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the location where a group of ports is declared without a corresponding comment:

Need a comment to describe group of ports

Tag Severity

Guideline

Suggested Fix

Add a comment before each port group.

PortName

Port name does not follow the naming convention

Language

Verilog, VHDL

Description

The PortName Tag flags the port names that do not follow the specified port naming convention.

For VHDL, the PortName Tag flags port names inside entities as well as components.

By default, the PortName Tag flags the following types of port names:

- Names that start with a non-alphabetical character
- Names that contain an uppercase alphabet (naming convention requires only lowercase alphabets)
- Names that contain characters other than alphanumeric characters and underscores
- Names that contain more than 16 characters (reduced readability)
- Names that contain two successive underscores (unsupported in some hardware accelerators and emulators)

Use the [portname](#) Tag parameter to define your own regular expression pattern for naming the ports.

Following the convention checked by this Tag ensures your design will encounter a minimum of port-naming problems across a wide range of EDA tools.

Related Standard

Section 5.2.1 of RMM

Parameter(s)

- [ignore_file](#): Default value is no. Set the value of the parameter to yes to enable the *PortName* Tag to not report violations for a Verilog file. This parameter supports regular expressions and is applicable for Verilog only.

- [inout_name](#): Default value is "". Set the value of the parameter to a perl regular expression pattern that an `inout` port name should follow.
- [input_name](#): Default value is "". Set the value of the parameter to a perl regular expression pattern that an `input` port name should follow.
- [output_name](#): Default value is "". Set the value of the parameter to a perl regular expression pattern that an `output` port name should follow.
- [print_recommended_regexp](#): Default value is no. If you set this parameter to yes or comma-separated Tag list, the violation messages of the Tag include the recommended regular expression pattern that is used.
- [check_reserve_word](#): Default value is no. Set the value of the parameter to yes to enable the tag to report a violation when the name of the port is one of the following c/c++ reserve words.

Verilog:

auto, break, char, const, continue, default, do double, enum, extern, float, goto, int, long, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, and malloc.

VHDL:

auto, char, default, double, float, goto, long, short, signed, sizeof, struct, switch, unsigned, volatile, and malloc.

Message Details

The following message appears at the location where a Tag-violating port name `<port-name>` is first encountered:

Port `<port-name>` does not follow naming convention

If you set the [print_recommended_regexp](#) parameter, the following message is reported that includes the recommended regular expression pattern:

Port `<port-name>` does not follow naming convention. Recommended pattern is '`<regular-expression-pattern>`'

Tag Severity

Guideline

Suggested Fix

Modify all references to this name to a form that meets the convention.

Examples (Verilog)

Consider the following example where port names start with a letter, are below 16 characters and do not have successive underscores. But this example shows a message because it contains upper score letters.

```
module PortName_mod4 (in1, in2, q1, q2, Clk);
    input in1, in2, Clk;
    output q1, q2;
    wire q1, q2;
    wire reset;
    Flop_one U_FF1
        (.Clk(Clk), .din(in1), .res(reset), .qout(q1));
    Flop_two U_FF2
        (.Clk(Clk), .din(in2), .res(reset), .qout(q2));
endmodule
```

Examples (VHDL)

Consider the following example, where the port names contain characters in upper case that causes Tag-violation:

```
library ieee;
use ieee.std_logic_1164.all

entity char_type1 is
    port ( in1: bit_vector(3 downto 0);
           IN2: bit_vector(3 downto 0);
           IN223n: bit_vector(3 downto 0);
           IN3: bit_vector(3 downto 0);
           OUT1: out bit_vector(3 downto 0)
        );
    signal signala : bit;
end char_type1;

architecture rtl of char_type1 is
    signal tmp : character;
```

```

function func (
    signal in1: character;
    signal in2,in3: bit_vector( 3 downto 0))
    return bit_vector is
begin
    case In1 is
        when 'a' => return in2 or in3;
        when 'b' => return in2 and in3;
        when 'c' => return in2 nand in3;
        when 'd' => return in2 xor in3;
        when others => return in2 nor in3;
    end case;
end;

begin
    test_PROC: process(in1,IN2,in3)
        variable var : character;
        procedure proc1 (
            variable data_v: bit;
            signal    s_data: bit ) is
            begin
            end proc1;
        begin
            case in1 is
                when "0000" => var:='a';
                when "0010" => var:='b';
                when "0100" => var:='c';
                when "0110" => var:='d';
                when "1000" => var:='e';
                when "1010" => var:='f';
                when "1100" => var:='g';
                when others => var:= 'h';
            end case;
            tmp <= var;
            out1<= func(tmp,IN2,in3);
        end process;
    end;

```

ProcessComment

Process is not commented

Language

VHDL

Description

The ProcessComment Tag flags all process definitions that do not have a corresponding comment on the same or immediately preceding line.

A comment explains clearly the function of the block (combinational or sequential), thus increasing understanding for future reuse.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the location where a process block is defined without a corresponding comment:

Process block is not commented

Tag Severity

Tag

Suggested Fix

Add a comment to briefly explain the purpose of this process.

[OpenMore Tags](#)

SigName

Signal name does not follow the naming convention

Language

Verilog, VHDL

Description

The SigName Tag flags signal names that do not follow the specified naming convention.

By default, the SigName Tag flags the following types of signal names:

- Names that start with a non-alphabetical character
- Names that contain an uppercase alphabet (naming convention requires only lowercase alphabets)
- Names that contain characters other than alphanumeric characters and underscores
- Names that contain more than 32 characters (reduced readability)
- Names that contain two successive underscores (unsupported in some hardware accelerators and emulators)

Use the [signame](#) Tag parameter to define your own regular expression pattern for naming the signals. For Verilog, checking is done for the following types of signals:

wire	tri	tri0	tri1	supply0	supply1
wor	wand	trior	triand	trireg	reg

Following the convention checked by this Tag ensures your design will encounter a minimum of signal-naming problems across a wide range of EDA tools.

By default, the [print_recommended_regexp](#) parameter is set to no. If you set this parameter to yes or comma-separated Tag list, the violation messages of the *SigName* Tag include the recommended regular expression pattern that is used.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the location where a Tag-violating signal name `<sig-name>` is first encountered:

signal `<sig-name>` does not follow naming convention

If you set the [print_recommended_regexp](#) parameter, the following message is reported that includes the recommended regular expression pattern:

Signal `<sig-name>` does not follow naming convention.
Recommended pattern is '`<regular-expression-pattern>`'

Tag Severity

Guideline

Suggested Fix

Modify all references to this name to a form that meets the convention.

Examples (Verilog)

Consider the following example, where the signal names start with a letter, are less than 32 characters long and have only lower-case letters and numerals:

```
module SigName_mod2(qout, qbar, data_input, clk);
    input data_input, clk;
    output qout, qbar;
    reg qout, qbar;
    wire default__inp1, default__inp2;
    parameter P_DELAY = 1; //default delay parameter
    always @(posedge clk)
    begin
        # P_DELAY qout = data_input & default__inp1;
        # P_DELAY qbar = ~data_input & default__inp2;
    end
endmodule
```

The SigName Tag reports a message because the signal names have successive underscores.

Examples (VHDL)

Consider the following example, where the signal names have characters in upper case that violates the SigName Tag:

```
library ieee;
use ieee.std_logic_1164.all

entity char_type1 is
    port ( in1: bit_vector(3 downto 0);
          in2: bit_vector(3 downto 0);
          in3: bit_vector(3 downto 0);
          out1: out bit_vector(3 downto 0)
        );
    signal SIGNa1A : bit;
    procedure proc (
        variable data_v: bit;
        signal S_Data: bit ) is
    begin
    end proc;
end char_type1;

architecture rtl of char_type1 is
    signal tmp : character;
    signal signalcheckedA : bit;
    signal signalcheckedABC : bit;
    function func (
        signal In1: character;
        signal in2,in3: bit_vector( 3 downto 0)) return
        bit_vector is
    begin
        case In1 is
            when 'a' => return in2 or in3;
            when 'b' => return in2 and in3;
            when 'c' => return in2 nand in3;
            when 'd' => return in2 xor in3;
```

```
        when others => return in2 nor in3;
    end case;
end;

begin
test_PRC: process(in1,IN2,in3)
variable var : character;
procedure procl (
    variable data_v: bit;
    signal S_Data: bit ) is
    begin
end procl;
begin
    case in1 is
        when "0000" => var:='a';
        when "0010" => var:='b';
        when "0100" => var:='c';
        when "0110" => var:='d';
        when "1000" => var:='e';
        when "1010" => var:='f';
        when "1100" => var:='g';
        when others => var:= 'h';
    end case;
    tmp <= var;
    out1<= func(tmp,IN2,in3);
end process;
end;
```

[OpenMore Tags](#)

TaskComment

A comment does not appear immediately before a task definition

Language

Verilog

Description

The TaskComment Tag flags all task definitions that do not have a preceding comment line.

A comment explains clearly the purpose and output of the task, thus increasing understanding for future reuse.

Related Standard

Section 5.2.5 of RMM

Message Details

The following message appears at the location where a task is defined without a corresponding comment in the preceding line:

Each task should be preceded by a comment

Tag Severity

Tag

Suggested Fix

Add a comment before the task explaining the purpose of the task, arguments and any restrictions on use of the task.

TypeName

Type or sub-type does not follow recommended naming convention

Language

VHDL

Description

The TypeName Tag flags the type and subtype names that do not follow specified naming conventions.

By default, the TypeName Tag flags the following types of type and subtype names:

- Names that start with a non-alphabetical character
- Names that contain a lowercase alphabet (naming convention requires only uppercase characters)
- Names that contain characters other than alphanumeric characters and underscores
- Names that contain more than 32 characters (reduced readability)
- Names that contain two successive underscores (unsupported in some hardware accelerators and emulators)

Use the [typename](#) Tag parameter to specify your own naming convention.

Following the convention checked by this Tag ensures your design will encounter a minimum of naming problems across a wide range of EDA tools.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the location where a Tag-violating type or subtype name `<type-name>` is first encountered:

Type/subtype name '`<type-name>`' does not follow naming convention

[OpenMore Tags](#)

Tag Severity

Guideline

Suggested Fix

Modify all references to this name to a form that meets the convention.

Examples

Consider the following example:

```

entity char_type1 is
  port ( in1: bit_vector(3 downto 0);
         in2: bit_vector(3 downto 0);
         in3: bit_vector(3 downto 0);
         out1: out bit_vector(3 downto 0));
  constant SIGNALB : bit := '0';
  type A_typ is range 1 to 10;
  subtype C_Typ is integer range 1 to 10;
  subtype X_TYP is integer range 11 to 20;
  procedure proc (
    variable data_v: bit;
    constant S_DATA: bit ) is
    type Enum_Typ is (R,G,B);
    type ENUM_TYP1 is (R,G,B,Y);
    subtype R_Typ is real range 5.0 to 10.0;
    subtype R_TYP1 is real range 5.5 to 10.5;
    begin
    end proc;
end char_type1;

```

UseDefine

Hard constants used to encode finite-state machine

Language

Verilog, VHDL

Description

The UseDefine Tag flags hard constants used in Finite-state machine (FSM) definition.

Optimizing FSM encoding is an important part of optimizing final design performance. Exploring multiple implementation options is significantly easier if encoding is defined through macros or parameters (Verilog) or enums (VHDL), rather than being hard-coded in the case statement. For Verilog, use of either macros or parameters consistently within one FSM is acceptable.

Related Standard

Section 5.5.9 of RMM

Parameter(s)

[*encoding_type*](#): The default value of the parameter is parameter, macro. Set the value of the parameter to one or more from the comma-separated list of parameter, macro, enum tags, to define the encoding types for finite state machines (FSM). This parameter is only applicable to Verilog and supports the encoding parameter,macro,enum styles.

Message Details

The following message appears at the first line of a construct that uses hard constants:

Verilog

State Machine should use macros or parameters for states

VHDL

State Machine should be enumerated data type

[OpenMore Tags](#)

Tag Severity

Guideline

Suggested Fix

Define macros or parameters (Verilog) or an enum (VHDL) for the state values and use those values rather than hard constants.

Examples (Verilog)

Consider the following example where hard constants are used for encoding finite-state machine:

```
...
case (C_STATE)
  2'b00 : begin
    FSMOUT = 3'b111;
    N_STATE = 2'b01;
  end
  2'b01 : begin
    FSMOUT = 3'b101;
    N_STATE = 2'b11;
  end
  2'b11 : begin
    FSMOUT = 3'b110;
    N_STATE = 2'b10;
  end
  2'b10 : begin
    FSMOUT = 3'b011;
    N_STATE = 2'b00;
  end
  default : begin
    FSMOUT = 3'b000;
    N_STATE = 2'b00;
  end
endcase
...
```

Examples (VHDL)

Consider the following example case construct that uses hard constants (0, 1, 2, 3) to assign the state variable `next_state`:

```
library ieee;
use ieee.std_logic_1164.all

...
case curr_state is
  when 0 => next_data <= data;
            next_state <= 1;
  when 1 => next_data <= curr_data or data;
            next_state <= 2;
  when 2 => next_data <= curr_data and data;
            next_state <= 3;
  when 3 => next_data <= curr_data xor data;
            next_state <= 0;
  when others => next_data <= (others => '0');
                next_state <= 0;
end case;
...
```

For this example, VC SpyGlass Lint generates the UseDefine message for the case statement.

To fix this message, first define an enumerated type (say `state_list`) as follows:

```
type state_list is ('0','1','2','3');
```

Then, rewrite the example CASE statement using the enumerated type `state_list` as follows:

```
library ieee;
use ieee.std_logic_1164.all

case curr_state is
  when 0 => next_data <= data;
            next_state <= state_list(1);
  when 1 => next_data <= curr_data or data;
            next_state <= state_list(2);
```

[OpenMore Tags](#)

```
when 2 => next_data <= curr_data and data;  
    next_state <= state_list(3);  
when 3 => next_data <= curr_data xor data;  
    next_state <= state_list(0);  
when others => next_data <= (others => '0');  
    next_state <= state_list(0);  
end case;
```

VarName

Variable does not follow recommended naming convention

Language

Verilog, VHDL

Description

The VarName Tag flags the variable names that do not follow the specified variable naming convention.

By default, the VarName Tag flags the following types of variable names:

- Names that start with a non-alphabetical character
- Names that contain an uppercase alphabet (naming convention requires only lowercase alphabets)
- Names that contain characters other than alphanumeric characters and underscores
- Names that contain more than 32 characters (reduced readability)
- Names that contain two successive underscores (unsupported in some hardware accelerators and emulators)

NOTE: *The VarName tag does not perform check on enum elements.*

Use the [varname](#) Tag parameter to define your own regular expression pattern for naming the variables. For Verilog, checking is done for the following types of variables:

shortreal	real	time	integer	bit	byte
realtime	int	shortint	longint	logic	

Following the convention checked by this Tag ensures your design will encounter a minimum of variable-naming problems across a wide range of EDA tools.

By default, the [print_recommended_regexp](#) parameter is set to no. If you set this parameter to yes or comma-separated Tag list, the violation messages of the Tag include the recommended regular expression pattern that is used.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the location where a Tag-violating variable name `<var-name>` is first encountered:

variable `<var-name>` does not follow naming convention

If you set the [print_recommended_regexp](#) parameter, the following message is reported that includes the recommended regular expression pattern:

variable `<var-name>` does not follow naming convention.
Recommended pattern is '`<regular-expression-pattern>`'

Tag Severity

Guideline

Suggested Fix

Modify all references to this name to a form that meets the convention.

Examples (Verilog)

Consider the following example where the variable names start with a letter, are less than 32 characters long and have only lower-case letters and numerals. This example shows a message because the variable names contain successive underscores.

```
module VarName_mod2(qout, sel, data_in1, data_in2, clk);
  input data_in1, data_in2, clk;
  input [1:0] sel;
  output qout;
  reg qout;
  wire default_in1, default_in2;
  integer my_int1, my__int2;
  always @(posedge clk)
    if(my_int1 > my__int2)
      case(sel)
        2'b00: qout = data_in1;
        2'b01: qout = default_in1;
```

```

        2'b10: qout = data_in2;
        2'b11: qout = default_in2;
    endcase
endmodule

```

Examples (VHDL)

Consider the following example, where the variable names contain characters in upper case that causes Tag-violation:

```

library ieee;
use ieee.std_logic_1164.all

entity char_type1 is
    port ( in1: bit_vector(3 downto 0);
          in2: bit_vector(3 downto 0);
          in3: bit_vector(3 downto 0);
          out1: out bit_vector(3 downto 0)
        );
    signal signala : bit;
    procedure proc (
        variable Data_v: bit;
        variable data: bit;
        signal s_data: bit ) is
        variable InProc: character;
        variable inpr: character;
    begin
    end proc;
end char_type1;

architecture rtl of char_type1 is
    signal tmp : character;
    function func (
        signal in1: character;
        signal in2,in3: bit_vector( 3 downto 0)) return
bit_vector is
        variable InFunc: character;
        variable nunc: character;
    begin

```

```

        case In1 is
            when 'a' => return in2 or in3;
            when 'b' => return in2 and in3;
            when 'c' => return in2 nand in3;
            when 'd' => return in2 xor in3;
            when others => return in2 nor in3;
        end case;
    end;

begin
test_PROC: process(in1,IN2,in3)
    variable VAR : character;
    variable var1 : character;
    procedure procl (
        variable Data_v: bit;
        signal    s_data: bit ) is
    variable InProc: character;
    begin
    end procl;
begin
    case in1 is
        when "0000" => var:='a';
        when "0010" => var:='b';
        when "0100" => var:='c';
        when "0110" => var:='d';
        when "1000" => var:='e';
        when "1010" => var:='f';
        when "1100" => var:='g';
        when others => var:= 'h';
    end case;
    tmp <= var;
    out1<= func(tmp,IN2,in3);
    end process;
end;

```

BufClock

Detects clock buffers

When to Use

Use this Tag to detect buffered clocks in the design.

Description

The *BufClock* Tag reports violation for the buffered clocks in the design.

The *BufClock* Tag ignores inferred buffers or assignments and checks for only ASIC cell buffer instantiations, where the buffer type is defined by an associated .lib library.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

- **strict**: The default value of the strict parameter is no. Set the value of the parameter to yes or *BufClock* to report violation for libraries specified in the following project file commands:

```
-v <file-names>  
-y { <space-separated path-names of directories> }
```
- **ignoreCellName**: By default, the ignoreCellName parameter is not set. Set this parameter to a PERL regular expression to specify the cell names that need to be ignored, through which the clocks are inverted.

Messages and Suggested Fix

Message 1

The following message appears at the first line of a module where a clock signal *<clk-name>* that defines the sequential element *<flop-name>* has been buffered:

[Guideline] Buffered Clock '*<clock-name>*' (flop: *<flop-name>*) detected

Potential Issues

Clock buffering is much more effectively handled by physical design tools. It is not possible to effectively manage clock skew prior to physical design.

Consequences of Not Fixing

If you follow the common practice of using a place-holder buffer for clock-tree synthesis, you should waive violations for those cells. However, using that technique would be a questionable practice for a block that is intended to be reusable (all clock-tree synthesis should be managed from the top of the design).

This means that using clock buffers reduces reusability for blocks.

How to Debug and Fix

The violation message indicates the buffer output net name and flip-flop output net name.

View the Incremental Schematic of the violation message. The Incremental schematic shows the path from the flip-flop clock pin to the offending buffer.

To fix the violation, remove buffers present in the clock tree.

Example Code and/or Schematic**Example 1: Verilog**

Consider the following example where the `bufClock` clock signal is buffered using the cell buffer `B1I` before it is used in the flip-flop description out:

```
module test(clock, in, out);
    input  clock;
    input  [63:0] in;
    output [63:0] out;

    reg    [63:0] out;
    wire   bufClock;
    B1I b( .A(clock), .Z(bufClock));

    always@(negedge bufClock)
        out = in;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Buffered Clock 'bufClock'(flop: test.out[0]) detected

Example 2: VHDL

Consider the following example where the clock signal `clk11` defines the sequential element `output1` and is buffered through instance `i_c_test1` of component `c_test1`:

```
library ieee;
use ieee.std_logic_1164.all

entity test is
  port(d1, ck1: in bit;
       output1: out bit);
end test;

architecture rtl of test is
  component c_test1 is
    port(A: in bit;
         Z: out bit);
  end component;

  signal ck11: bit;

begin
  i_c_test1:c_test1 port map (A=>ck1, Z=>ck11);
  process(ck11)
  begin
    if(ck11'event and ck11 = '1') then
      output1 <= d1;
    end if;
  end process;
end rtl;
```

For this example, VC SpyGlass Lint generates the following message:

Buffered Clock 'ck11'(flop: rtl.output1) detected

Example 3:

Consider the following example, where the Tag reports a violation because

the lib cell BUF is not caught by the default regular expression value "" as the *ignoreCellName* parameter is not set:

```
module top(input clk, a, b, sel, output reg y);
    wire w;
    always @ (posedge w)
    begin
        y = 0;
        if (sel == 0) begin
            y = a;

        end else begin
            y = b;
        end
    end
    CK_NOT n(clk, w); //Violation
endmodule
```

Example 4:

Consider the following example, where the Tag does not report a violation because the lib cell BUF is caught by the regular expression value "CK.*" as specified through the *ignoreCellName* parameter:

```
always @ (posedge w)
begin
    y = 0;
    if (sel == 0) begin
        y = a;
    end else begin
        y = b;
    end
end
CK_NOT n(clk, w); //No violation
endmodule
```

Default Severity Label

Guideline

Tag Group

Guidelines for Clocks and Resets

Report and Related File

No related reports or files

Related Standard

Section 5.4.2 of RMM

ClkHierName

Clock signal does not keep the same name through the hierarchy

Language

Verilog, VHDL

Description

The ClkHierName Tag flags those clock names that are not preserved through the hierarchy.

By default, the ClkHierName Tag flags a separate message for each case where a clock does not preserve its name across the hierarchy. Set the [ignoreMultipleInstances](#) Tag parameter to report only one message when a clock is connected to multiple instances of a design unit and the clock name is not preserved across the hierarchy.

Consider the following example where clock `clk` is connected to pin `CP` of design unit `A` in two instances `I1` and `I2`:

By default, the ClkHierName Tag flags separate message for each instance. When you set the `ignoreMultipleInstances` Tag parameter, the ClkHierName Tag flags only one message for both cases.

Tracing clock signals through a design becomes significantly easier if those signals maintain the same names through the hierarchy. This is especially true if the design has multiple clocks of generated clocks.

To comply with this Tag, a signal connected to a clock pin on a macro instance must have exactly the same name as the pin.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the location where the lower-level clock name `<low-clk-name>` in instance `<inst-name>` does not match the higher-level clock name `<high-clk-name>`:

clock signal name '`<high-clk-name>`' does not match lower-level

```
name '<low-clk-name>' (instance <inst-name>)
```

Tag Severity

Guideline

Suggested Fix

Modify the clock signal name to conform with the convention.

Examples (Verilog)

Consider the following example where clock signal name `clk` does not match lower level name `clk_d`:

```
module ClkHierName(a_in, B_in, rst, clk, q_out);
    input rst, clk;
    input a_in, B_in ;
    output q_out;

    dff df1(.d_in(a_in), .rst(rst),
           .clk_d(clk), .q_out(q_out));
endmodule
```

Examples (VHDL)

Consider the following example where the lower-level clock name `c1` of instance name `U1` does not match the higher-level clock name `clk`:

```
library ieee;
use ieee.std_logic_1164.all

entity test1 is
    port (c1, d: in STD_LOGIC;
          q: out STD_LOGIC);
end test1;

architecture rtl of test1 is
begin
    FF_PROC: process (c1)
    begin
```

[OpenMore Tags](#)

```

        if (c1'event and c1 = '1') then
            q <= d;
        end if;
    end process;
end rtl;

entity test is
    port (
        clk: in STD_LOGIC;
        rst: in STD_LOGIC;
        d: in STD_LOGIC;
        q: out STD_LOGIC
    );
end test;

architecture rtl of test is
    component a
        port (
            c1, d: in STD_LOGIC;
            q: out STD_LOGIC);
    end component;

    for all: a use entity work.test1(rtl)
    begin
        U1: a port map (c1 => clk, d => d, q => q);
    end rtl;

```

For this example, SpyGlass generates the following message:

Clock signal name 'c1' does not match lower-level name 'clk' (instance U1)

ClkName

Clock name does not follow the naming convention

Language

Verilog, VHDL

Description

The ClkName Tag flags clock signal names that do not follow the recommended naming convention.

By default, the ClkName Tag flags those clock signal names that are not `clk` or do not start with `clk_`. Use the [clkname](#) Tag parameter to define your own regular expression pattern for naming the clock signals.

Use the [clkname_vector](#) parameter to define a regular expression pattern for naming the vector type clock signals.

Following the convention checked by this Tag ensures that the clock signals stand out clearly from other signal names and variable names.

Related Standard

Section 5.2.1 of RMM

Message Details

With default name check

The following message appears at the location where a Tag-violating clock name `<clk-name>` is first encountered:

Clock signal name '`<clk-name>`' should be `clk` or start with `clk_`

With user-defined name check

If you have specified your own regular expression using the [clkname](#) Tag parameter, then the following message appears at the location where a Tag-violating clock name `<clk-name>` is first encountered:

Clock signal name '`<clk-name>`' should conform to the regular expression `$clkname`

[OpenMore Tags](#)

Tag Severity

Guideline

Suggested Fix

Modify the clock signal name to conform with the convention. Make sure you follow this name change through the hierarchy.

Examples (Verilog)

Consider the following example:

```
module test(w1_in, w2_in, clk, w1_out,
            w2_out, reset, bclk);
    input w1_in, w2_in, clk, reset, bclk;
    output w1_out, w2_out;

    reg w1_out, w2_out;

    always @(posedge bclk)
    begin
        if ( reset)
        begin
            w1_out <= 0;
            w2_out <= 0;
        end
        else
        begin
            w1_out <= w1_in;
            w2_out <= w2_in;
        end
    end
end
endmodule
```

Examples (VHDL)

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all
```

```
ENTITY myentity IS
  PORT (
    clock1 : in std_logic;
    clk2 : in std_logic;
    sig1 : IN std_logic;
    sig2 : IN std_logic;
    sig3 : OUT std_logic
  );
END myentity;

ARCHITECTURE myarch OF myentity IS
  BEGIN
    process (clk2, clock1)
    begin
      if ((clk2 = '1' AND clk2'event) OR
        (clock1 = '1' AND clk2'event)) then
        sig3 <= sig1 AND sig2;
      end if;
    end process;
  END myarch;
```

CombLoop

Detects combinational loops

When to Use

Use this Tag to detect combination loops in the design

Description

The *CombLoop* Tag reports violation for the combinational loops in the design.

NOTE: *When waiver is applied on block-level and is imported on a top-level, then sometime the waiver may not work, because entry point in combination loop may be different from top-level, resulting in different message and thus break in waivers. Therefore, you need to waive the violation manually.*

To limit the number of elements reported by the tag, use `lint_report_whole_path` app var, as shown below:

```
set_app_var lint_report_whole_path 4
```

By default, information for 3 nodes is reported in the violation message. Set the value of the app var to -1 to report all the nodes. You can also set the value of the app var to any other integer value to limit the number of nodes to be reported.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [*enableE2Q*](#): The default value is `no`. Set the value of the parameter to `yes` to detect loops that pass through enable pin to Q (output) pin path of a latch.
- [*enable_latch_based_clusters*](#): The default value is `no`. Set the value of the parameter to `yes` to report a violation for loops with more than one latch and also generate a report that is clustered into two parts:
 - ☐ loops with two or more latches

❑ other loops

- ***overlappingLoops***: The default value is *yes*. Set the value of the parameter to *no* to ignore the overlapping loops.
- ***ignoreSRlatch***: The default value is *no*. Set the value of the parameter to *yes* to ignore the SR-latch combinational loops.
- ***archname***: The default value is *no*. Set the ***archname*** Tag parameter to *yes* or *<Tag-name>* to report all violation messages.
- ***strict***: The default value is *no*. Set the value of the parameter to *yes* or *CombLoop* to enable the *CombLoop* Tag to report latches implemented through a MUX.
- ***report_mux_select***: The default value is *yes*. Set the value of the parameter to *no* to enable the *CombLoop* Tag to report combinational loops existing through the select pin of a MUX.
- ***report_flop_reset_loop***: The default value is *no*. Set the value of the parameter to *yes* to enable the Tag to report violation when a loop exists through set/reset pin and output pin of a flip-flop.
- ***report_flop_clock_loop***: The default value is *no*. Set the value of the parameter to *yes* to enable the Tag to report violation when a loop exists through clock pin and output pin of a flip-flop.
- ***effort_level***: The default value is 100. Set the value of the parameter to any positive integer to specify the number of times the *CombLoop* Tag should traverse a net to detect a combinational loop.
- ***ignore_iopad***: The default value is *no*. If you set this parameter to *yes*, the Tag does not report combinational loops that traverse input and output terminals of a pad cell.
- ***ignore_memory***: The default value is *no*. Set this parameter to *yes* to ignore the loops through combinational timing arcs of a sequential memory.
- ***check_loop_breakers***: The default value is *no*. Set this parameter to *yes* to check if the AND, OR and 2 bit MUX instances in the path are mutually exclusive loops. If a loop is mutually exclusive, no violation is reported for the loop.

- **net_count**: By default, this parameter is set to 10. Set this parameter to specify the number of nets of the loop that should be printed in violation message.

Tcl Commands

- **get_combloop**: This Tcl command creates a list of combinational loop in the current design that match certain criteria.

Messages and Suggested Fix

Message 1

[GUIDELINE] Combinational loop exists at '<signal-name-list>'

In the above message, the <signal-name-list> contains the list of signals in the combinational loop.

NOTE: *The combinatorial loops reported by the CombLoop Tag may also contain synthesis tool-generated nets that do not have the HDL correspondence. Their names will always be displayed as <synth_gen_net> so you must not try to search those nets in the HDL file.*

Potential Issues

A combinational loop is created when the output of a cell loops back onto its inputs directly or through many other combinational cells.

Consequences of Not Fixing

VC SpyGlass Lint DFT tools are generally unable to generate tests for combinational loop logic. Also, the timing behavior and ultimately functional behavior of logic in a combinational loop is highly dependent on the target technology because its behavior depends on gate and interconnect delays. Such designs may not be portable and may even have reliability problems if the timing dependencies are tight enough to fall within manufacturing process variations.

The output of a cell, which causes a combinational loop can:

- Get into a locked state

For example, consider a two input AND gate with output connected to one of the inputs. When other input of the AND gate is 0, the AND gate output is locked to a 0.

- Oscillate at potentially high frequency

For example, consider the output of an inverter being looped back to its input through an OR gate. When the OR gate is enabled, the inverter can oscillate between logic 0 and 1 (separated in time by the propagation delay of the loop). This can lead to unintended logic behavior and potentially high power consumption.

There can be variations in the scenarios specified above. Hence, it is best to avoid combinational loops.

How to Debug and Fix

The violation message lists all the nets for the violating loop, if the number of nets is less than or equal to ten. However, the violation message lists the first ten nets for the violating loop, if the number of nets is more than ten.

For SR type latches, use the hard macros from the vendor library where the macro is guaranteed to have deterministic timing and functional behavior.

To fix the violation, search a place to break the combinational loop with a flip-flop, unless the loop is introduced by mistake.

Message 2

[INFO] Please refer to 'CombLoopReport.rpt' for complete combinational loops

Potential Issues

The violation message appears only when [Message 1](#) is reported for the *CombLoop* Tag.

Consequences of Not Fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no fix is required for this message.

Example Code and/or Schematic

Verilog

Consider the following example:

```
module m0 (in1, out);  
    input [3:0] in1;
```

[OpenMore Tags](#)

```

output [7:0] out;
wire a;
assign a = ~in1[2];
assign out[3] = a;
assign in1[2] = ~out[3];
endmodule

```

In the above example, the CombLoop Tag reports a violation as a combinational loop is present in the design.

VHDL

Consider the following example:

```

library ieee;
use ieee.std_logic_1164.all

architecture rtl of test is
  component andgate
    port(a, b: in bit; c: out bit);
  end component;
  component orgate
    port(a, b: in bit; c: out bit);
  end component;
  component xorgate
    port(in1, in2: in bit; output: out bit);
  end component;
  signal s1, s2, s3, s4: bit;
begin
  INST1: andgate
    port map(a=>a, b=>b, c=>s1);
  INST2: orgate
    port map(a=>c, b=>s3, c=>s2);
  INST3: xorgate
    port map(in1=>s1, in2=>s2, output=>s3);
  output <= s3;
end rtl;

```

The following figure illustrates the functionality of the example architecture:

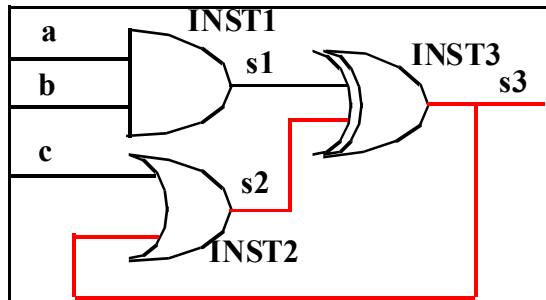


FIGURE 19. Example of a Combinational Loop

There is a combinational loop between the INST3 component instance and the INST2 component instance through signals s2 and s3.

For the above example, VC SpyGlass Lint generates the following message:
combinational loop exists at test.s3-test.s2

To fix this message, rewrite the design to remove the combinational loop.

Default Severity Label

Guidelines

Tag Group

Coding for Synthesis

Report and Related File

NA

Language

Verilog, VHDL

Related Standard

Section 5.5.4 of RMM

[OpenMore Tags](#)

GatedClock

Gated/generated clocks are generated from more than one block in the design

Language

Verilog, VHDL

Description

The GatedClock Tag flags gated clock signals.

Gating clocks creates skew and therefore timing problems. Such constructs also cause problems for ATPG tools.

While clock gating is necessary for low-power designs, the GatedClock Tag works as a tool to verify that the clock gating has been implemented only where intended.

Related Standard

Section 5.4.3 of RMM

Message Details

The following message appears at the first line of a module where a clock signal `<clk-name>` that defines the sequential element `<flop-name>` has been gated:

```
Gated clock '<clk-name>' (flop: <flop-name>) detected
```

Tag Severity

Guideline

Suggested Fix

No fix may be necessary, but review all gating to make sure it is where you expected. You should also consider whether it would be better to leave clock gating to the block user, rather than embedding it in the block.

Examples (Verilog)

Consider the following example where the clock signal, `clock1` has been

gated using the ASIC gate ND2:

```
module test(clock, in1, in2, out1, out2);
    input  clock;
    input  [63:0] in1, in2;
    output [63:0] out1, out2;

    reg    [63:0] out1, out2;
    wire    clock1, clock2;

    ND2 n(.A(clock), .B(clock2), .Z(clock1));

    assign clock2 = 1'b1;

    always@(posedge clock1)
        out1 = in1;

    always@(posedge clock1)
        out2 = in2;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:
Gated clock 'clock1' (flop: test.out2[63]) detected

Examples (VHDL)

Consider the following example where clock signal `clk_in` of sequential element `output1` has been generated by gating inputs `clk` and `en`:

```
library ieee;
use ieee.std_logic_1164.all

entity test is
    port (clk: in std_logic;
          en: in std_logic;
          input1: in std_logic;
          output1: out std_logic);
end test;

architecture rtl of test is
```

```

signal clk_gated: std_logic;

begin
    clk_gated <= clk and en;

    process(clk_gated)
    begin
        if clk_gated = '1' and clk_gated'event then
            output1 <= input1;
        end if;
    end process;
end rtl;

```

For this example, VC SpyGlass Lint generates the following message:

Gated Clock 'clk-gated' (flop: test.output1) detected

To fix this message, rewrite the example as follows:

```

entity test is
    port (clk: in std_logic;
          en: in std_logic;
          input1: in std_logic;
          output1: out std_logic);
end test;

architecture rtl of test is
    process(clk)
    begin
        if clk = '1' and clk'event then
            if en = '1' then
                output1 <= input1;
            end if;
        end if;
    end process;
end rtl;

```

InferLatch

Latch inferred

When to Use

Use this Tag to detect inferred latches in the design.

Description

The *InferLatch* Tag reports inferred latches in the design.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

- [*allow_latch_cell*](#): The default value is ' '. Set the value of the parameter to comma-separated list of strings, to specify only those lib cell instances that shall be considered for violation to be flagged.
- [*ignore_latch_cell*](#): The default value is ' '. Set this parameter to comma-separated list of strings, to specify lib cell instances that need to be ignored.
- [*ignoreRealLatch*](#): The default value is no. Set the value of the parameter to yes ignore real latches.
- [*reportLibLatch*](#): The default value is no. Set the value of the parameter to yes to report library latches from both `.sglib` and `.lib` libraries.
- [*reportalwayslatch*](#): The default value is no. Set the value of the parameter to yes to report `always_latch`.
- [*strict*](#): The default value is no. Set the value of the parameter to yes or Tag name to not to report violation for a latch whose input data pin is tied to a constant.
- [*reportHangingLatch*](#): The default value is no. Set the value of the parameter to yes to report violation for hanging latches.

- [reportLatchHierarchy](#): The default value is `no` and the InferLatch rule does not report the hierarchy in the violation message. Set the value of the parameter to `yes` to report violations with hierarchical information.
- [ignoreLatchHierarchy](#): The default value is `no` and the rule reports for all latch instance in every hierarchy. You can also specify a list of latch lib cells names and module names (regular expressions are supported) for which the rule should not report violations.

NOTE: Do not specify the `ignore_latch_cell` and `allow_latch_cell` parameters together. When used, the value of the `allow_latch_cell` parameter is discarded.

Messages and Suggested Fix

The following message appears at the location where a latch has been inferred for the signal `<sig-name>` in module `<mod-name>`:

[Tag] Latch inferred for signal '`<sig-name>`' in module '`<mod-name>`'

Potential Issues

Except where explicitly intended, latch inference is usually caused by a coding oversight like failure to define the default value for a variable defined in a conditional statement.

Consequences of Not Fixing

Unintended latches may lead to incorrect behavior. Also, inferred latches in a register-based design may cause problems for the VC SpyGlass Lint DFT tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the instance where the output is assigned a value inside the `always` block. Perform one of the following steps to verify whether the `always` block is used as a latch:

- Visual inspection of the `always` block
- Scroll up the HDL window, till you reach the `always` block
- If the `always` block is a very long block, view the HDL in an editor, and search backwards for the nearest `always` block

- Search for the instances where a variable is assigned under an `if` condition and not assigned under an `else` condition or an `if` condition has a missing `else` condition
- Search for the instances where a variable is assigned under a `case` condition and not assigned under a `default` clause or the `default` clause is missing

To fix the violation, add a final `else` clause to the `if` statement or a `default` clause to the `case` statement which infers the latch.

Example Code and/or Schematic

Verilog

Consider the following example where a latch is inferred for signal `out`:

```
module test(out, in, en);
    output [1:0] out;
    input  [1:0] in, en;
    reg     [1:0] out;

    always @ (en or in)
        case (en)
            2'b00 : out[0] <= 1'b1;
            2'b01 : out[0] <= 1'b0;
            2'b10 : out[0] <= in[1];
        endcase
endmodule
```

In the above example, the *InferLatch* Tag reports a violation as a latch is inferred in the design.

VHDL

Consider the following example where a latch is inferred for signal `q`:

```
library ieee;
use ieee.std_logic_1164.all

process (rst, d)
begin
    if (rst = '1') then
```

[OpenMore Tags](#)

```
    q <= d;  
end if;  
end process;
```

For this example, VC SpyGlass Lint generates the following message:

Latch inferred for signal 'q'

In case, a latch was intended, the above example is correct. Otherwise, the example needs to be rewritten to avoid latch inference.

Default Severity Label

Tag

Tag Group

Coding for Synthesis

Reports and Related Files

No related reports or files

Related Standard

Section 5.5.2 of RMM

IntClock

Internally generated clock detected

Language

Verilog, VHDL

Description

The IntClock Tag flags internally generated clocks in the design.

Internally generated clocks can have the following issues:

- Timing of internally generated clocks may be tightly dependent on target technology and therefore may have reduced portability.
- Internally generated clocks create significant problems for VC SpyGlass Lint DFT tools.
- It is difficult to put design constraints during synthesis for design with internally generated clocks.

NOTE: *Hanging nets are not considered internally generated. Only outputs of a flip-flop, combinational logic, or a black box are considered internally generated.*

The IntClock Tag will not report violation if the clock comes from fan-in/fan-out of black box as shown below:

```
wire a,w1;  
blackbox bb1(a,w1);  
always@(posedge w1)
```

Related Standard

Section 5.4.4 of RMM

Message Details

The following message appears at the location where an internally generated clock `<clk-name>` of flip-flop `<flop-name>` is encountered:

Internally generated clock '`<clk-name>`' (flop: `<flop-name>`) detected

Tag Severity

Guideline

Suggested Fix

No fix may be necessary, but review all generated clocks to make sure they are where you expected. You should also consider whether it would be better to have such clocks generated externally by the user, rather than in the block.

Examples (Verilog)

Consider the following example where the clock `seqClk` is internally generated:

```
module test(in1, in2, out);
    input in1;
    input [63:0] in2;
    output [63:0] out;

    reg    [63:0] out;
    wire intClk;
    reg  seqClk;

    always@(posedge intClk)
        seqClk = in1;

    always@(posedge seqClk)
        out = in2;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Internally generated clock 'seqClk' (flop: test.out) detected

Examples (VHDL)

Consider the following example where gated clock `ck1` of flip-flop output is not generated in a separate design unit:

```
library ieee;
use ieee.std_logic_1164.all

entity test is
    port( d1, d2, ck : in bit; output : out bit );
```

```
end test;

architecture rtl of test is
  signal ck1: bit;
begin
  process( ck )
  begin
    if( ck'event and ck = '1' ) then
      ck1 <= d1;
    end if;
  end process;

  process( ck1 )
  begin
    if( ck1'event and ck1 = '1' ) then
      output <= d2;
    end if;
  end process;
end rtl;
```

IntReset

Detects internally generated resets

When to Use

Use this Tag to detect internally generated resets.

Description

The *IntReset* Tag reports internally generated asynchronous resets in the design.

NOTE: *Hanging nets are not considered internally generated. Only outputs of a flip-flop or combinational logic (coming from flip-flop output) are considered internally generated.*

Tag Exceptions

Following are some of the exceptions to the IntReset Tag:

- The *IntReset* Tag supports block path analysis, that is, it supports constant value propagation. Therefore, this Tag does not traverse on the blocked path.
- The IntReset Tag will not report violation if reset comes from fan-in/fan-out of black box with no assume path specified.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

- [*ignore_hanging_flop*](#): Default value is no. When you set this parameter to yes, the Tag does not report a violation for the flip-flop if it's output is hanging

Constraint(s):

- [*set_case_analysis*](#) (Optional): Use this constraint to set the initial values for block path analysis.

- [assume_path](#) (optional): Use this constraint to enable the Tag to traverse further in case of black boxes.

Messages and Suggested Fix

The following message appears at the first line of a module where an internally generated asynchronous reset signal `<rst-name>` of flip-flop `<flop-name>` is encountered:

[GUIDELINE] Internally generated asynchronous `<type>` '`<rst-name>`' (flop: `<flop-name>`) detected

Where, `<type>` can be Set or Reset.

Potential Issues

The IntReset Tag reports a violation for internally generated asynchronous resets in the design.

Consequences of Not Fixing

Internally generated resets can cause following design problems:

- Timing of internally generated resets may be tightly dependent on target technology and therefore may have reduced portability.
- Internally generated resets create significant problems for VC SpyGlass Lint DFT tools.

It is difficult to put design constraints during synthesis for design with internally generated resets.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the instance in the code where the reset signal is internally generated. View the respective Incremental Schematic. The Incremental Schematic highlights the internally generated asynchronous reset. Right click on the highlighted reset signal and click Cross-Reference to RTL. This displays the line where the signal is used as reset.

It is recommended that you avoid internal generation, if possible. Else, ensure that you provide a testmode bypass, which can be checked with the VC SpyGlass Lint DFT product.

Example Code and/or Schematic

Verilog

Example 1

Consider the following example where the asynchronous reset signal `asynReset` is internally generated:

```
module top(clock,in1,in2,out);
    input in1;
    input [7:0] in2;
    input clock;
    output [7:0] out;
    reg [7:0] out;
    reg asynReset;
    always@(posedge clock)
        asynReset = in1;
    always@(posedge clock or posedge asynReset)
        if(asynReset == 1'b1)
            out = 8'b00000000;
        else
            out = in2;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:
Internally generated asynchronous reset 'asynReset' (flop: test.out) detected

Example 2

Consider the following example:

```
wire w1;
blackbox bbl(a,w1);
always@(posedge clk or posedge w1)
```

In the above example, the Tag does not report a violation.

VHDL

Consider the following example:

```
library ieee;
```

```
use ieee.std_logic_1164.all

entity test is
  port(
    ck                : in std_logic;
    asynReset,data    : in std_logic;
    output            : out std_logic
  );
end test;

architecture struct of test is
  signal asynReset1_temp , asynReset1: std_logic;
  begin
    -- Asynchronous reset signal derived from
    -- inferred DFF followed by AND gate
    process( ck )
    begin
      if( ck'event and ck = '1' ) then
        asynReset1_temp <= asynReset;
      end if;
    end process;

    asynReset1 <= asynReset1_temp and '1';
    --Asynchronous reset signal driving sequential element
    process( ck , asynReset1 )
    begin
      if( not asynReset1='1' ) then
        output<='0';
      elsif rising_edge(ck) then
        output<=data;
      end if;
    end process;
  end struct;
```

Default Severity

Guidelines

[OpenMore Tags](#)

Tag Group

Clocks and Reset Guidelines

Reports and Related Files

No related reports or files

Related Standard

Section 5.4.6 of RMM

NoMixedSynch

Do not mix synchronous and asynchronous logic in one module

Language

Verilog, VHDL

Description

The NoMixedSynch Tag flags modules that have both synchronous and asynchronous resets.

Since asynchronous logic is inherently more problematic than synchronous logic, it must be examined much more carefully. Localizing asynchronous logic makes careful inspection much easier.

Related Standard

Section 5.6.4 of RMM

Message Details

The following message appears at the first line of a module that has synchronous and asynchronous resets:

```
Do not mix synchronous and asynchronous logic in one module
```

Tag Severity

Guideline

Suggested Fix

Split the block into modules each of which uses synchronous reset or asynchronous reset alone.

Examples (Verilog)

Consider the following example where both synchronous logic and asynchronous logic are included in the same module:

```
module test(in1, res, clk, out1);  
  input [1:0] in1;  
  input res, clk;
```



```

output [1:0] out1;
reg [1:0] out1;
always @(negedge res or posedge clk)
begin
    if(!res)
        out1[0] = 1'b0;
    else
        out1[0] = in1[0];
    end
    always @(posedge clk)
begin
    out1[1] = in1[1];
end
endmodule

```

Examples (VHDL)

Consider the following example that has both asynchronous logic and synchronous logic in the same architecture:

```

library ieee;
use ieee.std_logic_1164.all

architecture rtl of test is
begin
    AsyncReset : process(clk, reset)
    begin
        if ( reset = '1' ) then
            q1 <= '0';
        elsif( clk'event and clk = '1' ) then
            q1 <= data;
        end if;
    end process AsyncReset;

    SyncReset : process(clk)
    begin
        if( clk'event and clk = '1') then
            if ( reset = '1' ) then
                q2 <= '0';
            end if;
        end if;
    end process SyncReset;
end architecture rtl;

```

```
        else
            q2 <= data;
        end if;
    end if;
end process SyncReset;
end rtl;
```

[OpenMore Tags](#)

NonBlockAssign

Do not use blocking assignments in sequential blocks

Language

Verilog

Description

The NonBlockAssign Tag flags blocking assignments used in sequential `always` constructs.

The NonBlockAssign Tag also flags blocking assignments in combinational `always` constructs that describe a latch.

Use of blocking assignments in sequential `always` constructs may cause RTL simulation and post-synthesis simulation results to differ.

A blocking assignment must complete before the next behavioral step (that is, another assignment) is evaluated. However, assignments in a sequential `always` construct are likely be synthesized as parallel logic which evaluates simultaneously and thus differs in behavior from the RTL logic.

Related Standard

Section 5.5.6 of RMM

Message Details

The following message appears at the location where a blocking assignment is encountered in a sequential `always` constructs:

Blocking assignment detected inside a sequential block

Tag Severity

Tag

Suggested Fix

Use only nonblocking assignments in sequential blocks.

Examples (Verilog)

Consider the following example:

```
module NonBlockAssign_mod(out, in1, in2, clk);  
    output out;  
    input in1, in2, clk;  
    reg out;  
    always @ (posedge clk)  
        out = #2 in1 | in2;  
endmodule
```

RegOutputs

Some output ports from a top module are not registered

When to Use

Use this Tag to report unregistered output from a top module./ Use this Tag to report output ports in a module that are not driven by a flip-flop.

Description

The *RegOutputs* Tag reports all output ports in a module that are not driven by a flip-flop.

The Tag allows any number of buffers and up to one inverter between the register and the output port. Other logic in that path will cause the Tag to be violated.

NOTE: *The Tag does not report a violation for output ports that are driven by clock inputs.*

Prerequisites

Run the prerequisite Tag, *Prereqs_RegOutputs*, to consider block pragmas for Tag checking.

Tag Exception

By default, the *RegOutputs* Tag will be checked only on the modules, which have block pragmas specified in following format:

■ For VHDL:

```
-- VC SpyGlass Lint block
entity ...
```

■ For Verilog:

```
// VC SpyGlass Lint block
module ...
```

If none of the module has block pragma specified, the *RegOutputs* Tag will be checked only for the top-level modules. No violation will be reported for the output ports connected to input or inout ports through feedthrough logic.

Also, the *RegOutputs* Tag ignores isolation and level shifter cells present between a flip-flop and a port.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

- [*lib_sync_cells*](#): The default value is NULL. Set the value of the parameter to comma-separated list of cell names to not report violations for the output ports passing through the lib cells.
- [*reportundrivenout*](#): The default value is yes. Set the value of the parameter to no to disable the violations for the undriven or unconnected ports. In this case, the Tag also ignores the unconnected/undriven output ports connected through buffers and inverters.
- [*reportlatchedoutput*](#): The default value is no. Set the value of the parameter to yes to report latched output ports.
- [*ignore_cells*](#): By default this parameter is not set. Set the `ignore_cells` parameter to comma separated list of cell names to allow the Tag to ignore these cells.
- [*check_regoutput_modules*](#): By default this parameter is not set. Set the `check_regoutput_modules` parameter to module names or PERL regular expression to specify the modules that should be included for the Tag checking.
- [*ignore_regoutput_modules*](#): By default this parameter is not set. Set the `check_regoutput_modules` parameter to module names or PERL regular expression to specify the modules that should be excluded from the Tag checking.
- [*report_regoutputs_ports*](#): By default this parameter is set to {}. Specify a comma-separated list of ports on which *RegOutputs* tag needs to be run. The parameter also supports regex.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a signal that is not a register output, is assigned to an output port `<port-name>`:

[GUIDELINE] Port '`<port-name>`' is not driven by a register

Potential Issues

Violations can arise in situations when your design has an unregistered output port.

Consequences of Not Fixing

If a block is intended to be reusable, it must have a predictable timing behavior. Outputs that are not registered have timing behavior dependent on internal delays that vary depending on target technology and process variations.

Registering all outputs assures that output timing is controlled by the clock, rather than by these potentially unpredictable parameters.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the offending output port and its driver.

To fix the violation, ensure that all the output signals are registered immediately before the module output port.

Example Code and/or Schematic

Example 1 (Verilog)

Consider the following example:

```
module top1N(clk, in, out);
    input clk;
    input  [3:0] in;
    output [4:0] out;

    reg [4:0] out;

    always @ (posedge clk)
        out [3:0] = in;
    wire [4:0] temp;
    non_top1N1 nt1(clk, in, temp);
```

```

endmodule

module non_top1N1(clk, in, out);
    input clk;
    input  [3:0] in;
    output [4:0] out;

    reg      [4:0] out;

    always @ (in)
        out [4:2] = in [2:0];
endmodule

```

In the above example, the *RegOutputs* Tag reports violation for the out port as it is not driven by a register.

Example 2 (Verilog)

Consider the following example:

```

// VC SpyGlass Lint block
module top(input d, input clk, output out1);
    wire w3,w4,w5;
    reg w1_out;

    // Flop 1
    always @(posedge clk)
        w1_out <= d;

    slave inst(w1_out,w5,w3,w4);
    assign out1 = w3;
endmodule

// VC SpyGlass Lint block
module slave(input w1, input d2, output o1, output o2);
    reg o2;
    wire temp, temp2;

    assign temp = w1 & temp2;
    assign o1 = temp;

```



```
// Flop 2
always @(posedge temp)
  o2 <= d2;
endmodule
```

In this example, the *RegOutputs* Tag does not report a violation because signal out1 is driven by the clock of flop 2.

Example 3 (VHDL)

Consider the following example where outputs out1 and out2 are driven by flip-flops but outputs out3 and out4 are not driven by flip-flops:

```
library ieee;
use ieee.std_logic_1164.all

architecture rtl of test is
begin
  process
  begin
    wait until clk'event and clk='1';
    out1 <= in1;
    out2 <= in2;
  end process;
  process (in1,in2)
  begin
    out3 <= in2;
    out4 <= out1(0) and out2;
  end process;
end rtl;
```

For this example, VC SpyGlass Lint generates the following messages:

Port 'out3' is not driven by a register

Port 'out4' is not driven by a register

Default Severity Label

Guideline

Tag Group

Partitioning for Synthesis

Report and Related Files

No related reports or files.

Related Standard

Section 5.6.1 of RMM

ResetName

Reset name does not follow the naming convention.

Language

Verilog, VHDL

Description

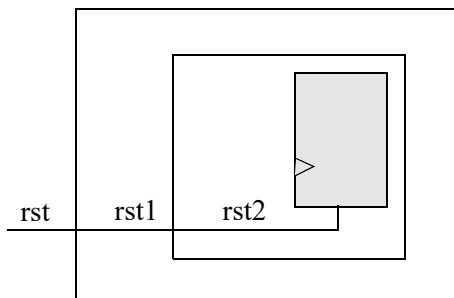
The ResetName Tag flags those reset type signal names that do not follow the recommended naming convention.

For all reset signals, this Tag checks the name of synchronous or asynchronous reset signals at the point of use only.

By default, the ResetName Tag flags those reset type signal names that do not start with `rst`. Use the [resetname](#) Tag parameter to specify your own naming convention.

All RTL nets from the source of the asynchronous reset signal to the location where the asynchronous reset signal is used in a flip-flop description are checked, including intermediate nets connecting across hierarchical boundaries.

For example, the ResetName Tag checks `rst` (source), `rst1` (intermediate), and `rst2` (point-of-use) in the hierarchical design shown in the following figure:



For synchronous reset signals, the ResetName Tag checks the name of the synchronous reset signal at the point-of-use only. Thus, the ResetName Tag checks the name `rst2` (point-of-use) only if the reset signal `rst2` is a synchronous reset signal.

Following this naming convention improves the readability of the design

and helps in debugging.

The ResetName Tag is case-sensitive.

Related Standard

Section 5.2.1 of RMM

Message Details

With the default name check

The following message appears at the location where an asynchronous set/reset signal `<signal-name>` is declared whose name does not start with `rst`:

```
<type> signal name '<signal-name>' should start with rst
```

Where `<type>` can be Set or Reset.

With user-defined name check

If you have specified your own regular expression using the [resetname](#) Tag parameter, then the following message appears at the location whose name does not match the regular expression specified with the `resetname` Tag parameter:

```
<type> signal name '<signal-name>' should conform to regular  
expression $resetname
```

Where `<type>` can be Set or Reset.

Tag Severity

Guideline

Suggested Fix

Modify the reset signal name to conform with the convention. Make sure you follow this name change through the hierarchy. If you do not, the SigHierName Tag will trigger.

Examples (Verilog)

Consider the following example that explains how VC SpyGlass Lint flags Tag-violating reset signal names from source to point-of-use:

```
module test(in1, in2, in3, clk, res1, res2,
            out1, out2);
    input in1, in2, in3, clk, res1, res2;
    output out1, out2;

    h1 ins1(in1, in2, res1, clk, out1);
    h2 ins2(in3, clk, res2, out2);
endmodule

module h1(in1_h1, in2_h1, res_h1, clk_h1, out_h1);
    input in1_h1, in2_h1, res_h1, clk_h1;
    output out_h1;

    reg out_h1;

    always @(posedge clk_h1 or negedge res_h1)
        if(res_h1 == 1'b0)
            out_h1 = 1'b0;
        else
            out_h1 = in1_h1 | in2_h1;
endmodule

module h2(in_h2, res_h2, clk_h2, out_h2);
    input in_h2, res_h2, clk_h2;
    output out_h2;

    reg out_h2;
    wire intRst_h2;

    assign intRst_h2 = res_h2;

    always @ (posedge clk_h2 or negedge intRst_h2)
        if(intRst_h2 ==1'b0)
            out_h2 = 1'b0;
        else
            out_h2 = in_h2;
endmodule
```

The module `h1` has a flip-flop description `out_h1` that has a reset signal named `res_h1`. As the name `res_h1` does not start with `rst`, VC SpyGlass Lint generates the following message at the start of the `always` construct:

```
Reset signal name 'res_h1' should start with rst
```

The module `test` has an instant `ins1` of module `h1` that has the signal `res1` as the reset signal argument. As the name `res1` does not start with `rst`, VC SpyGlass Lint generates the following message at the location of the instantiation:

```
Reset signal name 'res1' should start with rst
```

The module `h2` has a flip-flop description `out_h2` that has a reset signal named `intRst_h2`. As the name `intRst_h2` does not start with `rst`, VC SpyGlass Lint generates the following message at the start of the `always` construct:

```
Reset signal name 'intRst_h2' should start with rst
```

The reset signal `intRst_h2` is being assigned the value of signal `res_h2`. Hence, VC SpyGlass Lint generates the following message at the location of assignment:

```
Reset signal name 'res_h2' should start with rst
```

The module `test` has an instant `ins2` of module `h2` that has the signal `res2` as the reset signal argument. As the name `res2` does not start with `rst`, VC SpyGlass Lint generates the following message at the location of the instantiation:

```
Reset signal name 'res2' should start with rst
```

Examples (VHDL)

Consider the following example where the reset signal `r_s` does not follow the recommended naming convention:

```
library ieee;
use ieee.std_logic_1164.all

process(clk, r_s)
begin
    if clk'event and clk ='1' then
```

[OpenMore Tags](#)

```
    if r_s(1) ='1' then
        q <= '0';
    else
        q <= d;
    end if;
end if;
end process;
```

For this example, VC SpyGlass Lint generates the following message:

Reset signal name 'r_s' should start with rst

UseMuxBusses

Detect tristate nets

When to Use

Use this Tag to detect tristate signals in the design.

Description

The *UseMuxBusses* Tag reports violation for all the tristate signals found in the design. These tristates can result from Verilog/VHDL assignment to Z, Verilog primitive instances like `bufif0/bufif1/notif0/notif1`, and instances of tristate cells from `.lib` definitions.

Tag Exceptions

The *UseMuxBusses* Tag ignores tristates that do not drive any logic, and hence, do not contribute to the final netlist.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

CHECK_Z_ASSIGN: By default, this parameter is set to 0. This indicates that the tag will not report violations on Z assignment cases.

If you set this parameter to 1, the tag will report violations on Z assignment cases.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where the tristate signal `<sig-name>` is defined:

[GUIDELINE] Tristate Signal '`<sig-name>`' used. Use mux-based busses wherever possible

Potential Issues

Violation can arise in situations when your design has tristate signals.

Consequences of Not Fixing

Tristate nets cause problems for VC SpyGlass Lint DFT and require special attention to ensure the following:

- Tristate nets are driven under all circumstances
- Tristate nets are never driven by multiple drivers simultaneously to avoid multiple-driver situation on the tristate bus.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the instance where tristate signal is used. Alternatively, you can also view the Incremental schematic. The Incremental Schematic highlights the tristate signal .

To fix the violation, replace tristate nets control with mux net control. Mux-based connection is a much safer strategy, wherever feasible.

Example Code and/or Schematic**Verilog**

Consider the following example that has a tristate signal out:

```
module test (in1, in2, en1, en2, out);
  input [7:0] in1, in2;
  input en1, en2;
  output [3:0] out;

  reg [3:0] out;
  always @ (en1 or en2 or in1 or in2)
  begin
    out = en1?in1[3:0]:4'bz;
    out = en2?in2[6:3]:4'bz;
  end
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Tristate Signal 'out[3:0]' is used. Use mux-based busses wherever possible

VHDL

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

process(en, in1)
begin
  case en is
    when '0' =>
      out1 <= in1;
      out2_z <= '1';
    when '1' =>
      out1 <= 'Z';
      out2_z <= in1;
    when others =>
      out2_z <= 'Z';
  end case;
  out3 <= 'Z';
end process;
```

The signal out2_z is explicitly assigned a value of Z in a CASE construct controlled by an enable signal (en).

For this example, VC SpyGlass Lint generates the following message:

Tristate Signal 'out2_z' used. Use mux-based busses wherever possible

Default Severity Label

Guideline

Tag Group

System Level Design

Report and Related File(s)

No related reports/ or files.

[OpenMore Tags](#)

NoVar

Use Signals instead of variables to avoid simulation/synthesis mismatch

Language

VHDL

Description

The NoVar Tag flags variables used in the design.

It is recommended to use signals instead of variables so that pre- and post-synthesis simulation results match.

Related Standard

Section 5.5.7 of RMM

Message Details

The following message appears at the location where a variable declaration `<var-name>` is found:

Use signal declaration instead of variable declaration for '`<var-name>`' for synthesizable RTL

Tag Severity

Guideline

Suggested Fix

Convert variables to `std_logic` or `std_logic_vector` types wherever possible.

Examples

Consider the following example that uses a variable d:

```
architecture rtl of test is
  signal a, b, c : STD_LOGIC;
begin
  process
```

```
variable d : STD_LOGIC;  
begin  
    c <= a and b;  
    d := a and b;  
    wait;  
end process;  
end rtl;
```

SpyGlass generates the following message:

Use signal declaration instead of variable declaration for 'd'
for synthesizable RTL

To fix this message, rewrite the example as follows:

```
architecture rtl of test is  
    signal a, b, c : STD_LOGIC;  
begin  
    process  
        signal d : STD_LOGIC;  
    begin  
        c <= a and b;  
        d := a and b;  
        wait;  
    end process;  
end rtl;
```

[OpenMore Tags](#)

SepFSMLogic

Finite-state machine (FSM) and non-FSM logic is mixed in one module

Language

Verilog, VHDL

Description

The SepFSMLogic flags the non-FSM logic that have been included with the finite-state machine (FSM) logic in the same module.

Mixing different types of logic in one module makes it difficult to create a single effective synthesis strategy for that module.

Related Standard

Section 5.5.9 of RMM

Message Details

Verilog

The following message appears at the first line of a non-FSM logic that is included in the same module as finite-state machine (FSM) logic:

NonFSM logic appears with FSM in the same module

VHDL

The following message appears at the first line of a non-FSM logic *<process-name>* that is included in the same architecture as finite-state machine (FSM) logic:

Non-FSM process '*<process-name>*' appears with FSM modules in the same architecture

Tag Severity

Guideline

Suggested Fix

Separate the finite-state machine into a different design unit.

Examples (Verilog)

Consider the following example where non-FSM logic appears with finite-state machine (FSM) in the same module:

```
module SepLogic_mod(clk, reset, exin0, exin1, FSMOUT,
exout0);
    input          clk, reset, exin0, exin1;
    output [2:0] FSMOUT;
    output          exout0;
    reg [2:0] FSMOUT;
    reg [1:0] C_STATE, N_STATE;
    parameter ST0 = 2'b00;
    parameter ST1 = 2'b01;
    parameter ST2 = 2'b11;
    parameter ST3 = 2'b10;
    // and gate - logic unrelated to FSM.
    assign exout0 = exin0 & exin1;
    ...
endmodule
```

Examples (VHDL)

Consider the following example architecture rtl that contains both finite-state machine (FSM) logic (processes fsm_seq and fsm_com) and non-FSM logic (process extr):

```
library ieee;
use ieee.std_logic_1164.all

architecture rtl of test is
    <signal definitions>
begin
    fsm_seq : process (<port_list>)
        <FSM's sequential logic>
    end process fsm_seq;

    fsm_com: process (<port-list>)
        <FSM's combinational logic>
    end process fsm_com;
```

[OpenMore Tags](#)

```
    extr: process (<port-list>)  
        <Non-FSM logic>  
    end process extr;  
end rtl;
```

SpyGlass generates the following message for non-FSM process `extr`:

Non-FSM process 'extr' appears with FSM modules in the same architecture

To fix this message, separate the finite-state machine (FSM) and non-FSM logic in separate architecture blocks.

SepStateMachine

Combinational and sequential parts of a finite-state machine are combined in one process

Language

Verilog, VHDL

Description

The SepStateMachine Tag flags the combinational part of a finite-state machine (FSM), if it is included with the sequential part of the finite-state machine in the same edge-triggered (sequential) `always/process` construct.

Separating the two parts of an finite-state machine in different constructs enhances debug. The [InferLatch](#) Tag can then catch un-initialized variables in the `case` statement logic. If the two parts of the finite-state machine were combined, the un-initialized variables would infer registers and it would be difficult to detect them as errors as you expect to infer registers in the finite-state machine.

Related Standard

Section 5.5.9 of RMM

Message Details

The following message appears at the start of the combinational part of a finite-state machine design when the combinational part is included with the sequential part in the same construct:

```
State Machine should be separated into processes, one for
combinational and one for the sequential logic
```

Tag Severity

Guideline

Suggested Fix

Compute a next state variable in the combinational block and update the state vector with the next state in a separate sequential block.

Examples (Verilog)

Consider the following example:

```
module test(clk, reset, FSMOUT);
    input clk;
    input reset;
    output [2:0] FSMOUT;

    reg [2:0] FSMOUT;
    reg [1:0] C_STATE, N_STATE;
    parameter ST0 = 2'b00;
    parameter ST1 = 2'b01;
    parameter ST2 = 2'b11;
    parameter ST3 = 2'b10;

    // sequential current-outputs,next-state block
    always @(posedge clk or posedge reset)
    begin
        if (reset)
            C_STATE <= ST0;
        else
            C_STATE <= N_STATE;

            case(C_STATE)
                ST0 : begin
                    FSMOUT = 3'b111;
                    N_STATE = ST1;
                end
                ST1 : begin
                    FSMOUT = 3'b101;
                    N_STATE = ST2;
                end
                ST2 : begin
                    FSMOUT = 3'b110;
                    N_STATE = ST3;
                end
                ST3 : begin
                    FSMOUT = 3'b011;

```

```

        N_STATE = ST0;
    end
    default : begin
        FSMOUT = 3'b000;
        N_STATE = ST0;
    end
endcase
end
endmodule

```

Examples (VHDL)

Consider the following example architecture that has the Finite-state machine (FSM) combinational logic (CASE construct) and the FSM sequential logic (IF-THEN-ELSE construct) are in the same process block (seq):

```

library ieee;
use ieee.std_logic_1164.all

architecture rtl of test
    signal c_data, n_data : data_t;
    signal c_state, n_state : state_t;
begin
    seq: process (clk, reset)
    begin
        if reset = '0' then
            c_data <= (others => '0');
            c_state <= i_state;
        elsif clk'event and clk = '1' then
            case c_state is
                when i_state => c_data <= data;
                    c_state <= state_1;
                when state_1 => c_data <= c_data or data;
                    c_state <= state_2;
                when state_2 => c_data <= c_data and data;
                    c_state <= state_3;
                when state_3 => c_data <= c_data xor data;
                    c_state <= i_state;
            end case;
        end if;
    end process;
end architecture;

```

```

        when others => c_data <= (others => '0');
        c_state <= i_state;
    end case;
end if;
end process
end rtl;

```

For this example, SpyGlass generates the SepStateMachine Tag message for the CASE construct.

To fix this message, you can rewrite the above example to separate the combinational and sequential logic as follows:

```

architecture rtl of test is
    signal c_data, n_data : data_t;
    signal c_state, n_state : state_t;

begin
    seq: process (clk, reset)
    begin
        if rest = '0' then
            c_data <= (others => '0');
            c_state <= i_state;
        elsif clk'event and clk = '1' then
            c_state <= n_state;
            c_data <= n_data;
        end if;
    end process;

    com: process (c_state, c_data, data)
    begin
        case c_state is
            when i_state => c_data <= data;
            c_state <= state_1;
            when state_1 => c_data <= c_data or data;
            c_state <= state_2;
            when state_2 => c_data <= c_data and data;
            c_state <= state_3;
            when state_3 => c_data <= c_data xor data;
            c_state <= i_state;
        end case;
    end process;
end rtl;

```

```
        when others => c_data <= (others => '0');  
            c_state <= i_state;  
        end case;  
    end process;  
end rtl;
```

[OpenMore Tags](#)

ArchHdr

Standard header not found before architecture declaration

Language

VHDL

Description

The ArchHdr tag flags the architecture headers that do not conform to the recommended structure.

By default, SpyGlass checks for the architecture header in the following format:

```

-----
-- architecture name: <architecture name>
-- associated entity: <entity name>
-- purpose: <your comments>
-- Release History: <your comments>
-- VERSION      Date      AUTHOR      DESCRIPTION
-- <version>    <date>    <name>      <comments>
-- <version>    <date>    <name>      <comments>
-----

```

The exact format of the architecture headers should be adjusted to company standards.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears for each architecture header line that does not conform to the default architecture header format:

Non-standard architecture header: <error-type>

Where *<error-type>* is the type of error in the line.

tag Severity

tag

Suggested Fix

Determine and follow the local approved standard.

[OpenMore Tags](#)

ArchName

Architecture does not follow recommended naming convention

Language

VHDL

Description

The ArchName tag flags architecture names that do not follow the recommended naming conventions.

The ArchName tag requires that the architecture names should follow the following default naming convention:

- Name should be between 1 to 16 characters long
- Name should not have two successive underscores
- Name should be selected as follows:

Synthesis Models	Simulation Models
<code>rtl</code> (for RTL models)	<code>sim</code> (for behavioral models)
<code>str</code> (for structural models)	<code>tb</code> (for testbenches)

Use the [archname](#) tag parameter to specify your own architecture naming convention.

Following the recommended architecture naming convention helps is quickly identifying the type of the architectures.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the first line of an architecture *<arch-name>* that does not have a name as per the recommended list:

Architecture Name *<arch-name>* does not follow naming convention.

tag Severity

Guideline

Suggested Fix

Modify all references to this name to a form that meets the convention.

Examples

Consider the following example where the architecture name `myarch` does not follow the naming convention:

```
ENTITY myentity IS
  PORT (
    clock1 : in std_logic;
    clk2 : in std_logic;
    sig1 : IN std_logic;
    sig2 : IN std_logic;
    sig3 : OUT std_logic
  );
END myentity;

ARCHITECTURE myarch OF myentity IS
  BEGIN
    sig3 <= sig1 AND sig2;
END myarch;

ARCHITECTURE rtl OF myentity IS
  BEGIN
    sig3 <= sig1 AND sig2;
END rtl;

ARCHITECTURE str OF myentity IS
  BEGIN
    sig3 <= sig1 AND sig2;
END str;

ARCHITECTURE sim OF myentity IS
  BEGIN
```

[OpenMore Tags](#)

```
        sig3 <= sig1 AND sig2;
    END sim;

    ARCHITECTURE tb OF myentity IS
        BEGIN
            sig3 <= sig1 AND sig2;
        END tb;
```

AsyncName

Asynchronous signals do not follow naming convention

Language

Verilog, VHDL

Description

The AsyncName tag flags those asynchronous signal names that do not follow the recommended naming convention.

By default, the AsyncName tag flags those asynchronous signal names that do not end in `_a`. Use the [*asyncname*](#) tag parameter to set your own regular expression for checking the asynchronous signal names.

If a candidate for asynchronous signal is being assigned from another signal, the other signal is also a candidate for asynchronous signal.

NOTE: *The AsyncName tag currently checks asynchronous reset signal names only.*

Related Standard

Section 5.2.1 of RMM

Message Details

With the default name check

The following message appears at the location where a tag-violating asynchronous reset signal name `<sig-name>` is encountered:

Asynchronous `<type>` signal name '`<sig-name>`' should end with `_a`

Where `<type>` can be Set or Reset.

With user-defined name check

If you have specified your own regular expression using the [*asyncname*](#) tag parameter, then the following message appears at the location where a tag-violating asynchronous reset signal name `<sig-name>` is encountered:

Asynchronous `<type>` signal name '`<sig-name>`' should conform to the regular expression `$asyncname`

Where `<type>` can be Set or Reset.

[OpenMore Tags](#)

tag Severity

Guideline

Suggested Fix

Change signal names for truly asynchronous signals to conform with the convention. It is advisable for even those asynchronous signals that are not detected by SpyGlass.

Examples (Verilog)

Consider the following example where asynchronous reset signal name `rst` does not end with `_a`:

```

module AsyncName(Q, QB, X, rst, clk);
    input X;
    input rst, clk;
    output Q, QB;
    reg Q, QB;
    always @(posedge clk or negedge rst)
    begin
        if (!rst)
            Q <= 0;
        else
            Q <= X;
    end

    always @(Q)
    begin
        QB = ~Q;
    end
endmodule

```

Examples (VHDL)

Consider the following example where asynchronous signal name `reset` does not end with `_a`:

```

library ieee;
use ieee.std_logic_1164.all

```

```
entity dff is
  port (d, clk, reset : in std_logic ;
        q : out std_logic) ;
end dff ;

architecture behav of dff is
begin
  process(clk, reset)
  begin
    if (reset = '0') then
      q <= '0' ;
    elsif(clk'event and clk = '1') then
      q <= d ;
    end if ;
  end process ;
end behav ;
```

[OpenMore Tags](#)

AvoidAsync

Asynchronous reset used

Language

Verilog, VHDL

Description

The AvoidAsync tag flags all asynchronous reset type signals at the point of use.

The AvoidAsync tag flags messages for all asynchronous reset signals from their point-of-use to their source. You can view the complete path in the Incremental Schematic Window by clicking the **IS** button.

If you must use asynchronous reset in your design, you should ensure that there is only one asynchronous reset signal, driven from the top-level of the design.

Further, you may add logic so that assertion of the reset is asynchronous but de-assertion is synchronous with the clock. Thus, such logic prevents problems caused when reset is released too close to the active edge of the clock.

By default, the *AvoidAsync* tag ignores violations for the instantiations of the module with asynchronous reset signal. Set the value of the [report_hier_path](#) parameter to yes to enables the *AvoidAsync* tag to report violations for all the instantiations of the module with asynchronous reset signal. The violation message contains hierarchical name of the reset signal.

Related Standard

Section 5.6.4 of RMM

Message Details

The following message appears at the first line of an always construct where an asynchronous set/reset signal *<sig-name>* is used:

```
Asynchronous <type> signal '<sig-name>' used at flop '<flip-flop-name>'
```

Where *<type>* can be Set or Reset.

Suggested Fix

Nothing to fix if your convention is async reset, otherwise modify the logic to conform to the locally dominant convention.

tag Severity

Guideline

Examples (Verilog)

Consider the following example where an asynchronous reset signal `rst` has been used:

```
module test (clk, rst, in, out);
  input clk, rst, in;
  output out;

  reg out;

  always @ (posedge clk or posedge rst)
  begin
    if (rst)
      out = 1'b0;
    else
      out = in;
    end
endmodule
```

For this example, SpyGlass generates the following message:

Asynchronous reset signal 'rst' used

Examples (VHDL)

Consider the following example of a non-hierarchical design having sequential element `q1` being reset by an asynchronous reset `rst`:

```
library ieee;
use ieee.std_logic_1164.all

process(clk, rst)
  begin
```

[OpenMore Tags](#)

```
if (rst = '1') then
    q1 <= '0';
elsif(clk'event and clk = '1') then
    q1 <= in1 or in2;
end if;
end process;
```

For this example, SpyGlass generates the following message:

Asynchronous reset signal 'rst' used

ClockDomain

More than one clock domain in a module

Language

Verilog, VHDL

Description

The ClockDomain tag flags multiple clock domains in the design.

SpyGlass considers the innermost clock signal as the first clock domain and then reports messages for each subsequent clock domain found.

The ClockDomain tag displays the source of each clock signal driving a domain if there is more than one domain in one module.

Related Standard

Section 3.2.3 of RMM

Message Details

The following message appears at the location where the output of a flip-flop *<flop2-name>* in clock domain *<clk2-name>* is assigned when a flip-flop *<flop1-name>* of a different clock domain *<clk1-name>* has already been found:

Multiple clock domains. First: *<clk1-name>* (flop: *<flop1-name>*, current: *<clk2-name>* (flop: *<flop2-name>*)

tag Severity

tag

Suggested Fix

Generally nothing to fix, unless you thought your design was purely synchronous.

Examples (Verilog)

Consider the following example that has two clock domains (clk and clk_gated):


```

module test(in1, in2, in3, ena, clk, out1, out2);
    input  in1, in2, in3, ena, clk;
    output out1, out2;

    reg     out1, out2;
    wire    clk_gated;

    assign clk_gated = clk & ena;

    always @(negedge clk_gated)
        begin
            out1 <= in1 & in2;
        end

    always @(posedge clk)
        begin
            out2 <= in2 | in3;
        end
endmodule

```

For this example, SpyGlass generates the following message:

Multiple clock domains. First: 'test.clk'(flop: test.out2),
Current: 'test.clk_gated'(flop: test.out1)

Examples (VHDL)

Consider the following example that has two clock domains (clk1 and s1) inferred in an architecture of entity ClkDomainTest:

```

library ieee;
use ieee.std_logic_1164.all

process(clk1,s1)
begin
    if clk1'event and clk1='1' then
        d <= a and b;
    end if;

    if s1'event and s1='1' then

```

```
        e <= b and c;  
    end if;  
end process;
```

For this example, SpyGlass generates the following message:

Multiple clock domains. First: 'ClkDomainTest.s1'(flop: ClkDomainTest.e),
Current: 'ClkDomainTest.clk1'(flop: ClkDomainTest.d)

ClockPhase

Both positive and negative edges of a clock are used in this design

Language

Verilog, VHDL

Description

The ClockPhase tag flags the modules that have both positive and negative edges of the same clock signal defining different sequential elements in the same design.

Using both edges creates a design where the correct functioning may depend on the clock duty cycle. Thus, such designs are less portable.

Related Standard

Section 5.4.1 of RMM

Parameters

- [*allow_combo_logic_base*](#): Default value is no. Set this parameter to yes to skip all VC SpyGlass Lint-generated combinational elements in the fan-in cone while finding the reset/preset source net.
- [*waiver_compat*](#): Default value is no. If you set the value of this parameter to yes or <tag-name>, it ensures that the tag does not generate the line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.

Message Details

The following message appears at the first line of a module that uses both positive and negative edges of the clock signal <clk-name> to define different sequential elements <flop1-name> and <flop2-name>:

Clock '<clk-name>' is driving on two different edges in the design at flops '<flop2-name>' (line <line-num>) and '<flop1-name>' (line <line-no>)

tag Severity

Guideline

Suggested Fix

Limit cases where this is necessary and clearly document the fact at least in the file header.

Examples (Verilog)

Consider the following example where both edges of the clock signal clock are used in the design at flip-flops out1 and out2:

```
module test(clock, in1, in2, out1, out2);
    input  clock;
    input  [63:0] in1, in2;
    output [63:0] out1, out2;
    reg     [63:0] out1, out2;

    always@(posedge clock)
        out1 = in1;

    always@(negedge clock)
        out2 = in2;
endmodule
```

Examples (VHDL)

Consider the following example where both positive and negative edges of the clock clk1 are used to define the different sequential elements output1 and output2 respectively:

```
library ieee;
use ieee.std_logic_1164.all

entity test is
    port(d1,d2,clk1: in bit; output1,output2 : out bit);
end test;

architecture rtl of test is
```

```
begin
  process( clk1 )
  begin
    if( clk1'event and clk1 = '1' ) then
      output1 <= d1;
    end if;
  end process;

  process( clk1 )
  begin
    if( clk1'event and clk1 = '0' ) then
      output2 <= d2;
    end if;
  end process;
end rtl;
```

For this example, SpyGlass generates the following message:

Clock 'clk1' is driving on two different edges in the design at
flops'test.output2'(line 19) and 'test.output1'(line 12)

DesgPack

Constants and function definitions defined in design file.

Language

VHDL

Description

The DesgPack tag flags parameter values and function definitions in the entity or architecture definitions.

It is recommended to collect parameter values and function definitions in a separate package file. Such arrangements make the design easier to maintain as all globals are placed together in one location.

Related Standard

Section 5.3.3 of RMM

Message Details

The following message appears at the location where an object declaration *<object-name>* of type *<object-type>* is found in an entity or an architecture:

<object-type> '*<object-name>*' should be defined in a separate package

where *<object-type>* can be Constant or SubProgram.

tag Severity

Guideline

Suggested Fix

Move constant and function definitions out of design files either into existing packages or into new packages.

Examples

Consider the following example where a constant CONS1 and a function func1 are defined in an architecture:

[OpenMore Tags](#)

```
architecture rtl of test is
  constant CONS1 : bit := '1';
  ...
  function func1 (
    ...
  end rtl;
```

For this example, SpyGlass generates the following messages:

Constant 'CONS1' should be defined in a separate package.

SubProgram 'func1' should be defined in a separate package.

EntHdr

Standard header not found before entity declaration

Language

VHDL

Description

The EntHdr tag flags the entity headers that do not conform to the recommended structure.

By default, SpyGlass checks for the entity header in the following format:

```
-----  
-- entity name: <entity name>  
-- purpose: <your comments>  
-- Release History: <your comments>  
-- VERSION      Date      AUTHOR      DESCRIPTION  
-- <version> <date>      <name>      <comments>  
-- <version> <date>      <name>      <comments>  
-----
```

The exact format of the entity headers should be adjusted to company standards.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears for each entity header line that does not conform to the default entity header format:

Non-standard entity header: <error-type>

Where <error-type> is the type of error in the line.

tag Severity

tag

[OpenMore Tags](#)

Suggested Fix

Determine and follow the local approved standard.

GateClockAtTop

Gated/generated clocks exist in the designs that are not all generated from one top-level block

Language

Verilog, VHDL

Description

The GateClockAtTop tag flags clocks that are gated or generated at levels other than the top-level.

It is recommended that the clocks are generated or gated in a separate module at top-level and then distributed to other modules. This clock generation module can have hierarchy. Such practice improves readability of the design.

The GateClockAtTop tag reports violation if any one or more condition described below occurs:

- Top-level module has clock and they are not ports
- Clocks are generated in more than one instance
- All clocks are generated in an instance, but all output ports of that instance are not clock ports.

Related Standard

Section 5.4.5 of RMM

Message Details

The following message appears at the location where a clock `<clk-name>` is encountered that is not generated in a separate top-level module:

```
Gated clock '<clk-name>' not generated in a separate module at top-level
```

tag Severity

Guideline

[OpenMore Tags](#)

Suggested Fix

Move all clock generation/gating to a special module dedicated to that purpose.

Examples (Verilog)

Consider the following example where the clock signal `outClock` is not generated in a separate module at top-level unlike other clock signal `clock` (generated in module `clkGen` at the top-level):

```

module test(in, out);
    input [10:0] in;
    output [3:0] out;

    reg    [3:0] out;
    wire   [3:0] clock;

    clkGen cg(in[5:0], clock[2:0]);
    mod    mod1(in[6], clock[3]);

    always@(posedge clock[0])
        out[0] = in[7];

    always@(posedge clock[1])
        out[1] = in[8];

    always@(posedge clock[3])
        out[3] = in[10];

    always@(posedge clock[2])
        out[2] = in[9];
endmodule

module clkGen(in, out);
    input [5:0] in;
    output [2:0] out;

    reg    [2:0]out;

```

```
wire    net1;
reg     net2, net3;

blackBox bb1(out[0], in[0]); // blackBox

always@(in[1] or in[4])
begin
    net2 = in[1] ? in[2] : in[3];
    out[1] = net2 & in[4]; // Gated
end

always@(posedge in[5])
begin
    out[2] = net3;
    net3   = ~net3; // Sequential
end
endmodule

module mod(clock, outClock);
    input  clock;
    output outClock;

    reg    outClock;
    reg    net1;

    always@(posedge clock)
    begin
        outClock = net1; //Sequential Clock Source
        net1     = ~net1;
    end
endmodule
```

For this example, SpyGlass generates the following message:

Gated clock 'test.mod1.outClock' not generated in a separate module at top-level

GateResetAtTop

Gated/generated resets exist in the design that are not all generated from one top-level block

Language

Verilog, VHDL

Description

The GateResetAtTop tag flags internally generated reset signals that are not generated in a separate top-level module.

Localizing resets generation to one module at the top-level makes it easier to understand and control these features. This reset generation module can have hierarchy.

Internally generated resets can have the following issues:

- Timing of internally generated resets may be tightly dependent on target technology and therefore may have reduced portability.
- It is difficult to put design constraints during synthesis for design with internally generated resets.

The GateResetAtTop tag reports violation if any one or more condition described below occurs:

- Top-level module has asynchronous resets and they are not ports
- Asynchronous resets are generated in more than one instance
- All asynchronous resets are generated in a instance, but all output ports of that instance are not reset ports.

Parameters

REPORT_ONLY_USER_NETS: The default value of this parameter is 1. Specifies that the tag will not report violations when the reset source is not a user defined RTL net. Set the value of the parameter to 0 to enable reporting violations.

Message Details

The following message appears at the location where a gated reset signal *<rst-name>* is encountered that is not generated in a separate top-level

module:

Gated reset '<rst-name>' not generated in a separate module at top-level

tag Severity

Guideline

Suggested Fix

Move all reset generation/gating to a special module dedicated for that purpose.

Examples (Verilog)

Consider the following example where the reset signal `reset4` is not generated in a separate module at top-level unlike other reset signals `reset1`, `reset2`, and `reset3`:

```
module test2 (clk, in, out);
    input clk, in;
    output out;

    wire reset1, reset2, reset3, reset4, a, b;

    genReset ins1(clk, in,a, b, reset1, reset2, reset3);
    test3 ins2(clk, in, reset4);
    useReset2 eg2(clk, reset1, reset2, reset3, reset4);
endmodule

module genReset(clk, in,a, b, reset1, reset2, reset3);
    input clk, in, a, b;
    output reset1, reset2, reset3;

    reg reset2;

    assign reset1 = a & b;

    blackBox bb(reset3, a, b);
```

```
    always @ (posedge clk)
        reset2 = in;
endmodule

module test3 (clk, in, reset4);
    input clk, in;
    output reset4;

    reg reset4;

    always @ (posedge clk)
        reset4 = ~in;
endmodule

module useReset2(clk, reset1, reset2, reset3, reset4);
    input clk, reset1, reset2, reset3, reset4;

    wire in1, in2, in3, in4;
    reg out1, out2, out3, out4;

    always @ (posedge clk or posedge reset1)
        if (reset1)
            out1 = 1'b0;
        else
            out1 = in1;

    always @ (posedge clk or posedge reset2)
        if (reset2)
            out2 = 1'b0;
        else
            out2 = ~in2;

    always @ (posedge clk or posedge reset3)
        if (reset3)
            out3 = 1'b0;
        else
            out3 = in3;
```

```
always @ (posedge clk or posedge reset4)
  if (reset4)
    out4 = 1'b0;
  else
    out4 = ~in4;
endmodule
```

For the following example, SpyGlass generates the following message:

Gated reset 'test2.ins2.reset4' not generated in a separate module at top-level

[OpenMore Tags](#)

IEEEType

Signal type is not an IEEE type

Language

VHDL

Description

The IEEEType tag flags signal declarations that are of non-standard IEEE types.

While creating new types and subtypes is allowed from portability angle, these types and subtypes must be based on IEEE standard types.

Following this guideline helps maintain the same standard across the design and in resolving multiple drivers.

Related Standard

Section 5.3.1 of RMM

Message Details

The following message appears at the location of the declaration of a signal *<signal-name>* that is of a type mark *<signal-type>* other than IEEE type mark:

Type of signal '*<signal-name>*' is '*<signal-type>*'. Use only IEEE types

tag Severity

tag

Suggested Fix

Use standard IEEE types wherever possible.

Examples (VHDL)

Consider the following example:

```
library ieee;  
use ieee.std_logic_1164.all
```

```
entity andgate is
  port ( in1, in2 : in bit; output : out bit );
end andgate;

architecture behav of andgate is
  begin
    output <= in1 and in2;
  end behav;
```

[OpenMore Tags](#)

ModConst

Modification of constant parameter in subprogram

Language

VHDL

Description

The ModConst tag flags statements that modify constant values.

As there is no equivalent capability in Verilog, using such constructs hinders possible future translation of the design to Verilog.

Related Standard

Section 5.3.7 of RMM

Message Details

The following message appears at the location where the value of a constant parameter *<param-name>* is being modified:

Constant parameter '*<param-name>*' in subprogram is being modified

tag Severity

Guideline

Suggested Fix

Avoid use of this technique if possible.

Examples

Consider the following example:

```
library ieee;  
use ieee.std_logic_1164.all  
  
entity e is end;  
  
architecture e of e is
```

```
function double(  
    constant inp : bit_vector(3 downto 0) := O"17"  
    return bit_vector is  
    begin  
        return not(inp);  
    end function double;  
  
function sig_double(  
    signal inp : bit_vector(3 downto 0) )  
    return bit_vector is  
    begin  
        return not(inp);  
    end function sig_double;  
constant ten : bit_vector(3 downto 0) := B"0110";  
begin  
    process  
        constant v1 : bit_vector(3 downto 0) := X"F";  
        constant v7 : bit_vector(3 downto 0) := O"6";  
        variable v2,v4,v3,v5,v8 : bit_vector(3 downto 0);  
        variable v6 : bit_vector(3 downto 0) := B"0000";  
        begin  
            v3 := double(ten);  
            v2 := double;  
            v6 := B"0010";  
            v4 := double("111");  
            v5 := double(v1);  
            v8 := double(v7);  
            wait;  
        end process;  
    end;  
end;
```

[OpenMore Tags](#)

NoBlock

Block Statement detected

Language

VHDL

Description

The NoBlock tag flags `block` constructs used in the design.

As there is no equivalent of `block` constructs in Verilog, using such constructs hinders possible future translation of the design to Verilog.

Related Standard

Section 5.3.7 of RMM

Message Details

The following message appears at the location where a `BLOCK` keyword is encountered:

Avoid `block` statements. They are not portable to Verilog.

tag Severity

Guideline

Suggested Fix

Avoid use of `block` statements if possible.

Examples

Consider the following example where block statements are used in the design:

```
library ieee;  
use ieee.std_logic_1164.all  
  
entity entBlockHeader1 is  
  port( in1      : in bit;
```

```
        in2      : in bit;
        pcarry   : in bit;
        sum      : out bit;
        scarry    : out bit);
end entBlockHeader1;

architecture behavioral of entBlockHeader1 is
begin
    b1 : block --block statement used
        port ( a, b, c : in bit; d, e : out bit );
        port map (a => in1,b => in2,c => pcarry,
            d => sum,e => scarry);

        begin
            d <= (a xor b xor c) ;
            e <= ((a and b) or (a and c) or (b and c));
        end block;
    end behavioral;
```

[OpenMore Tags](#)

NameLength

Name length is inconsistent with convention

Language

Verilog, VHDL

Description

The NameLength tag flags following types of signal, variable, or port names:

- Names that do not start with an alphabet
- Names that do not have a length between the minimum and maximum allowed number of characters

By default, the minimum and the maximum permissible length to be checked are set to 3 and 8 respectively. Use the [name_min_length](#) and [name_max_length](#) tag parameters to change the minimum and the maximum permissible length respectively.

By default, the NameLength tag flags 1000 messages only. Set the [archname](#) tag parameter to *yes* or *<tag-name>* to report all violation messages.

The requirement checked by the NameLength tag is a guideline for enhanced readability.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the location where a tag-violating name *<name>* is first encountered:

Identifier '*<name>*' length is not between *<min-allowed-length>* and *<max-allowed-length>* characters

Where *<min-allowed-length>* is the minimum allowed number of characters and *<max-allowed-length>* is the maximum allowed number of characters.

tag Severity

Guideline

Suggested Fix

Rename reported objects to meet the required name-length restrictions.

Examples (VHDL)

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

entity e is
  port(i1 : in bit;
        o2 : out bit);
end e;
architecture e of e is
  function do(constant ip : integer := 10)
    return integer is
begin
  return ip * 2;
end function do;

  function nd(ip : integer := 10) return integer is
begin
  return ip * 2;
end function nd;

  function sd(signal ip : bit)    return bit is
begin
  return not ip ;
end function sd;

  function vd( ip : integer)    return integer is
begin
  return ip * 2 ;
end function vd;
```



```

constant tn : integer := 10;
  component cpg
    generic(g1 : integer := 10);
  end component;

signal v4 : bit;
  begin
    t1 : cpg
  generic map (20);
  l1:process
  variable v1, v2 : integer;
  variable v3 : bit;
  begin
    v1 := 10;
    v1 := do(10);
    v2 := do;
    v2 := nd(v1);
    v1 := nd(tn);
    v3 := sd(v4);
    wait;
  end process l1;
end;

```

Examples (Verilog)

Consider the following example:

```

module D_FF(q_out , d, clock_signal, reset);
  input d;
  //message - port name less than 3 chars long
  input reset;
  input clock_signal;
  output q_out;
  reg q_out;

  wire b;
  //message - signal name less than 3 chars long
  wire long_name;

```

```
//message - signal name more than 8 chars long
time save_sim_time;
integer i;
always @(posedge reset or negedge clock_signal)
begin
    if (reset)
        q_out = 1'b0;
    else
        q_out = d;
end
endmodule
```

[OpenMore Tags](#)

NoGenerate

Generate Statement detected

Language

VHDL

Description

The NoGenerate tag flags `generate` constructs used in the design.

As there is no equivalent of `generate` constructs in Verilog, using such constructs hinders possible future translation of the design to Verilog.

Related Standard

Section 5.3.7 of RMM

Message Details

The following message appears at the location where a `generate` keyword is encountered:

Avoid `generate` statements. They are not portable to Verilog.

tag Severity

Guideline

Suggested Fix

If you are using `generate` in a loop to create a set of instances, instead explicitly instantiate each case. If you are using `generate` to pick alternate bindings, consider if it possible to make that selection explicit without a `generate` (this technique will anyway not map to Verilog).

Examples

Consider the following example where `generate` statement is used in the design:

```
library ieee;  
use ieee.std_logic_1164.all
```

```
ENTITY myentity IS
  PORT (
    sig1 : IN std_logic;
    sig2 : IN std_logic;
    SIG3 : OUT std_logic;
    mysignal : OUT std_logic_vector (12 downto 0)
  );
END myentity;
ARCHITECTURE myarch OF myentity IS
  BEGIN
    SIG3 <= sig1 AND sig2;

    mygen1 : for i in 0 to 12 generate
--generate statement used
      mysignal(i) <= sig1 and sig2;
    end generate;
  END myarch;
```

NoScripts

Synthesis script detected

Language

Verilog, VHDL

Description

The Unscripted tag flags the synthesis scripts used in the design.

By default, the NoScripts tag flags the following lines:

```
//synopsys dc_script_begin  
//synopsys dc_script_end  
  
--pragma dc_script_begin  
--pragma dc_script_end
```

Use the [no_scripts](#) tag parameter to provide a list of scripts that should not be allowed in the design.

It is not recommended to use the synthesis scripts in the design for the following reasons:

1. Other users may not be aware of the exact purpose of the scripts and hence, may get poor synthesis results.
2. The design may not be reusable without major re-coding if the aim of the new application is different.
3. The scripts may be obsolete by the synthesis vendors.

Related Standard

Section 5.3.5 of RMM

tag Arguments

By default, SpyGlass flags the comment lines containing the keywords `dc_script_begin` and `dc_script_end`. You can modify the `@NoScripts_arg_list` variable in the PERL control script to search for other types of lines.

Message Details

The following message appears at the location where a synthesis script `<script-name>` is used:

Avoid using synthesis script '`<script-name>`' in design

tag Severity

Guideline

Suggested Fix

Remove scripts and document special synthesis requirements for this block elsewhere.

Examples (Verilog)

Consider the following example where synthesis scripts are used in the design:

```
module test(in1, in2, in3, sel, out);
    input in1, in2, in3;
    input [3:0] sel;
    output out;
    reg out;
    // synopsis dc_script_begin
    //set_attribute P1 INIT
    "01"
    //set_attribute P2 INIT
    "00"
    // synopsis dc_script_end

    always @(in1 or in2 or in3 or sel)
    begin
        case(sel)
            4'h0 : out = in1;
            4'h1 : out = in2;
            4'h2 : out = in3;
            4'h3 : out = in1 + in2;
            default : out = in1 * in2;
        endcase
    end
```

[OpenMore Tags](#)

```
end  
endmodule
```

Examples (VHDL)

Consider the following example:

```
library ieee;  
use ieee.std_logic_1164.all  
  
-- pragma dc_script_begin  
-- set_max_area 100.0  
-- pragma dc_script_end  
entity one is  
    port ( a : in bit;; b : in bit; c : out bit );  
end one;  
  
architecture arch_one of one is  
    begin  
        c <= a and b;  
    end arch_one;
```

OneFile

Some of the design units (entity, architectures, configurations, packages) relating to an entity are not found in the same source file.

Language

VHDL

Description

The OneFile tag flags those design units of specified types that are not in the same source file.

You can specify to check for the following design unit types to be present in the same source file:

- An entity and its architectures
- An entity, its architectures, and its configurations
- An entity and its configurations
- Architectures and configurations of an entity (the entity need not be in the same source file)
- A package and its package body

Following this guideline enhances readability of the design and reduces the chances of version mismatch between files.

Related Standard

Section 5.2.1 of RMM

tag Arguments

You can use any of the following arguments in the controlling PERL file to check for different combinations of design unit types:

```
library ieee;  
use ieee.std_logic_1164.all  
  
entity arch  
entity arch config  
entity config
```


[OpenMore Tags](#)

```
arch config
package packagebody
```

NOTE: *The order of the design unit type names in the tag arguments must not be altered.*

Message Details

The following message appears at the location where the specified types of design units are not found in the same source file:

```
Entity('<ent-name>'), Architecture('<arch-name>'),
Configuration('<config-name>'), Package('<package-name>') and
Package Body('<packagebody-name>') should be in the same source
file
```

The message shows the names of the specified design unit types only. Other design unit type names are shown as n/a. For example, you have specified entity arch as the OneFile tag argument and an entity named char_type1 and its architecture named rtl are not found in the same source file, then the message is as follows:

```
Entity('char_type1'), Architecture('rtl'), Configuration('n/
a'), Package('n/a') and Package Body('n/a') should be in the
same source file
```

tag Severity

Guideline

Suggested Fix

Group related design units (entity, architecture and configuration) into the same file.

Examples (VHDL)

Consider the following example where the entity ent_fact and architecture behav are in separate files:

```
library ieee;
use ieee.std_logic_1164.all
```

```
architecture behav of ent_fact is
begin
```

```
process(datain)
  variable temp: Integer :=1;
  begin
    for i in 1 to datain loop
      temp := temp * i ;
    end loop ;
    dataout <= temp ;
  end process;
end behav;
```

[OpenMore Tags](#)

PackHdr

Standard header not found before package declaration

Language

VHDL

Description

The *PackHdr* tag flags the package headers that do not conform to the recommended structure.

By default, SpyGlass checks for the package header in the following format:

```
-----
-- package name: <package name>
-- purpose: <your comments>
-- compile to: <library name>
-- Release History: <your comments>
-- VERSION      Date      AUTHOR      DESCRIPTION
-- <version> <date>      <name>      <comments>
-- <version> <date>      <name>      <comments>
-----
```

The exact format of the package headers should be adjusted to company standards.

You can use the [package_header_check_file](#) parameter to adjust the format of the file header to your own company standards.

If this parameter is not set then the default header format mentioned above is considered.

NOTE: Make sure that the subroutine name in this file is always `pack_check` for the **PackHdr** tag. The subroutine name should always be `ve_header_check` for the [FileHdr](#) tag.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears for each package header line that does not conform to the default package header format:

Non-standard package header: `<error-type>`

Where `<error-type>` is the type of error in the line.

tag Severity

tag

Suggested Fix

Determine and follow the local approved standard.

ProcName

Process label does not follow the naming convention

Language

VHDL

Description

The ProcName tag flags process labels that do not follow the recommended naming conventions.

By default, the ProcName tag requires that the process labels should follow the following naming convention:

- Labels should contain only alphanumeric characters or underscore
- Labels should start with an alphabet
- Labels should end with `_PROC`
- Labels should not be longer than 16 characters
- Labels should not have two successive underscores (Use of two successive underscores are unsupported in some hardware accelerators and emulators)

Use the [procname](#) tag parameter to specify your own process label naming convention.

Following the same naming convention for process labels enhances readability of the design.

Related Standard

Section 5.2.1 of RMM

Message Details

The following message appears at the first line of a process block `<process-name>` whose label does not follow recommended naming convention:

Process label '`<process-name>`' does not follow naming convention.

tag Severity

Guideline

Suggested Fix

Modify all references to this name to a form that meets the convention.

Examples

Consider the following example where process label `test_PROCCC` does not follow the naming convention:

```
library ieee;
use ieee.std_logic_1164.all

entity char_type1 is
  port ( in1: bit_vector(3 downto 0);
         in2: bit_vector(3 downto 0);
         in3: bit_vector(3 downto 0);
         out1: out bit_vector(3 downto 0)
        );
end char_type1;

architecture Rtl of char_type1 is
  signal signalchecked : bit;
  function func (
    in1: character;
    signal in2,in3: bit_vector( 3 downto 0)) return
bit_vector is
  begin
    case in1 is
      when 'a' => return in2 or in3;
      when 'b' => return in2 and in3;
      when 'c' => return in2 nand in3;
      when 'd' => return in2 xor in3;
      when others => return in2 nor in3;
    end case;
  end;
end;
```

```
begin
  test_PROCCC: process(in1,in2,in3)
  variable var : character;
  begin
    case in1 is
      when "0000" => var:='a';
      when "0010" => var:='b';
      when "0100" => var:='c';
      when "0110" => var:='d';
      when "1000" => var:='e';
      when "1010" => var:='f';
      when "1100" => var:='g';
      when others => var:= 'h';
    end case;
    out1<= func(var,in2,in3);
  end process;
a_POCC: process(in1,in2,in3)
  variable var : character;
  begin
    case in1 is
      when "0000" => var:='a';
      when "0010" => var:='b';
      when "0100" => var:='c';
      when "0110" => var:='d';
      when "1000" => var:='e';
      when "1010" => var:='f';
      when "1100" => var:='g';
      when others => var:= 'h';
    end case;
    out1<= func(var,in2,in3);
  end process;
end;
```

ReserveName

A reserve name has been used.

Language

Verilog, VHDL

Description

Verilog

The ReserveName tag flags the object names that are same as the VHDL reserved words.

The ReserveName tag also flags the case variants of the VHDL keywords.

By default, the ReserveName tag flags the following VHDL keywords used in Verilog design”

abs	access	after	alias	all
and	architecture	array	assert	attribute
begin	block	body	buffer	bus
component	configuration	constant	disconnect	downto
elsif	entity	exit	file	generate
generic	group	guarded	impure	in
inertial	is	label	library	linkage
literal	loop	map	mod	new
next	null	of	on	open
others	out	package	port	postponed
procedure	process	pure	range	record
register	reject	rem	report	return
rol	ror	select	severity	signal
shared	sla	sll	sra	srl
subtype	then	to	transport	type
unaffected	units	until	use	variable
when	with			

You can add more VHDL keywords to the above list by using the

[reserve_names](#) tag parameter.

It is recommended not to use VHDL keywords in Verilog designs. Using VHDL keywords in Verilog designs hinders future translation of Verilog design to VHDL, and creates reusability problem for VHDL-based end-users.

By default, the ReserveName tag flags only the first occurrence of a reserved keyword. Set the [allReserveWords](#) tag parameter to yes to report all occurrences of the reserved keywords.

NOTE: *This tag might flag duplicate violations when a UDP is declared within the v/y file after a module declaration.*

VHDL

The ReserveName tag flags the object names that are same as Verilog reserved words.

By default, the ReserveName tag flags the following Verilog keywords used in VHDL design:

always	assign	buf	bufif0	bufif1
case	casez	cmos	deassign	default
defparam	disable	edge	endattribute	endcase
endfunction	endmodule	endprimitive	endspecify	endtable
endtask	entity	force	forever	fork
highz0	highz1	ifnone	initial	join
large	macromodule	medium	module	negedge
nmos	notif0	notif1	parameter	pmos
posedge	primitive	pull0	pull1	pulldown
pullup	rcmos	real	realtime	reg
release	repeat	rnmos	rpmos	rtran
rtranif0	rtranif1	tranif0	tranif1	
scalared	small	specify	specparam	strength
strong0	strong1	supply0	supply1	table
task	tran	tranif0	tranif1	tri
tri0	tri1	triand	trior	trireg
wire	vectored	wand	weak0	weak1
wire	xnor	xor		

You can add more Verilog keywords to the above list by using the [reserve_names_vh](#) tag parameter.

It is recommended not to use Verilog keywords in VHDL designs. Using Verilog keywords in VHDL designs hinder future translation of VHDL design to Verilog, and create reusability problem for Verilog-based end-users.

Related Standard

Section 5.2.9 of RMM

Message Details

Verilog

The following message appears at the location where a VHDL reserved name `<res-name>` is encountered in a Verilog design:

Reserved name '`<res-name>`' used

VHDL

The following message appears at the location where a Verilog reserved word `<res-word>` is encountered in a VHDL design:

Reserved word '`<res-word>`' used

tag Severity

tag

Suggested Fix

With the rapidly evolving strategies, alliances, and acquisitions, it is difficult to guarantee that a design block will never be used in an 'alien' context. For example, a module designed in Verilog may have a high value in a design written in VHDL. Hence, avoid using VHDL keywords in Verilog designs to avoid such problems.

PortOrder_C

Ports not declared in recommended order

Language

VHDL

Description

The PortOrder_C tag flags port declarations that are not in the recommended order for entities without any architecture defined.

By default for VHDL designs, the recommended order for port declaration is as follows:

1. Input ports
2. Output ports
3. Inout ports
4. Buffer ports
5. Linkage ports

Under the recommended port definition order, a message is generated for the following cases:

- Every input port specified after an output, inout, buffer or linkage port.
- Every output port specified after an inout, buffer or linkage port
- Every inout port specified after a buffer or linkage port
- Every buffer port specified after a linkage port

Use the [portorder](#) tag parameter to define your own order of port declaration.

Declaring ports in a consistent order aids understanding and reduces the likelihood of incorrect connections.

Related Standard

Section 5.2.10 of RMM

Message Details

The following message appears at the location of the port declaration of the port `<port-name>` that is not in the recommended order:

Port '<port-name>' not defined in recommended order

tag Severity

tag

Suggested Fix

Reorder port definitions to comply with this convention.

[OpenMore Tags](#)

SepClock

Both positive and negative edge clocks are used in the same module

Language

Verilog, VHDL

Description

The SepClock tag flags modules that use both positive and negative edges of clock signals to describe sequential elements.

If you need to use both clock edges, it is recommended to separate the description of two edges in to different modules to minimize the chances of unforeseen dependencies on duty cycle.

Related Standard

Section 5.4.1 of RMM

Message Details

The following message appears at the location of the flip-flop *<flop1-name>* based on a positive (negative) edge of the clock signal *<clk1-name>* when a flip-flop *<flop2-name>* already exists in the module *<module-name>* that uses the negative (positive) edge of clock signal *<clk2-name>*:

Different edges of clocks '*<clk1-name>*' (flop '*<flop1-name>*') and '*<clk2-name>*' (flop '*<flop2-name>*') used in module '*<module-name>*'

tag Severity

Guideline

Suggested Fix

Limit cases where this is necessary and clearly document the fact at least in the file header.

Examples (Verilog)

Consider the following example where different edges of clocks `clock1` and `clock2` are being used to describe flip-flops:

```
module test(clock1, clock2, in1, in2, out1, out2);
  input clock1, clock2;
  input in1, in2;
  output out1, out2;
  reg out1, out2;

  always@(posedge clock1)
    out1 = in1;

  always@(negedge clock2)
    out2 = in2;
endmodule
```

For this example, SpyGlass generates the following message:

Different edges of clocks 'test.clock1' (flop 'out1') and 'test.clock2' (flop 'out2') used in module 'test'

Examples (VHDL)

Example of Different Edges of the Same Clock Used

Consider the following example where the positive and negative edges of the clock `clk1` are used to define the sequential elements `output1` and `output2` respectively:

```
library ieee;
use ieee.std_logic_1164.all

architecture rtl of test is
begin
  process(ck1)
  begin
    if(ck1'event and ck1 = '1') then
      output1 <= d1;
    end if;
  end process;
```

```

    process(ck1)
    begin
        if(ck1'event and ck1 = '0') then
            output2 <= d2;
        end if;
    end process;
end rtl;

```

For this example, SpyGlass generates the following message:

Different edges of clocks 'rtl.ck1' (flop 'output1') and 'rtl.ck1' (flop 'output2') used in module 'rtl'

To fix this message, separate the two sequential elements into different architectures as follows:

```

architecture rtl_1 of test is
begin
    process(ck1)
    begin
        if(ck1'event and ck1 = '1') then
            output1 <= d1;
        end if;
    end process;
end rtl_1;
architecture rtl_2 of test is
begin
    process(ck1)
    begin
        if(ck1'event and ck1 = '0') then
            output2 <= d2;
        end if;
    end process;
end rtl_2;

```

Uniq8Char

Signal, variable, port or other named object is not unique up to defined character limit

Language

Verilog, VHDL

Description

The Uniq8Char tag flags those names that have the same unique first 8 characters as a name found earlier.

Some tools allow longer signal names, variable names, or port names but store or consider unique only the first 8 characters of the signal, variable, or port name.

By default, the Uniq8Char tag flags those identifiers that are not unique in the first 8 characters. Use the [uniq_char_val](#) tag parameter to change the default value (8) for the number of characters up to which the names have to be unique.

For Verilog, the name of a specific construct will be checked against the names of some other specific (not all) constructs only, within the same scope. The following table clearly describes the constructs that should be compared against a specific language construct:

First Construct	Second Construct
signal	All the constructs in the same scope (signals / variables / ports / parameters / function/task/instance/named-block)
variable	All the constructs in the same scope (signals / variables/ports / parameters/ function/task/instance/named-block)
port	All the constructs in the same scope (signals / variables / ports / parameters/ function/task/instance/named-block)
parameter	All the constructs in the same scope (signals/ variables/ports/parameters/ function/task/instance/named-block)
module in instantiation	All modules/UDP's

OpenMore Tags

First Construct	Second Construct
instance in instantiation	All the constructs in the same scope (signals/variables/ports/parameters/function/task/instance/named-block/module)
function	All the constructs in the same scope (signals/variables/ports/parameters/function/task/instance/named-block)
task	All the constructs in the same scope (signals/variables/ports/parameters/function/task/instance/named-block)
named-block	All the constructs in the same scope (signals/variables/ports/parameters/function/task/instance/named-block)
module	All modules/UDP's
UDP	All module/UDP's

NOTE: *The name of the constructs (signal/variable/port/parameter/instance/function/task/named-block) will also be checked against the name of the module containing the construct.*

The scope is defined by a module/UDP, function, task and named-block boundaries. The visibility of a construct defined in one scope is restricted to the boundaries of that scope. If a scope is contained in another, it makes the construct defined in the containing scope also visible in the contained scope. Variables defined within parallel scopes, however, are not visible across boundaries.

Related Standard

Section 5.2.1 of RMM

Message Details

Verilog

The following message appears at the location where an object name *<curr-name>* is first encountered that has the same first 8 characters as the *<first-name>* encountered earlier at line *<num>* in file *<file-name>*:

Name '<curr-name>' not unique in first <uniq-char-val> characters (prior usage '<first-name>', at file: <file-name>, line <num>)

VHDL

The following message appears at the location where an object name <curr-name> is first encountered that has the same first 8 characters as the <first-name> encountered earlier at line <num> in file <file-name>:

Names not unique in first <uniq-char-val> characters. '<curr-name>' conflicts with '<first-name>' defined at <file-name>(line <num>)

tag Severity

Guideline

Suggested Fix

Rename report signals, variables, etc. so they become unique in the first 8 characters.

Examples (Verilog)

Consider the following example:

```
module Uniq8Char(a_in, b_in, rst, clk, q_output);
  input rst, clk;
  input a_in, b_in ;
  output q_output;
  reg q_output;
  reg q_outputbar;

  always@(posedge clk)
  begin
    if(rst)
    begin
      q_output = 1'b0;
      q_outputbar = 1'b1;
    end
  end
```

```

        else
            q_output = a_in | b_in ;
        end

always@(q_output)
begin
    q_outputbar = !q_output;
end
endmodule

```

Examples (VHDL)

Consider the following example:

```

library ieee;
use ieee.std_logic_1164.all

ENTITY myentity IS
    GENERIC (mygensdhfsjdfhskdhfkdsjfdjfd :integer:= 4);
    PORT (
        sig12345a : IN std_logic;
        sig12345b : IN std_logic;
        sig12345c : OUT std_logic;
        firstsignal : IN std_logic;
        secondsignal : IN std_logic;
        thirdsignal : OUT std_logic
    );
END myentity;

ARCHITECTURE myarch OF myentity IS
    BEGIN
        sig12345c <= sig12345a AND sig12345b;
        thirdsignal <= firstsignal OR secondsignal;

        process
            variable howamidoing1 : std_logic;
            variable howamidoing2 : std_logic;
            variable howamidoing3 : std_logic;
            begin

```

```
        howamidoing1 := '1';  
end process;  
END myarch;
```

Simulation Tags

This section explains the simulation Tags in VC SpyGlass Lint.

sim_loop01

Possible mismatch in gate-RTL simulation due to simulation event loops

When to Use

Use this Tag to detect event simulation loops in a module.

Description

The *sim_loop01* Tag reports event simulation loops in a module. These loops create possible mismatch in gate-RTL simulation.

During simulation, an event can be propagated in any of the following ways:

- From a signal in the sensitivity list to a signal in LHS of blocking assignment statements of an `always` block
- From a signal in RHS to a signal in LHS of a continuous assignment statement, which includes a wire assignment
- From input ports to output ports in case of Verilog built-in gates instantiation

An event simulation loop is detected when an event on a signal in the sensitivity list of a combinatorial `always` block, say A1, directly or indirectly triggers continuous assignment statements or combinatorial `always` blocks or built-in gate, which again triggers the `always` block A1.

An event loop should meet the following criteria:

- At least one combinatorial `always` block
- A cyclic loop among the combinatorial `always` block or built-in gate instantiation or continuous assignment statements

Tag Exceptions

The *sim_loop01* Tag does not propagate an event on a signal across RTL modules. Refer to [Example 5](#) for an illustration.

Parameters

- [*effort_simloop_level*](#): Default value is 200. This indicates a net is traversed 200 times during loop detection. Set this parameter to a

different integer to change the amount of efforts put by the Tag in detecting a simulation loop.

- ***overlapsimloops***: Default value is `no`. This indicates the Tag does not report overlapping simulation loops. Set this parameter to `yes` to report overlapping simulation loops. Refer to [Example 4](#) for an illustration.
- ***strict***: Default value is `no`. Set this parameter to `yes` to report loops containing multiple always blocks.

Constraints

None

Messages and Suggested Fix

The following message appears when the event simulation loop `<loop-name>` is detected inside the module `<module-name>`.

[WARNING] Simulation loop `<loop-name>` detected in module `<module-name>` [Hierarchy: '`<hier-path>`']

Potential Issues

This violation appears when an event simulation loop is detected inside a module.

Consequences of Not Fixing

An event simulation loop leads to possible mismatch in gate-RTL simulation.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where an event simulation loop is detected in a module.

To fix the problem, review the RTL and break the undesired loop. It is advisable to avoid using an event simulation loop in a module.

Example Code and/or Schematic

Example 1

Consider the following example:

```

assign c = b;
always @(a or c)
begin
    d = c;
    b = a;
end

```

A diagram illustrating a feedback loop. A green arrow points from the variable `b` in the assignment `b = a;` inside the `always` block to the variable `b` in the continuous assignment `assign c = b;`. A red arrow points from the variable `c` in the `always` block's sensitivity list `@(a or c)` back to the variable `c` in the continuous assignment.

In this case, signal `b` is assigned in the `always` block. The signal triggers signal `c` of the continuous assignment statement, which again triggers the `always` block. This creates an event simulation loop. As a result, the `sim_loop01` Tag reports a violation.

Example 2

Consider the following example:

```

assign c = f;           //first continuous assignment statement
assign f = b;           //second continuous assignment statement
always @(a or c)
begin
    d = c;
    b = a;
end

```

A diagram illustrating a feedback loop. A green arrow points from the variable `b` in the assignment `b = a;` inside the `always` block to the variable `b` in the continuous assignment `assign f = b;`. A blue arrow points from the variable `f` in the continuous assignment `assign f = b;` to the variable `f` in the continuous assignment `assign c = f;`. A red arrow points from the variable `c` in the `always` block's sensitivity list `@(a or c)` back to the variable `c` in the continuous assignment `assign c = f;`.

In this case, signal `b` is assigned in the `always` block. The signal triggers signal `f` of the second continuous assignment statement, which further triggers signal `c` of the first continuous assignment statement. Signal `c` again triggers the `always` block. This creates an event simulation loop. As a result, the `sim_loop01` Tag reports a violation.

Example 3

Consider the following example that contains two `always` blocks:


```

always @(b)                                //first always block
    c = b;

always @(a or c)                            //second always block
begin
    d = c;
    b = a;
end

```

In this case, signal *b* is assigned in the second always block. The signal triggers signal *c* of the first always block, which again triggers the second always block. This creates an event simulation loop. As a result, the *sim_loop01* Tag reports a violation.

Example 4

Consider the following example that contains three always blocks:

```

//first always block
always @(b)
    c = b;

//second always block
always @(b)
    e = b;

//third always block
always @(a or c or e)
begin
    d = c & e;
    b = a;
end

```

In this case, the two overlapping loops are created as follows:

- **First overlapping loop:** Signal *b* is assigned in the third `always` block. The signal triggers signal *c* of the first `always` block, which again triggers the third `always` block. This creates an event simulation loop.
- **Second overlapping loop:** Signal *b* is assigned in the third `always` block. The signal triggers signal *e* of the second `always` block, which again triggers the third `always` block. This creates an event simulation loop.

By default, the Tag reports only one violation for any one loop. If you set the [overlapsimloops](#) parameter to `yes`, the Tag reports two violations for these overlapping loops.

Example 5

Consider the following example:

```
assign c = f;
always @(a or c or e)
begin
    d = c & e;
    b = a;
end
mod A1(.in(b), .out(f)); //mod A1 is an RTL module
```

In this case, module `A1` is an RTL module. As the `sim_loop01` Tag does not propagate an event on a signal across RTL modules, the Tag does not report a violation.

Default Severity Label

Warning

Tag Group

SimulationTags

Simulation Tags

Reports and Related Files

None

sim_race04

Avoid Event Counting Code

The *sim_race04* Tag flags `always` constructs with functionality that depends on the number of times the `always` construct is invoked in the same simulation step.

If a value is updated in an `always` construct, this value should be insensitive to the number of times that construct can be invoked in a simulation time step. An `always` construct can be invoked more than once in a simulation time step if its sensitivity list contains more than one signal. If there is logic in the `always` construct that depends on the value of a variable set during the previous invocation of the same `always` construct, then that functionality can be implementation dependent.

The *sim_race04* Tag checks for the following situations and flags the constructs that meet both the following conditions:

1. A statement in the body of `always` construct has at least one variable used in both LHS and RHS.
2. There are more than one item in the sensitivity list

Message Details

The following message appears at the location of an event counting statement:

[SIMULATION-RACE] Event counting code used

The message is back-annotated to the RTL display.

Tag Severity

Simulation-Race

Examples

Consider the following examples where the sensitivity list of the `always` construct has two variables and the variable `i` counts the number of times the `always` construct is invoked:

```
always @(A or B)
begin
```

Simulation Tags

```
        i = i + 1;  
    end  
always @ (event1 or event2)  
begin  
    i = i + 1;  
end
```

sim_race05

Avoid Time Zero race

The *sim_race05* Tag flags signals in event triggers that are initialized to values that could cause non-deterministic event trigger.

Message Details

The following message appears when there is a possible race condition for signal *<sig-name>* due to initialization at line *<num1>* and use at line *<num2>*:

[SIMULATION-RACE] Time Zero Race condition for signal *<sig-name>* on line *<num1>* and line *<num2>*

The message is back-annotated to the RTL display.

Tag Severity

Simulation-Race

Examples

Consider the following example where the `always` construct could be triggered if the event trigger is established before the initial value is assigned to the clock signal `clk`:

```
initial
begin
    clock = 1;
    forever #50 clock = ~clock;
end

always @(posedge clk)
    $display("...")
```

The `always` construct may or may not be triggered at time 0 depending on when the clock `clk` is initialized and when the event trigger is set by the simulator.

Since the clock has been initialized to 1, the `always` construct could be triggered if the event trigger is established before the initial value is assigned to the clock signal but there would not have been any problem if

either clock was initialized with 0 or event triggering would have been happening on a negative edge.

This race condition can be fixed by ensuring that there is no transition at time 0 as follows:

```
initial
begin
    clock = 1'bx;
    #50 clock = 1'b0;
    forever #50 clock = ~clock;
end
```

sim_race01

Reports signals that are assigned and used in the same simulation cycle

When to Use

Use this Tag to detect signals that are assigned and used in the same simulation cycle.

Description

The *sim_race01* Tag reports signals that are assigned and used within the same module in the same simulation cycle. This Tag detects signals that are in two `always` constructs in the same module or one `always` construct and one continuous assignment statement in the same module.

Tag Exceptions

The *sim_race01* Tag does not report race conditions based on the time concept only, as shown in [Example 6: Tag Limitations](#).

Parameters

- [check_complete_sensitivity_list](#): The default value is `yes`. Set the value of the parameter to `no` to ignore any violation for race with respect to implicit or complete `always` block.
- [waiver_compat](#): Default value is `no`. If you set the value of this parameter to `yes` or `<Tag-name>`, it ensures that the Tag does not generate the line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.
- [effort_level_sim_race](#): Default value is 0. Set this parameter to an integer value in case of high run time to specify the amount of effort to be applied while detecting the race condition.
- [handle_large_bus](#): Default value is `no`. Set this parameter to `yes` to enable the *sim_race01* Tag to process large bus signals.
- [check_seq_assign](#): Default value is `no`. Set this parameter to `yes` to not report the race condition for a signal that is assigned in the sequential block bus signals.

Constraints

None

Messages and Suggested Fix

The following message appears at the location where the signal `<sig-name>` is assigned (line `<num1>`) and read (line `<num2>`) in the same simulation cycle:

[RACE] `<type>` Read-write Race for signal '`<sig-name>`'
(assigned/read on line `<num1>` and line `<num2>`)

Where, `<type>` can be `Direct` or `Multi-level`.

The message is back-annotated to the RTL display.

Potential Issues

This violation appears when a signal is assigned and used in the same simulation cycle.

Consequences of Not Fixing

Assigning and using signals in the same simulation cycle leads to a read-write race condition.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the location of the signal that is either assigned or read after read or assigned, respectively, somewhere else in RTL in the same simulation cycle. Trace back the RTL from this line to see other usage of the violating signal. The violation message itself shows the line number of both the usages of signal, read and assigned, which helps to debug in RTL.

To fix this problem, avoid read and assign on the same signal in the same simulation cycle.

Example Code and/or Schematic

Example 1: Race Condition Based on Triggering and Logic

In the following example, the signal `b` is assigned and read by the same triggering event that is a change in the signal `a` or `c`:

```
assign b = a | c;
always @(a or c)
```

```
begin
  if (b)
    begin
      ...
    end
  end
end
```

Example 2: Blocking Assignment Resulting in Concurrent Assign and Use

In the following example, when there is a change in the signal `d`, the signal `a` is reevaluated that in turn causes the signal `b` to be updated in the `assign` statement. However, the updated value of the signal `b` may not be visible to the `if` statement in the `always` construct, which leads to a race condition.

```
assign b = a | c;
always @ (d)
begin
  a = 0;
  if (b)
    begin
      ...
    end
end
end
```

In the above example, change the blocking assignment to signal `a` to a nonblocking assignment to avoid the race condition.

Example 3: No Race Condition

Consider the following example where assignment to the signal `a` is a nonblocking assignment that prevents the race condition:

```
assign b =a | c;
always @ (d)
begin
  a <= 0;
  if (b)
    begin
      ...
    end
end
```

```
end
```

There is no race condition even if the `assign` statement is replaced by the equivalent `always` construct.

Example 4: Race Conditions in Flip-Flop Descriptions

Consider the following example where the output of one flip-flop is fed to the input of another flip-flop at the same clock event:

```
always @(posedge clk)
    sig1 = data;

always @ (posedge clk)
    sig2 = sig1;

always @ (posedge clk)
    sig = sig2;
```

To fix this issue, remodel the example so that there exists at least one nonblocking assignment in each path leading to the data input of the flip-flop to ensure determinism between the clock and the data path.

Example 5: Multilevel Race Condition

Consider the following example where signals `ad` and `bd` depend on the same clock `clk` through signals `a` and `b`:

```
module test (reset, clk);
input reset, clk;

reg [3:0] b;
reg [5:0] out;
reg a;
wire ad;
wire [3:0] bd;

always @(posedge clk)
    if ( reset ) a = 0;
    else      a = ~a;

always @(posedge clk)
    if ( reset ) b = 0;
```

```

        else                b = b + 1;

assign ad = a;
assign bd = b;

always @(bd or ad)
    if ( ad ) out = out + 1;
endmodule

```

Depending on which signal out of `ad` and `bd` achieves the new value first, signal `out` may or may not be reevaluated. Therefore, there is a multilevel race condition between signals `ad` and `bd`.

Example 6: Tag Limitations

The *sim_race01* Tag does not report race conditions based on the time concept only, as shown in the following example:

```

initial
begin
    fork
        #5 a = 1'b0;
        #7 b = 1'b0;
    join
        #8 a = 1'b1;
    end

initial
begin
    ...
    #15 if (a)
        $display("....");
    end

```

Example 7: Implicit always blocks

Consider the following examples where, by default, the Tag reports a violation for the implicit always block. However, the Tag does not report a violation when the [check_complete_sensitivity_list](#) parameter is set to no.

```

■ module test;
  reg a, b, c, d,e;

```

Simulation Tags

```

always@(*) begin
    a= 1'b0;
    a= 1'b1;
    b = c;
end

```

```

always@(*) begin
    c = 1'b0;
    c = 1'b1;
    d = a;
end
endmodule

```

```

■ module test;
  reg a, b, c, d,e;
  always@(*) begin
    a= 1'b0;
    a= 1'b1;
    b = c;
  end

```

```

always@(d) begin
    c = 1'b0;
    c = 1'b1;
    d = a;
end
endmodule

```

```

■ module test;
  reg a, b, c, d,e;

  always@(*) begin
    a= 1'b0;
    a= 1'b1;
    b = c;
  end
  always@(posedge d) begin
    c = 1'b0;
    c = 1'b1;
    a = d;
  end
endmodule

```

Default Severity Label

Race

Tag Group

simulation

Reports and Related Files

None

sim_race02

Reports signals that have multiple assignments in the same simulation cycle

When to Use

Use this Tag to detect signals that have multiple assignments in the same simulation cycle.

Description

The *sim_race02* Tag reports signals that have multiple assignments in the same simulation cycle. Design may contain assignments to a signal in two `always` constructs in the same simulation cycle.

Tag Exceptions

The *sim_race02* Tag has the following limitations:

1. This Tag checks for race conditions across two `always` constructs only.
2. This Tag does not report race conditions based on the time concept only, as shown in [Example 4](#).

NOTE: *If you want to detect multi-driven nets, it is better to use the W415 Tag instead of *sim_race02*.*

Parameters

- ***waiver_compat***: Default value is no. If you set the value of this parameter to yes or <Tag-name>, it ensures that the Tag does not generate the line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.

Constraints

None

Messages and Suggested Fix

The following message appears at the assignment of the signal <*sig-name*> at the line <*num2*> when the signal has already been assigned at the line <*num1*>:

[RACE] write-write Race for signal '<sig-name>' (assignments on line <num1> and line <num2>)

The message is back-annotated to the RTL display.

Potential Issues

This violation appears when signals have multiple assignments in the same simulation cycle.

Consequences of Not Fixing

Multiple assignments in the same simulation cycle leads to a race condition as which assignment is executed first is not defined by Verilog runtime semantics.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the second instance of the signal assignment inside the other `always` block.

You can confirm the multiple assignments to a signal in one of the following ways:

- Simple visual inspection of the block
- Scrolling up the HDL Viewer window till you reach the `always` block
- Search for the signal assigned value in the `always` blocks of a design unit

To fix this problem, avoid multiple assignments to a signal in the same simulation cycle.

Example Code and/or Schematic

Example 1

Consider the following example where a race condition exists in combinational logic for the signal `b`:

```
always @ (a)
    b = a | c;

always @(a)
begin
    b = d & e;
end
```


Example 2

Consider the following example where a race condition exists in sequential logic for the signal b:

```
always @(posedge clka)
    b = a;

always @(posedge clka)
begin
    b = d & e;
    f = a;
end
```

Example 3

Consider the following example that illustrates a design containing multiple assignments to the signal b in two `always` constructs in the same simulation cycle:

```
module test(a,b);

input a;
output b;
reg b,c,e,d;
always @ (a)
b = a | c;
always @(a)
begin
    b = d & e;
end
endmodule
```

Example 4

This Tag does not report race conditions based on the time concept only, as shown in the following example.

```
initial
begin
  fork
    #5 a = 1'b0;
    #7 b = 1'b0;
  join
    #8 a = 1'b1;
end

initial
begin
  ...
  #15 if (a)
    $display("....");
end
```

Default Severity Label

Race

Tag Group

simulation

Reports and Related Files

None

sim_race03

Avoid assigning flip-flop outputs in blocking manner

Description

The *sim_race03* tag flags flip-flop outputs that have been assigned using blocking assignment statements in an `always` construct.

Using blocking assignment for flip-flop outputs may lead to race conditions if the output of one flip-flop is the data input of another flip-flop. Then, the sampling of data input of the second flip-flop may be occurring in the same simulation step as the output of the first flip-flop, resulting in a race condition.

Using non-blocking assignments for flip-flop outputs orders the setting and sampling of signals.

Parameter(s)

[*ignore_blocking_hanging_flop*](#): Default value is no. Set the value of the parameter to yes to enable the tags to ignore hanging flip-flops.

Message Details

The following message appears at the location of a blocking assignment of a flip-flop output *<sig-name>*:

```
[SIMULATION-RACE] Blocking assignment used for flip-flop output
'<sig-name>'
```

The message is back-annotated to the RTL display.

Severity

Simulation-Race

Examples

Consider the following example where the flip-flop output `a` of instance `dff0` of module `dff` is being used as the data input of the flip-flop `dff1` of the module `dff` and the flip-flop output `q` of the module `dff` is being assigned using blocking assignment:

```
module test (out, in, clk);
```

```
input in, clk;
output out;

wire a;

dff dff0(a, in, clk);
dff dff1(out, a, clk);
endmodule

module dff (q, d, clk);
input d, clk;
output q;

reg q;

always@(posedge clk)
    q = d;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Blocking assignment used for flip-flop output 'q'

To fix this message, change the blocking assignment to non-blocking assignment as follows:

```
always@(posedge clk)
    q <= d;
```

NOTE: The *sim_race03* tag flags a message for the module `dff` even if it was not instantiated anywhere.

sim_race06

Flip-flop data path should not be entirely blocking

Description

The *sim_race06* tag flags data path of a sequential device if any path in the fan-in cone of the data path has blocking assignments only.

To ensure that the new value of data arrives *after* the current edge of the clock has been processed, the data path should appear as the LHS of a non-blocking assignment statement. It is not necessary for all the expressions involved in the data path computation to be non-blocking assignments. However, all paths having a cone-in into the data pin of a flip-flop or a latch should have at least one non-blocking assignment.

Message Details

The following message appears when a path from net *<sig1-name>* to the data input *<sig2-name>* of a flip-flop has blocking assignments only:

[SIMULATION-RACE] Entire data path from *<sig1-name>* to *<sig2-name>* (Data pin) is Blocking

The message is back-annotated to the RTL and schematic display.

Severity

Simulation-Race

Examples

Consider the following example where there is no non-blocking assignment in the path from signal *d3* to signal *data* (data pin of flip-flop *q_reg*):

```
module test (clk, d1, d2, d3, d4, out);
    input clk, d1, d2, d3, d4;
    output out;

    reg q_reg;
    reg data1, data2;
    wire data;

    assign out = q_reg;
```

```
always @(d1 or d2)
    data1 <= d1 & d2;

always @(d3 or d4)
    data2 = d3 || d4;

assign data = data1 ^ data2;

always @ (posedge clk)
    q_reg = data;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Entire data path from d3 to data (Data pin) is Blocking

sim_race07

Non-blocking assignment should not be used in clock or enable path

Description

The *sim_race07* tag flags non-blocking assignment statements in the clock or enable path to a sequential element.

In order to ensure determinism between clock and data-paths, it is necessary that the clocks are evaluated before the change in data path, so that the current clock can capture the data before the data takes its new value.

All assignments leading to evaluation of clock signal should be a blocking assignment. Similarly, all signals that appear in the clock path (including control circuitry) should be evaluated as LHS of blocking assignments. Any signal that would eventually reach the clock pin of a flip-flop or enable pin of a latch should be generated using a blocking assignment only.

The *sim_race07* tag also checks the paths that are sourced by clock dividers.

Message Details

The following message appears when a path from net *<sig1-name>* to the clock or enable input *<sig2-name>* of a flip-flop has a non-blocking assignment:

[SIMULATION-RACE] Non-blocking assignment used in clock path from *<sig1-name>* to *<sig2-name>* (Clock/Enable pin)

The message is back-annotated to the RTL and schematic display.

Severity

Simulation-Race

Examples

Consider the following example where the clock *clk1* of the flip-flop *q_req* is generated by using a non-blocking assignment:

```
module test (gating_sig1, clk, data_in, out);
    input gating_sig1, clk, data_in;
```

```
output out;

reg q_reg;
reg clk1;
wire clk_w;

assign out = q_reg;
assign clk_w = clk1;

always @(gating_sig1 or clk)
    clk1 <= gating_sig1 & clk;

always @(posedge clk1)
    q_reg <= data_in;
endmodule
```


sim_race08

Control signals of a sequential device should not be multiply driven.

The *sim_race08* Tag flags control signals (clock, asynchronous set, and asynchronous reset signals) of sequential devices that are directly driven by multiple drivers.

Multi-driven clock, set, or reset signals may drive the sequential element to unknown or incorrect states.

The *sim_race08* Tag checks all control signals in the sensitivity list of a sequential element and flags a message if any control signal is directly driven by multiple drivers in the same or different module.

Multi-driven data input to sequential elements is allowed.

Message Details

The following message appears at the location where a multi-driven signal `<sig-name>` is encountered in the sensitivity list of a sequential element:

```
[RACE] Multiple driven signal '<sig-name>' fed to clock/Set/Reset input of sequential element.
```

The message is back-annotated to the RTL and schematic display.

Severity

Race

Examples

Example of Multi-driven Signal Used in the Same Module

Consider the following example where signal `muxed_clk` is a multi-driven signal and is in the sensitivity list of the sequential element `q_reg`:

```
module test (scan, clk, tclk, data, scan_in, out);
    input scan, clk, tclk, data, scan_in;
    output out;

    reg q_reg;
```

```
wire muxed_clk, non_scan;

assign out = q_reg;
assign muxed_clk = scan ? tclk : 1'bz;
assign non_scan = ~ scan;
assign muxed_clk = non_scan ? clk : 1'bz;

always @(posedge muxed_clk)
    q_reg <= scan ? scan_in : data;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Multiple driven signal 'muxed_clk' fed to Clock/Set/Reset input of sequential element.

Example of Multi-driven Signal Used in a Different Module

Consider the following example where the signal clock is multi-driven in module test and is used as a clock signal for sequential element through instantiation of module inst:

```
module test (tclk1, tclk2, scan1, scan2, d);
    input tclk1, tclk2, scan1, scan2;
    output d;

    reg d;
    wire clock;

    assign clock = scan1 ? tclk1 : 1'bz;
    assign clock = scan2 ? tclk2 : 1'bz;

    inst inst1 (.clock(clock));
endmodule
```

Simulation Tags

```
module inst (d, in1, clock);  
  
    input in1, clock;  
    output d;  
  
    reg d;  
  
    always @(posedge clock)  
        d <= in1;  
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Multiple driven signal 'clock' fed to clock/Set/Reset input of sequential element.

STARC05 Tags

This section explains the STARC05 Tags in VC SpyGlass Lint.

STARC05-1.1.1.1

Source file name should be same as the name of the module in the file. (Verilog)
File names containing entities should be <entity_name>.vhd or .vhdl (VHDL)

Description

Verilog

The STARC05-1.1.1.1 Tag flags design source files whose name is not similar to the name of any of the module in the design source file.

File names should be the same as module names described in the file to make them easily understand in debugging at simulation and in logic synthesis.

As a Tag, one source file should have only one module described. Then the file name should be *<module-name>.v*, where *<module-name>* is the name of a module.

The STARC05-1.1.1.1 Tag reports a violation in the following cases:

- If a design source file contains unrelated modules
- If a design source file contains related modules, but the name of the file does not match with the name of any of the modules in that file.

Consider an example in which a module, modA, is instantiated within another module, modB. In this case, the STARC05-1.1.1.1 Tag reports a violation, if the name of the source file is other than modA.v or modB.v

Consider another example in which two top-level modules, modC and modD, are defined in a source file, modD.v. In this case, the STARC05-1.1.1.1 Tag reports violation as these modules are not related. Here, the Tag specifies the reason of violation as multiple modules in the violation message.

Generally, IC designs contain a number of HDL source files. In such cases, using descriptive file names help in file management.

The STARC05-1.1.1.1 Tag checks only those source files that contain at least one module declaration.

The [*starc_file_ext_vlog*](#) parameter checks for the valid extension of the design source files. For example, by default it is recommended to name the design source files as *<module-name>.v* or *<module-name>.sv*, where

`<module-name>` is the name of a module in a design source file.

By default, the [ignore_file_with_multiple_modules](#) is not set and the *STARCO5-1.1.1.1* Tag reports violations for files containing multiple modules. Set this parameter to `yes` to not report violations for such cases.

VHDL

The *STARCO5-1.1.1.1* Tag flags source file names that are not based on entity names.

File names should be the same as entity names described in the file to make them easily understand in debugging at simulation and in logic synthesis.

As a Tag, one source file should have only one entity described. Then the file name should be `<entity-name>.vhd` or `<entity-name>.vhd1`, where `<entity-name>` is the name of the entity.

The *STARCO5-1.1.1.1* Tag reports a violation in the following cases:

- If a design source file contains unrelated entities
- If a design source file contains related entities, but the name of the file does not match with the name of any of the entities in that file.

Consider an example in which an entity, `entA` is instantiated within architecture of entity, `entB`. In this case, the *STARCO5-1.1.1.1* Tag reports a violation if the name of the source file is other than `entA.vhd` or `entB.vhd`.

Consider another example in which two top-level entities, `entC` and `entD`, are defined in a source file, `entD.v`. In this case, the *STARCO5-1.1.1.1* Tag reports violation as these entities are not related. Here, the Tag specifies the reason of violation as multiple entities in the violation message.

Generally, IC designs contain a number of HDL source files. In such cases, using descriptive file names helps in file management.

The *STARCO5-1.1.1.1* Tag checks only those source files that contain at least one entity declaration.

NOTE: *The *STARCO5-1.1.1.1* Tag is case-insensitive.*

The [starc_file_ext_vhdl](#) parameter checks for the valid extension of the design source files. For example, by default, it is recommended to name the design source files as `<entity-name>.vhd` or `<entity-`

name>.vhdI, where *<entity-name>* is the name of an entity in a design source file.

Message Details

Verilog

The following message appears if the source file name *<filename>* is not same as the name of any one module in the source file:

Filename '*<filename>*' does not indicate module and should be '*<module-name>.v*'

Where *<module-name>* is the name of any one module in the design file.

Also, the following message appears, if there are multiple independent or unrelated modules in the same file:

Filename '*<filename>*' does not indicate module and should be '*<module-name>.v*' reason multiple modules

Where *<module-name>* is the name of any one module in the design file.

If you set the [use_new_message](#) parameter to yes, this message is updated as follows:

Filename '*<filename>*' does not indicate module and should be '*<module-name>.v*' reason multiple independent modules/entities

VHDL

The following message appears for a Tag-violating source file *<file-name>*:

File name '*<file-name>*' is not based on the entity name *<entity-name>*

Where *<entity-name>* is the name of any one entity in the design file.

Also, the following message appears, if there are multiple independent or unrelated entities in the same file:

File name '*<file-name>*' is not based on the entity name *<entity-name>* reason multiple entities

Where *<entity-name>* is the name of any one entity in the design file.

Severity

Recommended

STARCO5-1.1.1.2

Names should follow recommended naming convention

Description

Verilog

The STARCO5-1.1.1.2 Tag flags those signal, port, module, UDP, task, function, instance, label, or variable names that:

- Start with a non-alphabetical character
- Contain characters other than alphanumeric characters and underscores

Although Verilog design names can contain escaped special characters, such names may cause problems when the design is ported to VHDL or during subsequent sTages in the design cycle.

Following the convention checked by this Tag ensures your design will encounter a minimum of naming problems across a wide range of EDA tools.

If a violated name is resulting from an expansion of a macro, whose macro name matches with any of regular expression patterns specified through the *ignore_macro* parameter, then the Tag does not report violation.

By default, the *report_typedef* parameter is set to *no* and this Tag reports violations for typedef elements, however it does not report any violations for typedef itself. If you set the *report_typedef* parameter to *yes*, the Tag also reports violations on typedefs. The Tag reports the unused typedefs as well.

By default, the *check_task_ports* parameter report violations for the input and output of tasks. Set the value of the parameter to *yes* to report such violations.

NOTE: *By default, VC SpyGlass Lint checks all types of objects. You can change the types of objects to be Tag-checked by specifying one or more of the following arguments in the PERL Tagdeck file:*

Argument	To check...
signal	Signal names
port	Port names
module	Module names

Argument	To check...
udp	UDP names
task	Task names
function	Function names
allinstance	Instance names
label	Label names
variable	Variable names

VHDL

The STARCO5-1.1.1.2 Tag flags those object names that:

- Start with a non-alphabetical character
- Contain characters other than alphanumeric characters and underscores

The STARCO5-1.1.1.2 Tag checks names of the following constructs:

entity	architecture	package	configuration
signal	port	variable	constant
attribute	generic	types	sub-types
library	function	component	process labels
instance	procedure		

While VHDL-87 did not allow use of characters (! , @ , # , \$, % , { , }) by using the Escape identifier, such escaped characters are supported in VHDL-93. However, only VHDL-87 compliant simulator may be in use or the special characters are not supported in systems used at subsequent levels. Thus, no other characters apart from alphanumeric or underscore should be used.

Following the convention checked by this Tag ensures your design will encounter a minimum of naming problems across a wide range of EDA tools.

Message Details

The following message appears at the location where a Tag-violating object name *<name>* is first encountered:

'<name>' does not follow naming convention

Severity

Recommended

STARC05-1.1.1.5

Do not use case variants of names in the same scope. (Verilog)
Do not use names that differ in case only, within the same scope (VHDL)

Description

Verilog

The STARC05-1.1.1.5 Tag flags names that are case variants of a name already encountered in the same scope.

The STARC05-1.1.1.5 Tag checks for name case variants within a specific scope i.e., the boundaries of a module, a UDP, a function, a task, or a named block. If a scope is contained in another scope (say, a named always block in a module), the names in the containing scope (module) are also checked against the names in the contained scope (named always block).

Names are not checked across parallel scopes.

The STARC05-1.1.1.5 Tag reports a violation when an outer scope variable is used inside a function or a task and the name is a case variant of an outer scope variable.

As Verilog is case-sensitive, it is legal to have a module named `FOO` and a signal `f00` in the design. However, such names hinder possible future translation of the design to VHDL as VHDL is case-insensitive and thus considers objects named `FOO` and `f00` as the same object.

Also, overuse of case variants can cause significant readability problems and can increase the possibility of outright error.

The name of a specific construct is checked against the names of some other specific (not all) constructs only, within the same scope. The following table clearly describes the constructs that should be compared against a specific language construct:

First Construct	Second Construct
signal	All the constructs in the same scope (signals / variables / ports / parameters / function/task/instance/named-block)
variable	All the constructs in the same scope (signals / variables/ports / parameters/ function/task/instance/named-block)
port	All the constructs in the same scope (signals / variables / ports / parameters/ function/task/instance/named-block)
parameter	All the constructs in the same scope (signals/ variables/ports/parameters/ function/task/instance/named-block)
module in instantiation	All modules/UDP's
instance in instantiation	All the constructs in the same scope (signals/ variables/ports/parameters/ function/task/instance/named-block/ module)
function	All the constructs in the same scope (signals/ variables/ports/parameters/ function/task/instance/named-block)
task	All the constructs in the same scope (signals/ variables/ports/parameters/ function/task/instance/named-block)
named-block	All the constructs in the same scope (signals/ variables/ports/parameters/ function/task/instance/named-block)
module	All modules/UDP's
UDP	All module/UDP's

NOTE: *The name of the constructs (signal/variable/port/parameter/instance/function/task/named-block) will also be checked against the name of the module containing the construct.*

By default, the [functask_match_global_var](#) parameter is set to yes and the STARCO5-1.1.1.5 Tag reports violation if outer scope variable is used inside a function or task. Set this parameter to no to not report such violations.

VHDL

The STARCO5-1.1.1.5 Tag flags names that are case variants of a name already encountered in the same scope.

As VHDL is case-insensitive, it is legal to refer an object named `FOO` as `f00` elsewhere in the design. However, such case variants hinder possible future translation of the design to Verilog as Verilog is case-sensitive and thus considers objects named `FOO` and `f00` as the different objects.

Thus, it is recommended to use the names within the same scope in same case. Using the names in same case also increases readability and reduces the overwrite errors.

The STARCO5-1.1.1.5 Tag checks the names used in entity body and architecture body against the names used in all the packages in the design.

Processes are considered as a separate scope and names are not checked across parallel processes.

Besides comparing names within the same scope, the following table lists the constructs on which the STARCO5-1.1.1.5 Tag compares where the scope of the two objects is also different:

First name	Second name
Instance in configuration spec	Instance in instantiation
Component	All entities
Component in configuration spec	Component declared in architecture
Entity in binding indication of configuration spec	All entities
Architecture in binding indication in configuration spec	Architecture of the entity to which that belongs to
Entity in configuration	All entities
Instance in configuration	Instance in corresponding architecture
Component in configuration	Components in corresponding architecture
Architecture in for block in configuration	Architecture of parent entity

STARCO5 Tags

First name	Second name
Entity in binding indication in configuration	All entities
Architecture in binding indication in configuration	Architecture of the entity to which that belongs to

The STARCO5-1.1.1.5 Tag reports only one violation for each case variant name. Set value of the *strict* parameter to *yes* to report violations for all case variant usage of the name.

By default, the *STARCO5-1.1.1.5* tag reports a violation for all definitions of the same names that differ in case, within the same scope. You can specify a set of constructor types using the *design_attributes* parameter so that the tag considers only those names of selected construct types.

Message Details

The following message appears at the location where a name *<name-case2>* is encountered that is a case variant of the name *<name-case1>* already found in same scope and in same or different design source file *<file-name>* at line *<num>*:

Verilog

Name '*<name-case2>*' is a case variant of name '*<name-case1>*' (file:*<file-name>*, line:*<num>*) in the same scope

VHDL

Name '*<name-case2>*' [previously '*<name-case1>*' (*<file-name>* line#*<num>*)] not used with consistent case

Severity

Mandatory

STARC05-1.1.4.1va

File name containing package(s) must end strings specified by parameter `starc2005_ext_string_vhdl`. (VHDL)

Description

The STARC05-1.1.4.1va Tag flags package files that do not follow the recommended naming convention.

By default, the STARC05-1.1.4.1va Tag flags those package files that contain one or more package declarations or package bodies and whose names do not end with `_pac.vhd.` or `_pac.vhdl`. Use the [starc2005_ext_string_vhdl](#) Tag parameter to specify your preferred suffixes/extensions.

NOTE: *You need to escape any dot (.) characters. That is, you need to use backslash (\) before a dot (.), if the exact character (.) is to be matched; otherwise, (.) will match any character.*

Using descriptive file names makes file management easier.

Message Details

VHDL

The following message appears for a Tag-violating package file `<file-name>`:

File name '`<file-name>`' does not end with
`$starc2005_ext_string_vhdl`

Severity

Recommended2

STARC05-1.1.4.4

**Macros defined outside a module must not be used in the module.
(Verilog)**

Description

The STARC05-1.1.4.4 Tag flags statements using macros that are defined outside the module.

``define` constructs used across module boundaries may lead to accidental usage. Also, it is not possible to synthesize such modules separately.

NOTE: *This Tag flags when a macro is defined locally in a module by including a file that holds the macro declaration even though it may be used only within that module.*

Message Details

The following message appears at the location in a module where a macro `<macro-name>` is used that is defined outside the module:

Macro `<macro-name>` used across module boundaries

Severity

Recommended1

STARC05-1.1.4.4v

Constants should not be redefined. (VHDL)

Description

The STARC05-1.1.4.4v Tag flags package constant redefinitions in architectures.

Redefinition of package constants in architectures can be confusing while packaging the constant declarations later.

Message Details

The following message appears at the location where a constant *<name>* is declared in an architecture *<name>* when a constant with the same name was already defined in package *<package-name>* in file *<file-name>* at line *<num>*:

Constant '*<name>*' is already declared in package '*<package-name>*' (file:*<file-name>*, line:*<num>*)

Severity

Mandatory

STARC05-1.1.4.6a

Fixed value should not be connected directly to the output ports

Description

The STARC05-1.1.4.6a Tag flags output ports of a module that are directly connected to a fixed value.

After synthesis optimization, ports that are directly connected to a fixed value may become unconnected ports. This situation may cause problems during logic equivalency checks.

The STARC05-1.1.4.6a Tag traverses from the output port of the module and ignores buffers and inverters during the fan-in traversal.

Message Details

The following message appears at the location where the port `<port-name>` is connected to a fixed value `<value>` (tied high or tied low):

Port '`<port-name>`' is '`<value>`'

where `<value>` can be `tiedhigh` or `tiedlow`.

Severity

Recommended1

STARC05-1.2.1.1a

Design should use only one edge of a clock

Description

The STARC05-1.2.1.1a tag designs that use different edges of the same clock to define separate sequential elements.

Designs using both edges of a clock signal to define separate sequential elements are difficult to analyze.

Parameter(s)

[*allow_combo_logic_base*](#): Default value is no. Set this parameter to yes to skip all VC SpyGlass Lint-generated combinational elements in the fan-in cone while finding the reset/preset source net.

Message Details

The following message appears at the location where another edge of the clock *<clk-name>* is used in the module:

Both edges of clock '*<clk-name>*' used in the design

Severity

Recommended1

Verilog Examples

Consider the following example where both positive and negative edges of clock signal CLK are used to describe flip-flops:

```
module test (OUT, IN, CLK);  
  output OUT;  
  input IN, CLK;  
  
  wire in2, reset;  
  reg OUT, out2;  
  
  always @ (posedge CLK)  
    out2 <= IN;
```

```

always @ (negedge CLK or negedge reset)
  if (!reset)
    OUT <= 1'b0;
  else
    OUT <= in2;
endmodule

```

For this example, VC SpyGlass Lint generates the following message:

Both edges of clock 'CLK' used in the design

VHDL Examples

Consider the following example where both positive and negative edges of the clock signal `clk` are used to describe flip-flops:

```

library ieee;
use ieee.std_logic_1164.all

architecture struct of test is
begin
  CLK_PROC1 : process (d_test(0), clk_test)
  begin
    if (clk_test'event and clk_test='1') then
      q_test(0) <= d_test(0);
    end if;
  end process CLK_PROC1;

  CLK_PROC2 : process (d_test(1), clk_test)
  begin
    if (clk_test'event and clk_test='0') then
      q_test(1) <= d_test(1);
    end if;
  end process CLK_PROC2;
end struct;

```

For this example, VC SpyGlass Lint generates the following message:

Both edges of clock 'clk_test' used in the design

STARC05-1.2.1.3

Remove combinational loops

Description

The STARC05-1.2.1.3 Tag flags combinational loops in the design.

Combinational loops have the following issues:

- VC SpyGlass Lint DFT tools are generally unable to generate tests for combinational loop logic.
- The timing behavior and ultimately functional behavior of a combinational loop logic is highly dependent on the target technology because its behavior depends on gate and interconnect delays. Thus, designs with combinational loops may not be portable.
- Designs with combinational loops may have reliability problems if the timing dependencies are tight enough to fall within manufacturing process variations.

NOTE: *For Verilog, this Tag lags cell-defined modules also. If there is a loop within another combinational loop, only the outer loop would be reported.*

By default, the STARC05-1.2.1.3 Tag reports SR-Latch type combinational loops. Set the [srlatch](#) Tag parameter to ignore SR-Latch type combinational loops.

By default, the STARC05-1.2.1.3 Tag reports all loops for overlapping loops. Set the [overlappingLoops](#) Tag parameter to no to report only the first found loop.

By default, the STARC05-1.2.1.3 Tag reports only first 500 messages for each top design unit. Set the [ignore_chain_cell](#) Tag parameter to yes or *<Tag-name>* to report all violation messages.

By default, the STARC05-1.2.1.3 Tag will report violation if a combinational loop exists through a multiplexer's select pin. Set the [report_mux_select](#) Tag parameter to no, and the STARC05-1.2.1.3 Tag will not report violation for combinational loops existing through a MUX's select pin.

By default, the STARC05-1.2.1.3 Tag does not flag latches implemented through a MUX; that is, a MUX in which the output converges to the input of the MUX. Set the [strict](#) Tag parameter to yes or STARC05-1.2.1.3 to report latches implemented through a MUX.

Use the [srlatch](#) and [strict](#) Tag parameters as per your design methodology.

STARC05 Tags

The [effort_level](#) parameter specifies the effort that VC SpyGlass Lint should make for checking the STARC05-1.2.1.3 Tag. There may be cases where VC SpyGlass Lint may take excessive time, if the [effort_level](#) parameter is set to a high value. If this parameter is set to a small value, then the STARC05-1.2.1.3 Tag can miss out some violations.

The STARC05-1.2.1.3 Tag supports block path analysis, i.e, it supports constant value propagation. Therefore, this Tag does not traverse on the blocked path. The [set_case_analysis](#) constraint is used to set the initial values for block path analysis.

NOTE: *Combinatorial loops may also contain synthesis tool generated nets that do not have the HDL correspondence. Their names will always be displayed as `<synth_gen_net>` so you must not try to search those nets in the HDL file.*

By default, the STARC05-1.2.1.3 Tag does not report violation if loop exists through set/reset pin and output pin of a flip-flop. Set the [report_flop_reset_loop](#) Tag parameter to yes to report such cases.

By default, the STARC05-1.2.1.3 Tag traverses through a particular net 100 times to detect a combinational loop. You can specify a different number to the [effort_level](#) Tag parameter to enable the STARC05-1.2.1.3 Tag to traverse a net that number of times.

By default, the [net_count](#) parameter is set to 10. Set this parameter to specify the number of nets of the loop that should be printed in violation message.

If the violation message contains too many nets, VC SpyGlass Lint shortens the message and displays it in the following format:

```
<net-path1>"....."<net-pathN>
```

Design Impact

Functionality, testability, and reliability

Message Details

The following message appears at the location where a combinational loop closes:

```
Combinational loop exists at '<loop-list>'
```

Severity

Mandatory

Verilog Examples

Consider the following example that has a combination loop that includes OUT[3], a, and b:

```
module test (IN1, OUT);  
    input [3:0] IN1;  
    output [7:0] OUT;  
    wire a, b;  
  
    assign a = ~b;  
    assign OUT[3] = a;  
    assign b = ~OUT[3];  
endmodule
```

For this example, VC SpyGlass Lint creates the following message:

Combinational loop exists at 'test.OUT[3]-test.b-test.a'

To fix this message, rewrite the code to remove the combinational loop.

STARC05-1.3.1.6

Signals must not be used both as asynchronous reset and synchronous reset

Description

The STARC05-1.3.1.6 Tag flags signals that are used both as synchronous reset and asynchronous reset in the same module.

Using the same signal both as asynchronous reset and synchronous reset may cause problems with logic synthesis and layout.

The STARC05-1.3.1.6 Tag reports only one violation for reset net. Set the [srp_gen_1316_report](#) Tag parameter to `yes` to generate the [srp_gen_1316_report](#), which shows the synchronous and asynchronous usage of the reset net.

NOTE: *The STARC05-1.3.1.6 Tag works with simple synchronous reset constructs only.*

Parameter(s)

- [traverse_fanin_for_src_net](#): Default value is 0. Set this parameter to 1 to stop the traversal in fanin of synchronous reset to detect flattened source reset.
- [report_local_name](#): By default, the value of the parameter is set to `yes`. It will report non hierarchical reset signal name in violation message. Set the value of the parameter to `no` to report hierarchical signal name.

Message Details

The following message appears at the location where a signal `<sig-name>` that is used both as synchronous reset and asynchronous reset in the same module, is first declared:

'<sig-name>' used as both asynchronous reset and synchronous reset

Severity

Mandatory

Verilog Examples

Consider the following example where the signal RESET is used both as synchronous and asynchronous resets in the same module:

```
module test (OUT, IN, CLK, RESET);
  input IN, CLK, RESET;
  output OUT;

  reg OUT, out2;
  wire in2, clk2;

  always @ (posedge CLK or negedge RESET)
  begin
    if (!RESET)
      OUT = 1'b0;
    else
      OUT = IN;
  end

  always @ (posedge clk2)
  if (RESET)
    out2 = 1'b0;
  else
    out2 = in2;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

'RESET' used as both asynchronous reset and synchronous reset

To fix this message, rewrite the example to use different inputs as asynchronous and synchronous resets as follows:

```
module test (OUT, IN, CLK, RESET1, RESET2);
  input IN, CLK, RESET1, RESET2;
  output OUT;

  reg OUT, out2;
  wire in2, clk2;

  always @ (posedge CLK or negedge RESET1)
```

```

begin
    if (!RESET1)
        OUT = 1'b0;
    else
        OUT = IN;
    end

always @ (posedge clk2)
    if (RESET2)
        out2 = 1'b0;
    else
        out2 = in2;
endmodule

```

VHDL Examples

Consider the following example where the signal `RST_test` is used both as an asynchronous reset and as a synchronous reset signal in the same architecture:

```

library ieee;
use ieee.std_logic_1164.all

architecture struct of test is
begin
    PROCAsync : process(D_test, CLK_test, RST_test)
    begin
        if(RST_test = '1') then
            Q_test <= '0';
        else
            if(CLK_test'event and CLK_test = '1') then
                Q_test <= D_test;
            end if;
        end if;
    end process PROCAsync;

    PROCsync : process(D_test, CLK_test, RST_test)
    begin
        if(CLK_test'event and CLK_test = '1') then

```

```
        if(RST_test = '1') then
            Q_test <= '0';
        else
            Q_test <= D_test;
        end if;
    end if;
end process PROCsync;
end struct;
```

For this example, VC SpyGlass Lint generates the following message at the location where the signal `RST_test` is declared:

'RST_test' used as both asynchronous reset and synchronous reset

STARC05-1.3.1.7

A flip-flop must not have both asynchronous set and asynchronous reset

Description

The STARC05-1.3.1.7 Tag flags those flip-flops that have both asynchronous set and asynchronous reset signals.

Typical synthesis library may not contain sequential elements with both asynchronous set and asynchronous reset inputs. Hence, inference of such device indicates a potential issue later in design process.

By default, the STARC05-1.3.1.7 Tag does not flag a violation, if the set and reset pin use logic driven by a common input. Set the *strict* parameter to *yes* or STARC05-1.3.1.7 to report such violations.

Parameters

ignore_const_zero_only: The default value of this parameter is *yes*. It enables the tag to ignore violations if set or reset pin is connected to only constant zero. It will not ignore violation when pin is connected to constant one.

Message Details

The following message appears at the start of an *always* construct that describes a flip-flop *<flop-name>* with both asynchronous set and asynchronous reset in hierarchy *<hier-path>*:

Flip-flop '*<flop-name>*' has both asynchronous set and asynchronous reset. [Hierarchy: '*<hier-path>*']

Severity

Mandatory

Tag Limitations

If both set and reset nets exist but are tied to inactive values subsequently, this condition is also reported as a message.

Verilog Examples

Example 1

Consider the following example where the flip-flop OUT_CLK has both asynchronous set signal (SET) and asynchronous reset signal (RESET):

```
module test (OUT_CLK, IN, CLK, SET, RESET);
    input IN, CLK, SET, RESET;
    output OUT_CLK;

    reg OUT_CLK;

    always@(posedge CLK or negedge SET or negedge RESET)
        if (!SET)
            OUT_CLK <= 1'b1;
        else if (!RESET)
            OUT_CLK <= 1'b0;
        else
            OUT_CLK <= IN;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Flip-flop 'OUT_CLK' has both asynchronous set and asynchronous reset. [Hierarchy: 'test']

To fix this message, remove the SET signal from the example as the purpose of both SET and RESET is similar. The rewritten example is as follows:

```
module test (OUT_CLK, IN, CLK, RESET);
    input IN, CLK, RESET;
    output OUT_CLK;

    reg OUT_CLK;

    always@(posedge CLK or negedge RESET)
        if (!RESET)
            OUT_CLK <= 1'b0;
        else
            OUT_CLK <= IN;
```

```
endmodule
```

Example2

Consider the following example where the *strict* parameter is set:

```
module m(q, d, clk, rst, a);
  input clk, rst, d, a ;
  output reg q;
  always @(posedge clk or posedge rst)
    if (rst)
      q <= a ;
    else
      q <= d ;
endmodule
```

In the above example, the STARCO5-1.3.1.7 Tag reports violation due to internal asynchronous set and reset generation.

VHDL Examples

Consider the following example where the flip-flop output1 has both asynchronous set and asynchronous reset:

```
library ieee;
use ieee.std_logic_1164.all

architecture struct of test is
begin
  process(ck, reset, set)
  begin
    if( reset='0' ) then
      output1<='0';
    elsif (set ='0') then
      output1<='1';
    elsif rising_edge(ck) then
      output1<=data1;
    end if;
  end process;
end struct;
```

For this example, VC SpyGlass Lint generates the following message:

Flip-flop 'output1' has both asynchronous set and asynchronous reset. [Hierarchy: 'test']

STARCO5-1.3.2.1b

Reset must be generated in separate module instantiated at top-level

Description

The STARCO5-1.3.2.1.b Tag flags reset signals that are not primary ports and are not generated in a separate design unit that is instantiated at top-level.

Reset generation should follow these guidelines:

1. Resets must be generated in a separate design unit that is instantiated at the top-level of the design.
2. The Reset Generation design unit should not have any logic other than the reset generation.
3. All ports of the Reset Generation design unit should be used only for reset generation.

Resets that do not follow these guidelines have the following issues:

- Timing of such resets may be very dependent on target technology and therefore reduce portability.
- Such resets create significant problems for VC SpyGlass Lint DFT tools.

Consolidating reset generation to one module that is instantiated at the top-level makes it easier to understand and control these features. This reset generation module can have hierarchy.

Parameters

REPORT_ONLY_USER_NETS: The default value of this parameter is 1. Specifies that the tag will not report violations when the reset source is not a user defined RTL net. Set the value of the parameter to 0 to enable reporting violations.

Message Details

The following message appears at the location where a Tag-violating reset signal *<sig-name>* is encountered:

Reset '*<sig-name>*' is not generated in a separate module instantiated at top-level

Consequences of Not Fixing

It may cause simulation-synthesis mismatch.

Severity

Recommended1

Verilog Examples

Consider the following example that uses a reset signal `reset` that is not generated in a separate module at top-level:

```
module test (CLK, IN, OUT);
    input CLK, IN;
    output OUT;

    reg OUT;
    reg reset;

    always @ (posedge CLK)
        reset = IN;

    always @ (posedge CLK or posedge reset)
        if (reset)
            OUT = 1'b0;
        else
            OUT = ~IN;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Reset 'test.reset' is not generated in a separate module instantiated at top-level

To fix this message, create a separate module `rstGen` for reset generation and a wrapper module `top` that instantiates both `test` and `rstGen` module as follows:

```
module top (CLK, IN, IN1, OUT);
    input CLK, IN, IN1;
    output OUT;
```

STARCO5 Tags

```
    wire reset;

    rstGen rstGen(CLK, IN, reset);
    test test(CLK, IN1, reset, OUT);
endmodule

module rstGen (CLK, IN, RESET);
    input CLK, IN;
    output RESET;

    reg RESET;

    always @ (posedge CLK)
        RESET = IN;
endmodule

module test (CLK, IN, RESET,OUT);
    input CLK, IN, RESET;
    output OUT;

    reg OUT;

    always @ (posedge CLK or posedge RESET)
        if (RESET)
            OUT = 1'b0;
        else
            OUT = ~IN;
endmodule
```

STARC05-1.3.2.2

Flip-flop outputs must not be used as reset signals

Description

The STARC05-1.3.2.2 tag flags those flip-flop outputs that are (ignoring buffers, inverters and combinational logic) used as asynchronous resets to other flip-flops.

Registered signals used as asynchronous resets cause significant problems for VC SpyGlass Lint DFT methodologies.

The [effort_level](#) parameter specifies the effort that VC SpyGlass Lint should make for checking the STARC05-1.3.2.2 Tag. There may be cases where VC SpyGlass Lint may take excessive time, if the [effort_level](#) parameter is set to a high value. If this parameter is set to a small value, then the STARC05-1.3.2.2 Tag can miss out some violations.

The STARC05-1.3.2.2 Tag supports [set_case_analysis](#), that is, it supports constant value propagation. Therefore, this Tag does not traverse on the blocked path. For black boxes, this Tag traverses further only if the [assume_path](#) constraint is specified in the SGDC file.

By default, the [ignore_pad_cell](#) parameter is not set. You can set this parameter to yes to ignore violations when reset is passing through a pad cell.

By default, the [show_connected_net](#) parameter is set to no. Set this parameter to yes, to enable the Tag to show immediate net connected to a flip-flop.

By default, the STARC05-1.3.2.2 Tag reports asynchronous resets only. You can set the [report_reset_type](#) parameter to async, sync, or all. The behavior of STARC05-1.3.2.2 Tag for these parameter values is as follows:

- **sync**: The Tag reports sync resets.
- **all**: The Tag reports violation for both asynchronous and synchronous resets.
- **async**: The Tag reports for asynchronous resets only.

By default, the [allow_combo_logic_base](#) parameter considers all VC SpyGlass Lint-generated combinational elements in the fan-in cone while finding the reset/preset source net.

Message Details

The following message appears at the first location found where the output of the flip-flop, <flop1-name> is directly used as a reset for another flip-flop, <flop2-name>:

Flip-flop output '<flop1-name>' used as <asynchronous | synchronous> reset for flip-flop '<flop2-name>'

Severity

Mandatory

Verilog Examples

Consider the following example where the flip-flop output RESET in module rstGen is used as an asynchronous reset for flip-flop OUT in module instance test_1:

```
module top (CLK, IN, IN1, OUT);
    input CLK, IN, IN1;
    output OUT;

    wire reset;

    rstGen rstGen(CLK, IN, reset);
    test test_1(CLK, IN1, reset, OUT);
endmodule

module rstGen (CLK, IN, RESET);
    input CLK, IN;
    output RESET;

    reg RESET;

    always @ (posedge CLK)
        RESET = IN;
endmodule

module test (CLK, IN, RESET, OUT);
    input CLK, IN, RESET;
```

```

output OUT;

reg OUT;

always @ (posedge CLK or posedge RESET)
  if (RESET)
    OUT = 1'b0;
  else
    OUT = ~IN;
endmodule

```

For this example, VC SpyGlass Lint generates the following message:

Flip-flop output 'RESET' used as asynchronous reset for flip-flop 'top.test_1.OUT'

To fix this message, rewrite the `rstGen` module so that the `RESET` signal is not the output of a flip-flop.

VHDL Examples

Consider the following example where the asynchronous reset of flip-flop `Q_test` is the output of flip-flop `RST`:

```

library ieee;
use ieee.std_logic_1164.all

architecture struct_test of test is
  signal RST : std_logic;
  begin
    PROCdff : process(D_test, CLK_test)
    begin
      if (CLK_test'event and CLK_test = '1') then
        RST <= D_test;
      end if;
    end process PROCdff;

    PROCsyncRst : process(D_test, CLK_test, RST)
    begin
      if (RST = '1') then
        Q_test <= '0';
      end if;
    end process PROCsyncRst;
  end struct_test;

```

```
        elsif (CLK_test'event and CLK_test='1') then
            Q_test <= D_test;
        end if;
    end process PROCsyncRst;
end struct_test;
```

For this example, VC SpyGlass Lint generates the following message:

Flip-flop output 'RST' used as asynchronous reset for flip-flop
'test.Q_test'

STARC05-1.4.1.1

Preferably get all clock nets from primary or centrally generated clock

Description

All clock nets are either master input clocks or outputs of a unique clock management instance driven by a master input clock.

For each clock pin of a flip-flop, source must be either a top level input port or an output port of a unique instance. If not, the flip-flop must be reported.

NOTE: *This Tag reports one violation per clock source.*

This Tag ignores combinatorial instances with single unblocked terminal during detection of the clock source.

NOTE: *Clocks of the design should be managed by a single clock management unit.*

NOTE: *This Tag does not report violation for the undriven clock inputs.*

Parameters

- ***clkGeneratorName***: You can use the `clkGeneratorName` Tag parameter to specify the design unit names or cell names corresponding to clock generation unit in the design.

Messages and Suggested Fix

The following message appears at the location where a clock signal `<clk-name>` is locally generated:

[RECOMMENDED] Clock '`<clk-name>`' is not generated in a separate module

How to Debug and Fix

Clock all flip-flops from primary or centrally generated clock pins.

Default Severity Label

Recommended

Tag Group

Clock

STARCO5 Tags

Reports and Related Files

No related reports or files.

STARC05-1.4.3.1c

Both clock edges should not be used in the same module

Description

The STARC05-1.4.3.1c Tag flags different clock edges of the same or different clock signal used in a module.

If you need to use both edges of a clock signal, it is recommended to separate the two edges in to different modules to minimize the chances of unforeseen dependencies on duty cycle.

Also, you should not create flip-flops using with different clock edges of different clock signals in the same module.

Parameters

- [REPORT_ALL_POSEDGE_NEGEDGE_PAIRS](#): Reports all pairs of violation between posedge and negedge modules. The default value is yes.

Message Details

The following message appears at the location where a different edge of the same or different clock is encountered in the same module *<module-name>*:

Different edges of clocks '*<clk2-name>*' (flop '*<flop2-name>*') and '*<clk1-name>*' (flop '*<flop1-name>*') used in module '*<module-name>*'

Severity

Recommended

Verilog Examples

Consider the following example where flip-flops Q1 and Q2 use posedge and negedge of the same clock signal CLK respectively:

```
module test (DIN, SET, Q1, Q2, CLK);  
    input DIN, SET, CLK;
```

STARCO5 Tags

```

output Q1, Q2;

reg Q1, Q2;
wire data;

assign data = ~DIN;

always @(posedge CLK)
begin
    if(!SET) Q1 = 1'b0;
    else Q1 = DIN;
end

always @(negedge CLK)
    Q2 = data;
endmodule

```

For this example, VC SpyGlass Lint generates the following message:

Different edges of clocks 'test.CLK' (flop 'Q1') and 'test.CLK' (flop 'Q2') used in module 'test'

Now, consider the following example:

```

module test (CLK1, CLK2,
             DIN1, DIN2, DIN3, DIN4,
             DOUT1, DOUT2, DOUT3, DOUT4);
    input CLK1, CLK2, DIN1, DIN2, DIN3, DIN4;
    output DOUT1, DOUT2, DOUT3, DOUT4;

    hier1 inst1(CLK1, CLK2,
               DIN1, DIN2, DIN3, DIN4,
               DOUT1, DOUT2, DOUT3, DOUT4);
endmodule

module hier1(CLOCK1, CLOCK2,
             IN1, IN2, IN3, IN4,
             OUT1, OUT2, OUT3, OUT4);
    input CLOCK1, CLOCK2, IN1, IN2, IN3, IN4;
    output OUT1, OUT2, OUT3, OUT4;

    reg     OUT1, OUT2, OUT3, OUT4;

```

```
always@(posedge CLOCK1)
    OUT1 = IN1;

always@(negedge CLOCK1)
    OUT2 = IN2;

always@(posedge CLOCK2)
    OUT3 = IN3;

always@(negedge CLOCK2)
    OUT4 = IN4;
endmodule
```

The posedge of clock CLK1 is used to define flip-flop DOUT1 and negedge of clock CLK2 is used to define flip-flop DOUT4. This design style is a Tag-violation as different edges of different clock signals are being used to define flip-flops. Hence, VC SpyGlass Lint generates the following message:

```
Different clock edges of clock 'test.CLK1' (flop 'DOUT1') and
clock 'test.CLK2' (flop 'DOUT4') used in module 'hier1'
```

Also, the posedge and negedge of clock CLK2 are used to define flip-flops DOUT3 and DOUT4. This design style is a Tag-violation as different edges of the same clock signal are being used to define flip-flops. Hence, VC SpyGlass Lint generates the following message:

```
Different edges of clocks 'test.CLK2' (flop 'DOUT3') and
'test.CLK2' (flop 'DOUT4') used in module 'hier1'
```

VHDL Examples

Consider the following example where the flip-flop out is described using the positive edge of the clock signal CLK1 when another flip-flop out has already been defined using the negative edge of the clock signal CLK2 in the same architecture struct:

```
library ieee;
use ieee.std_logic_1164.all
```

```
architecture struct of test is
begin
  process(in1, in2, CLK1, CLK2)
  begin
    if(CLK1'event and CLK1='1') then
      out <= in1;
    end if;
  end process;
  process(in1, in2, CLK1, CLK2)
  begin
    if(CLK2'event and CLK2='0') then
      out <= in2;
    end if;
  end process;
end struct;
```

For this example, VC SpyGlass Lint generates the following message:

Different edges of clocks 'test.CLK1' (flop 'out') and
'test.CLK2' (flop 'out') used in module 'struct'

STARC05-1.4.4.2

One hierarchical level should have a single clock only. (Verilog)

Description

The STARC05-1.4.4.2 Tag flags flip-flop descriptions triggered by different clocks in the same module.

To facilitate clock-tree synthesis during layout, it is recommended that flip-flops in a module should be driven by a single clock only.

Message Details

The following message appears at the location where the output of a flip-flop `<flop2-name>` of clock `<clk2-name>`, is first assigned when another flip-flop `<flop1-name>` of clock `<clk1-name>` was already found in the same module:

Multiple clock domains in one module. First:
'<clk1-name>'(flop: <flop1-name>), Current: '<clk2-name>'(flop: <flop2-name>)

Severity

Reference

Verilog Examples

Consider the following example that has flip-flops `out1` and `out2` triggered by different clocks (`clk1` and `clk2` respectively) described in the same module:

```
module test (in1, in2, in3, out1, out2);
    input in1, in2, in3;
    output out1, out2;

    reg out1, out2;
    wire clk1, clk2, a;

    assign clk1 = in1 & in2;
    assign clk2 = in1 & in3;
    assign a = ~clk2;
```

```
always @ (posedge clk1)
    out1 = in2;

always @ (posedge a)
    out2 = in2 & in3;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Multiple clock domains in one module. First: 'test.clk2'(flop: test.out2), Current: 'test.clk1'(flop: test.out1)

STARC05-1.6.2.1

Top-level module outputs should be directly driven by flip-flops.

Description

The *STARC05-1.6.2.1* Tag flags those outputs of a module that are not outputs of flip-flops.

By default, the *STARC05-1.6.2.1* Tag checks outputs of top-level modules only. You can also do one of the following:

- Use the SpyGlass block pragmas to select the modules to be checked. See Specifying Blocks for more details.
- If none of the modules have the SpyGlass block pragma specified, then use the *depth* Tag parameter to specify a different hierarchy level. Then, the *STARC05-1.6.2.1* Tag checks outputs of modules at the hierarchy level specified by the *depth* Tag parameter.

By default, the *STARC05-1.6.2.1* Tag allows all the buffers and single inverter between the output port and the corresponding register output port. Use the *buf_count* Tag parameter to specify the number of buffers to be allowed between the output port and the corresponding register output port.

By default, the *STARC05-1.6.2.1* Tag allows one inverter between the output port and the corresponding register output port. Use the *inv_count* Tag parameter to specify different number of inverters to be allowed.

By default, the *STARC05-1.6.2.1* Tag reports violation if the output port of a module is connected to its input through buffers. Specify the value of parameter *ignore_direct_connection* to *yes* to ignore such violations.

If a block is intended to be reusable, it must have predictable timing behavior. Outputs which are not registered have timing behavior dependent on internal delays which vary depending on target technology and process variations. Registering all outputs assures that output timing is controlled by the clock, rather than by these potentially unpredictable parameters.

Message Details

The following message appears at the location where the output port `<port-name>` that is not the output of a flip-flop is encountered:

Output port '`<port-name>`' is not driven by a flip-flop

Severity

Recommended

Verilog Examples

Consider the following example where the module output OUT is not the output of a flip-flop:

```
module test (OUT, IN);  
    input IN;  
    output OUT;  
  
    inst1 inst (OUT, IN);  
endmodule
```

```
module inst (OUT, IN);  
    input IN;  
    output OUT;  
  
    assign OUT = IN;  
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Output port 'OUT' is not driven by a flip-flop

To fix this message, rewrite the example to include a flip-flop description for output OUT as follows:

```
module test (OUT, IN, CLK);  
    input IN, CLK;  
    output OUT;  
  
    inst1 inst (OUT, IN, CLK);  
endmodule
```

```
module inst (OUT, IN, CLK);  
    input IN, CLK;  
    output OUT;  
  
    reg OUT;  
  
    always @(posedge CLK)  
        OUT = IN;  
endmodule
```

STARC05-1.6.2.2

Combinatorial Path should not span more than specified number of modules

Rule Description

The STARC05-1.6.2.2 tag flags combinational paths spanning more than specified number of design units. Depending upon the parameter, the rule can run in the following two modes:

- Set the rule [snake_length](#) parameter to any positive integer to specify and change the maximum number of design units. By default, the STARC05-1.6.2.2 rule flags combinational paths that span more than two design units. SpyGlass flags a message if the path spans more than `$snake_length` design units. All paths, irrespective of their endpoints (data pins of flip-flops/latches, primary output ports, or black box terminals), are checked.
- You can also set the [snakepath_ff](#) rule parameter to `yes` and the STARC05-1.6.2.2 rule checks only those combinational paths that end at the data pin of a flip-flop or at a primary output port.

While doing rule checking following kind of modules are skipped or considered:

1. Intermediate modules are not considered (i.e. feedthrough modules are not considered).
2. Modules where the signal is simply renamed by a continuous assignment (`assign out=in;`) are also not considered.
3. Modules where the signal is inverted (`assign out=!in;`) are considered.
4. Originating module is also taken into consideration, even if the flip-flop output goes directly to the module output without any intervening logic on the way. The same is true of the destination module.

The [effort_level](#) parameter specifies the effort that spyglass should make for checking this rule. There may be some cases where SpyGlass may take excessive time, if the `effort_level` parameter is set to a high value. If this parameter is set to a small value, then rule can miss out some violations.

The [through](#) parameter specifies the module names for which snake path should be reported. Set the `through` parameter to a comma

separated list of module names or a PERL regular expression to specify the module names for which snake path needs to be reported.

By default, the [consider_cont_assign](#) parameter is set to no and the *STARCO5-1.6.2.2* rule does not consider continuous statement or feedthrough module while checking for snake path. Set the `consider_cont_assign` parameter to yes to enable the rule to consider such cases.

By default, the [starc_through_block](#) parameter is not set. You can set the this parameter to a comma-separated list of module names or PERL regular expressions to enable the rule to report violations only for the paths going through specified modules. If multiple module names are specified, the rule reports violations only for the paths going through all the specified modules.

Set the value of the [report_full_path](#) parameter to yes to enable the rule to report the complete snake path. By default, this parameter is not set.

By default, the *STARCO5-1.6.2.2* rule reports a maximum of 200 messages per design. Set the [allviol](#) rule parameter to yes or <rule-name> to report all violation messages.

NOTE: *The STARCO5-1.6.2.2 rule-checking with the `snakepath_ff` rule parameter to yes can be CPU- and memory-intensive depending on the design complexity and this mode should be used with care.*

Message Details

The following message appears at the location of a net <net-name> where a combinational path starting from the output of flip-flop <flip-flop-name> has crossed more than \$snake_length design units (with the `snakepath_ff` rule parameter not set):

Snake Path from Input flop '<flip-flop-name>' to net '<net-name>' exceeds specified size

The following message appears at the location where the output of flip-flop <flip-flop2-name> is first set when the combinational path from output pin of flip-flop <flip-flop1-name> to the data pin of the flip-flop <flip-flop2-name> spans more than \$snake_length design units (with the `snakepath_ff` rule parameter set to yes):

STARCO5 Tags

Snake Path from Input flop '<flop-name>' to flop '<flop2-name>' exceeds specified size

The following message appears at the location where the primary output port *<port-name>* is set when the combinational path from output pin of flip-flop *<flop1-name>* to the primary output port *<port-name>* spans more than `$snake_length` design units (with the `snakepath_ff` rule parameter set to `yes`):

Snake Path from Input flop '<flop-name>' to output port '<port-name>' exceeds specified size

If [report_full_path](#) parameter is set to yes, below is the violation message, the tag reports the following message:

Snake Path from Input flop '<flop-name>' to <net,flop,output port>'<name>' exceeds specified size. Full path:<path name>

Severity

Error

STARC05-2.1.1.1v

**Package IEEE.std_logic_1164 must be included in each entity.
(VHDL)**

Description

The STARC05-2.1.1.1v Tag flags those entities that do not include IEEE package std_logic_1164.

Types std_logic and std_logic_vector, which must be used for RTL design, are defined in the IEEE package std_logic_1164.

Message Details

The following message appears at the first line of an entity declaration *<entity-name>* that does not include IEEE package std_logic_1164:

```
Package IEEE.std_logic_1164 not included in entity '<entity-name>'
```

Severity

Mandatory

STARC05-2.1.1.2

A function description must assign return values to all possible states of the function. (Verilog)

Description

The STARC05-2.1.1.2 Tag flags functions that do not have return values defined for all possible states of the function.

Some simulation tools assume the function return value to be 0 for the undefined state. The synthesis tools assume the function return value to be x for the undefined state. Thus, there is a possibility of pre- and post-synthesis simulation mismatch of the design.

By default, the Tag also checks for temporary variables. Set the [strict](#) parameter to `yes` to not check for temporary variables.

By default, the tag reports violations for multiple hierarchies. Set the value of [report_only_from_one_hierarchy](#) parameter to `yes` to suppress the hierarchy field and report only one violation per assignment.

Message Details

The following message appears at the location of a function where return values for all possible states of the function are not provided:

Return value of function '<function-name>' is not described for all possible states.[Hierarchy: '<hier-path>']

Severity

Mandatory

Verilog Examples

Example of Function with while Construct

Consider the following example where the function `Adder` does not have the logic for all possible values of signal `select`:

```
function [4:0] Adder;  
    input [3:0] first;  
    input [3:0] second;  
    input select;
```

```
begin
  while(select == 1'b0)
    Adder = first + second;
  end
endfunction
```

For this example, VC SpyGlass Lint generates the STARCO5-2.1.1.2 Tag message.

Example of Function with case Construct

Consider the following example where the function `Adder` does not have the logic for all possible values of signal `select`:

```
function [4:0] Adder;
  input [3:0] first;
  input [3:0] second;
  input select;

  begin
    case(select)
      1'b0: Adder = first + second;
      default: ;
    endcase
  end
endfunction
```

Also, the default branch does not specify the return value. For this example, VC SpyGlass Lint generates the STARCO5-2.1.1.2 Tag message.

Example of Function with if-else Construct

Consider the following example where the function `Adder` has logic for all possible values of signal `select` and hence does not violate the STARCO5-2.1.1.2 Tag:

```
function [4:0] Adder;
  input [3:0] first;
  input [3:0] second;
  input select;

  begin
```


STARCO5 Tags

```
        if(select == 1'b0)
            Adder = first + second;
        else
            Adder = 4'b0000;
        end
    endfunction
```

STARC05-2.1.2.1

Function statement should not be used for asynchronous reset line logic in an always construct for FF inference. (Verilog)

Description

The STARC05-2.1.2.1 Tag flags always constructs inferring flip-flops where function calls have been used as asynchronous reset descriptions.

When function is used for conditional expression to infer reset in if statement, it will not be synthesized correctly. Function statement should not be used for asynchronous reset line. It is safer to consider function statement only for combinational circuits, and to use a call from continuous assignment.

Message Details

The following message appears at the location where function call *<func-name>* has been used as the asynchronous reset description:

Function '*<func-name>*' used for asynchronous reset '*<sig-name>*'

Where *<sig-name>* is an input to the function call.

Severity

Mandatory

STARC05-2.1.2.1v

Use `std_logic` or `std_logic_vector` data types to describe ports of an entity. (VHDL)

Description

The STARC05-2.1.2.1v Tag flags ports of types other than `std_logic` or `std_logic_vector`.

During synthesis, ports of types other than `std_logic` or `std_logic_vector`, may be converted into type `std_logic_vector` at gate-level. If gate-level and RTL port descriptions are different, there may be failure in connection of layers.

It is recommended to describe ports of types `std_logic` or `std_logic_vector` only.

Message Details

The following message appears at the location where a port `<port-name>` of type other than `std_logic` or `std_logic_vector` is described:

Port '`<port-name>`' not described with `std_logic` or `std_logic_vector` data type

Severity

Mandatory

STARC05-2.1.2.2

**Do not use nonblocking assignment in function description.
(Verilog)**

Description

The STARC05-2.1.2.2 Tag flags the use of nonblocking assignment in function description.

Using nonblocking assignment statements in a function construct may not be supported by some synthesis tools.

Message Details

The following message appears at the location where a nonblocking assignment is encountered in a function construct:

nonblocking assignment used in function description

Severity

Mandatory

STARC05-2.1.2.2v

Use `std_logic`, `std_logic_vector`, `integer`, `Boolean`, `unsigned`, `signed`, `time`, or user-defined data type inside the architecture. (VHDL)

Description

The STARC05-2.1.2.2 Tag flags objects of types other than the recommended types.

Only objects of following data types must be used inside an architecture body:

<code>std_logic</code>	<code>std_logic_vector</code>
<code>integer</code>	<code>Boolean</code>
<code>unsigned</code>	<code>signed</code>
<code>user-defined data type</code>	<code>time</code>

By default, the STARC05-2.1.2.2 Tag does not flag objects of data type `time`. Set the [strict](#) Tag parameter to flag objects of data type `time` also.

Message Details

The following message appears at the location where an object `<obj-name>` of type other than the recommended types is described:
object '`<obj-name>`' not described with any of the allowed types

Severity

Mandatory

STARC05-2.1.2.3

Global signals must not be read in a function description. (Verilog)

Description

The STARC05-2.1.2.3 Tag flags global signals read directly inside a function description.

A variable that is outside the scope of a function must not be read inside the function construct as changes in the variable outside the function may result in unknown function behavior.

Message Details

The following message appears at the location where a global variable `<var-name>` is read inside a function construct:

Function reads the global variable '`<var-name>`'

Severity

Mandatory

STARC05-2.1.2.4

task constructs should not be used in the design. (Verilog)

Description

The STARC05-2.1.2.4 Tag flags task calls in the design.

NOTE: *The STARC05-2.1.2.4 Tag does not flag task constructs that are defined but not called.*

By default, the STARC05-2.1.2.4 tag reports LRM-allowed system tasks. Set the value of the [ignore_system_tasks](#) parameter to no to ignore LRM-allowed system tasks.

By default, the STARC05-2.1.2.4 tag reports violations for the user-defined tasks. Set the value of the [ignore_user_defined_tasks](#) parameter to yes to ignore violations for the user-defined tasks.

Message Details

The following message appears at the location where a task `<task-name>` is called:

Task '`<task-name>`' used in the design

Severity

Recommended1

STARC05-2.1.2.4v

Do not use bit and bit_vector data types in the design. (VHDL)

Description

The STARC05-2.1.2.4v Tag flags object declarations of data types bit or bit_vector.

The X or Z values of objects of type std_logic would become 0 when converted to bit data type. Therefore, such types must not be used for describing data objects.

Message Details

The following message appears at the location where an data object `<obj-name>` of data type bit or bit_vector is described:

Object '`<obj-name>`' uses bit or bit_vector data type

Severity

Mandatory

STARCO5-2.1.2.5

Clock edges should not be used in a task description. (Verilog)

Description

The STARCO5-2.1.2.5 Tag flags edge description used in a task description.

An always construct that calls a `task`, which has an edge statement (of format `@(posedge/negedge . . .)`) is not synthesizable.

Message Details

The following message appears at the location where a clock edge (`posedge` or `negedge`) of a clock `<clk-name>` is used in a task description:

Clock edge 'posedge | negedge <clk-name>' used in task

Severity

Mandatory

Verilog Examples

Consider the following example where the task `myTask` has an event control description using the `posedge` of signal `clk`:

```
module test(in1,in2,out1);
  input in1, in2;
  output out1;

  reg out1;
  wire clk;

  always @(posedge clk)
  begin
    myTask(out1, in1, in2);
  end

  task myTask;
    output o1;
```

```
input i1, i2;

begin
  @(posedge clk)
    o1 = i1 & i2;
end
endtask
endmodule
```

For this example, VC SpyGlass Lint generates the following message at the location of the event control description:

clock edge 'posedge clk' used in task

STARC05-2.1.2.5v

Do not use attribute enum_encoding. (VHDL)

Description

The STARC05-2.1.2.5v Tag flags enum_encoding attribute used in the design.
The enum_encoding attribute is not supported by most synthesis tools.

Message Details

The following message appears at the location where the enum_encoding attribute is encountered:

Attribute ENUM_ENCODING used in the design

Severity

Mandatory

Examples

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all

library IEEE;
use ieee.std_logic_1164.all;

library synopsys;
use synopsys.ATTRIBUTES.all;

entity ent is
  port (
    p : in bit;
    q : out bit );
end ent;

architecture structure_ent of ent is
  attribute ENUM_ENCODING of p : signal is "INPUT_P";
```

```
begin
  process(p)
  begin
    if (p'ENUM_ENCODING = "INPUT_P") then
      q <= p;
    end if;
  end process;
end structure_ent;
```

In the above example, the *STARCO5-2.1.2.5v* Tag reports a violation because the `enum_encoding` attribute is used in the design. The following message is reported by this example:

Attribute `ENUM_ENCODING` used in the design

STARC05-2.1.3.1

Ensure that the bit-width of the function arguments match the bit-width of the corresponding function inputs.(Verilog)

When to Use

Use this Tag to identify the bit-width mismatch between the function inputs and the corresponding function arguments.

Description

The *STARC05-2.1.3.1* Tag flags the bit-width mismatch between the function input and corresponding function call argument.

If the function input is narrower than the function call argument, the function call argument value is left-truncated and the function is evaluated on the truncated value.

If the function input is wider than the function call argument, the function call argument value is left-padded with zeros and the function is evaluated with the padded value.

The *STARC05-2.1.3.1* Tag does not report violation for the expression, where the width of the constant expression, constant integer, or base number is less than or equal to the width of the function call argument.

For example:

```
function func;
    input [3:0] in1_func;
    input in2_func;
    begin
        func = in1_func[0] & in2_func;
    end
endfunction

assign out_lower = func (5, in2_lower);
assign out_lower = func (20, in2_lower);
```

If function call argument is a shift expression, then the behavior is as follows:

- If the right operand of shift expression is constant integer or based number, then the violation is not reported, in case the shifted or non-shifted width matches the function input.

- If the right operand of shift expression is variable, then violation is not reported, in case the non-shifted width matches the function input.

The behavior mentioned above is explained in the following example:

```
module top(in1_3b,in1_4b,out1_3b,in1_1b,in1_2b);
input [2:0] in1_3b;
input [3:0] in1_4b;
input [0:0] in1_1b;
input [1:0] in1_2b;
output [2:0] out1_3b;

    assign out1_3b = func(in1_1b<<3); //violation
    assign out1_3b = func(in1_2b<<1);
    assign out1_3b = func(in1_3b<<1);
    assign out1_3b = func(in1_4b>>1);

    assign out1_3b = func(in1_4b>>1'b1);
    assign out1_3b = func(in1_3b>>2'b11);
    assign out1_3b = func(in1_1b<<2'b10);

    assign out1_3b = func(in1_1b<<in1_2b); //violation
    assign out1_3b = func(in1_3b>>in1_1b);

    function [2:0] func;
        input [2:0] in1;
        func = in1;
    endfunction
endmodule
```

The behavior remains the same for shift expression when parameter [nocheckoverflow](#) is set to yes

NOTE: The STARCO5-2.1.3.1 Tag supports generate-for block.

NOTE: The Tag supports the multi-line functionality.

Default Weight

5

Language

Verilog

Parameter(s)

- ***check_pad_concat***: The default value is no. the *STARC05-2.1.3.1* Tag ignores leading zeros in concat/multi-concat operator while width calculation. Set the value of the parameter to yes or <Tag-name> to consider leading zeros in concatenation/multi-concatenation operator while width calculation.
- ***check_shifted_width***: The default value is no. When you set the value of the parameter to yes, the non-shifted width of the function argument is not compared with the function input width to filter the violation.
- ***ignore_param_type***: The default value is no. Set the value of the parameter to yes to ignore the specified type for parameter and take natural width if value of parameter is a constant integer.
- ***use_lrm_width***: The default value is no. Set the value of the parameter to yes to consider the LRM width (32 bits) for integer constants. However, for constants, natural width ($\log_2(N)+1$) is considered.
- ***nocheckoverflow***: Width of function argument is calculated as per the following methods:
 - ❑ If the *nocheckoverflow* parameter is set to yes, the width is calculated as per LRM. However for constants, the natural width is considered.
 - ❑ If the *nocheckoverflow* parameter is set to no, the width is calculated according to following methods:
 - ◆ For plus/minus operator, the value based width is considered. For example:


```
foo (.in1(a[2:0]+b[2:0])) ; //Width 4 (7+7=14, 4 bits wide)
foo (.in1(a[2:0]-b[2:0])) ; //Width 4 (7+7=14, 4 bits wide)
```
 - ◆ For multiplication operator, width is considered to be the sum of operand widths. For example:


```
foo (.in1(a[2:0]*b[2:0])) ; //Width 6
```
 - ◆ For division operator, the LHS width is considered. For example:


```
inst IN1 (.in1(a[2:0]/b[2])) ;//Width 3
```

- ♦ For concatenation operator, the width is calculated as sum of all the operands. For example:

```
foo (.in1({1'b0,a[2:0]})) ; //Width=3 bits
(leading 0
 ignored)
foo (.in1({1'b1,a[2:0]})) ; //Width=4 bits (1+3
bits)
```

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location of a function call where there is bit-width mismatch between the function call argument *<argu-name>* of *<bit-width1>* and the corresponding function input *<input-name>* of *<bit-width2>*:

[MANDATORY] Bit-width mismatch between function call argument '*<argu-name>*' (*<bit-width1>* bits) and function input '*<input-name>*' (*<bit-width2>* bits). [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the hierarchical name of the instance (in *<module-name>:<inst-name>@<master-module-name>* format) where the Tag-violating construct is encountered.

Potential Issues

A violation is reported when there is a bit-width mismatch between the function call argument and the corresponding function input.

Consequences of Not Fixing

If the function input is narrower than the function call argument, the function call argument value is left-truncated and the function is evaluated on the truncated value.

If the function input is wider than the function call argument, the function call argument value is left-padded with zeros and the function is evaluated with the padded value.

This leads to inaccurate results and an unexpected code behavior.

How to Debug and Fix

The violation message shows the argument for which there is width mismatch.

Double-click the violation message. The HDL Viewer window highlights the line of the function call. Locate the function definition using one of the following ways.

- Scroll through the HDL Viewer window to search for the function definition.
- Open the HDL in an editor, and search for the function definition.
- Search in the design for the function definition.

To fix the violation, compare the widths of violating function call argument and the function input.

Example Code and/or Schematic

Example 1

Consider the following example where the function call argument `in1_lower` is a one-bit signal and the corresponding function input argument `in1_func` is a 3-bit signal thus leading to a bit-width mismatch:

```
module test (out, in1, in2);
    output out;
    input in1, in2;

    lower lower (out, in1, in2);
endmodule

module lower (out_lower, in1_lower, in2_lower);
    output out_lower;
    input in1_lower, in2_lower;

    assign out_lower = func (in1_lower, in2_lower);

    function func;
        input [2:0] in1_func;
        input in2_func;
```

```
begin
    func = in1_func[0] & in2_func;
end
endfunction
endmodule
```

For this example, VC SpyGlass Lint generates the following message at the location of the function call:

Bit-width mismatch between function call argument 'in1_lower' ('1' bits) and function input 'in1_func' ('3' bits).
[Hierarchy: 'test:lower@lower']

Example 2

Consider the following example:

```
wire a,b;
    assign b = func({2'b00,a});
```

In the above example, the default width for {2'b00,a} is 1 bit and when the [check_pad_concat](#) parameter is set to yes, the width becomes 3 bits.

Default Severity Label

Mandatory

Tag Group

quality1, RTL Description Techniques

Reports and Related Files

No related reports or files.

STARC05-2.1.3.4

Return value assignment statement must be the last statement in function description. (Verilog)

Description

The STARC05-2.1.3.4 Tag flags statements in a function description that follow the last return value assignment statement.

As the statements that follow the last return value assignment statement are ignored, such statements can be confusing.

Message Details

The following message appears at the first line of a function description when other statement(s) follow the last return value assignment statement:

Last statement in a function is not the return value assignment statement

Severity

Recommended1

Verilog Examples

Consider the following example where another statement follows the last function return value assignment statement:

```
function [4:0] Adder;
  input [3:0] first;
  input [3:0] second;
  input select;

  reg done;
  begin
    if(select == 1'b1)
      Adder = first + second;
    done = 1'b1;
  end
endfunction
```

For this example, VC SpyGlass Lint generates the following message at the

first line of the function description:

Last statement in a function is not the return value assignment
statement

STARC05-2.1.3.4v

Input port must not be described with initial value. (VHDL)

Description

The STARC05-2.1.3.4v Tag flags in type ports in entity declarations that have initial values specified.

When the layer where initial value of an in type port is described and its upper layer are synthesized separately, there may be connection issues.

By default, the STARC05-2.1.3.4v Tag checks the in type port declarations in entity declarations only. Use the [strict](#) Tag parameter to check in type port declarations of block constructs also.

Message Details

The following message appears at the location where an in type port *<port-name>* is declared with an initial value:

IN port '<port-name>' described with initial value

Severity

Mandatory

STARC05-2.1.4.1

**Parenthesis should be used in expressions with multiple operators.
(Verilog)**

Description

The STARC05-2.1.4.1 Tag flags complex expressions using multiple operators that do not use parenthesis.

In complex expressions using multiple operators, it is recommended to use parenthesis to force the required order of execution.

Use of parenthesis also increases the readability of expressions and reduces the possibility of errors.

Message Details

The following message appears at the location where a complex expression using multiple operators does not have correct parenthesis:

Use parenthesis in complex expressions <opr>.

Where <opr> is the first offending operator found.

Severity

Recommended

STARC05-2.1.4.5

Use bit-wise operators instead of logic operators in multi-bit operations.

When to Use

Use this Tag to detect logical operators used on vector operands.

Description

The STARC05-2.1.4.5 Tag reports violation for logical operators that are used on vector operands.

No violation is reported for logical negation on integer variables.

Default Weight

5

Language

Verilog

Parameter(s)

use_secondary_message: By default, the parameter is set to `no` and the tag uses the old message format to report violations. Set this parameter to `yes` to enable the tag to report the updated message that includes '*Avoid using logical operator '<Logical_Operator>' in expression '<Expression>'*' to report violations.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a logical operator *<op-name>* is used for a vector operand:

[MANDATORY] Use bit-wise operator instead of logical operator '*<op-name>*'

Potential Issues

The logical operations on vector operands have single-bit results (may be

incorrect results) while bit-wise operations on vector operands are vectors (correct results).

Consequences of Not Fixing

Use of logic operators in multi-bit operations results in single-bit results, which may not be the expected result.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the location where the logic operator is used in the multi-bit operation.

To fix the violation, use bit-wise operators instead of logic operators in multi-bit operations.

Example Code and/or Schematic

Consider the following example:

```
reg [7:0] A,X,Y,Z;  
always@(A) begin  
  if((X || A)==1'b1)  
    Z=X && Y;  
end
```

In above example, the STARCO5-2.1.4.5 Tag flags violation for logical operators `||` and `&&` as the operations are performed on multi-bit signals.

Default Severity Label

Mandatory

Tag Group

Combinational Logic

Reports and Related File

No related reports or files.

STARC05-2.1.5.1

Conditional operators (?:) should not be nested. (Verilog)

Description

The STARC05-2.1.5.1 Tag flags nesting in conditional operator (?:) expressions.

By default, the STARC05-2.1.5.1 Tag flags all nested conditional operator (?:) expressions.

You can set the [nesting_level](#) Tag parameter to any positive integer number. Then, the STARC05-2.1.5.1 Tag flags only those conditional operator (?:) expressions where the nesting level is more than the specified number.

Using nesting in conditional operator expression can be confusing and may cause coding errors. It is recommended to use if-else constructs instead of nested conditional operators.

Message Details

The following message appears at the location where the conditional operator is nested more than one level:

Nested conditional operators used

Severity

Recommended3

Verilog Examples

Some examples of nested conditional operators that violate the STARC05-2.1.5.1 Tag are as follows:

```
assign out = en1?en2?1'b1:in:1'b0;
assign out = en1?1'b0:en2?1'b1:in;
out = en1?in1:(en2?1'b1:(en3?in2:1'b0));
```

STARC05-2.1.5.3

Ensure that conditional expressions evaluate to a scalar.

When to Use

Use this Tag to report conditional expressions used in if constructs and with conditional operators that evaluate to a non-scalar value.

Description

The STARC05-2.1.5.3 Tag flags conditional expressions used in if constructs and with conditional operators that evaluate to a non-scalar value.

The conditional expression used in if constructs and with conditional operators should always result in TRUE or FALSE result.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where the conditional expression used in an if construct or with a conditional operator does not result in a single-bit result:

[RECOMMENDED] Conditional expression ('<Expression>') does not evaluate to a scalar [Hierarchy: '<Hierarchy>']

Potential Issues

Violations may arise when a conditional expression evaluates to a non-scalar value.

Consequences of Not Fixing

If the conditional expressions do not evaluate to a scalar value, the code may behave unexpectedly.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where a conditional expression is not evaluated to a scalar value.

To fix the violation, ensure that the conditional expression used in if constructs and with conditional operators evaluates to TRUE or FALSE result.

Example Code and/or Schematic

Consider the following example where the condition expression (en1) of the if-else construct and the condition expression of condition operator (en2) do not evaluate to a scalar:

```
module test(out1, out2, in1, in2, in3, in4, en1, en2);
    input in1, in2, in3, in4;
    output out1, out2;
    input [1:0] en1, en2;

    reg out1, out2;

    always @ (en1 or in1 or in2)
        if (en1)
            out1 <= in1;
        else
            out1 <= in2;
    always @ (en2 or in3 or in4)
        out2 <= (en2) ? in3 : in4;
endmodule
```

For this example, VC SpyGlass Lint generates the following message at the first line of the if-else construct and at the start of the assignment statement using the conditional operator:

[RECOMMENDED] Conditional expression does not evaluate to a scalar

Default Severity Label

Recommended

Tag Group

Combinational Logic

Reports and Related File

No related reports or files.

STARC05-2.1.6.1v

Specification of one-dimensional array should be downto (VHDL)

Description

The STARC05-2.1.6.1v Tag flags one-dimensional array specifications that are not in (MSB downto LSB) format.

Following downto direction to specify one-dimensional arrays eliminates the possibility of mixed orders (some high-bit first, some low-bit first) and the possibility of connecting buses in reverse order.

It is, therefore recommended to use the (MSB downto LSB) when declaring one-dimensional arrays.

The STARC05-2.1.6.1v Tag ignores one-dimensional arrays of array as they are considered as multi-dimensional arrays.

Message Details

The following message appears at the location where an array *<array-name>* is not specified in (MSB downto LSB) format:

Declare array '*<array-name>*' as (MSB downto LSB)

Severity

Recommended3

STARC05-2.1.6.2

LSB of vectors/memory should be zero. (Verilog)
LSB of one-dimensional arrays should be zero. (VHDL)

Description

Verilog

The STARC05-2.1.6.2 Tag flags vector or memory specifications where LSB is non-zero.

Working with vectors or memories with a non-zero LSB can be complicated and may cause coding errors.

In case of memory, the STARC05-2.1.6.2 Tag checks both memory size and memory index. For example, this Tag will report violation in the following cases:

```
reg [7:0] memory [0:15] //memory size [0:15] should
                        //be [15:0]
reg [7:1] memory [15:0] //index specification [7:1]
                        //should be [7:0]
```

By default, the STARC05-2.1.6.2 Tag ignores multidimensional arrays and memory. Set the value of the [strict](#) Tag parameter to `yes` or STARC05-2.1.6.2 to check multidimensional arrays and memory also.

VHDL

The STARC05-2.1.6.2 Tag flags one-dimensional array specifications where the LSB is non-zero.

Working with arrays with a non-zero LSB can be complicated and may cause coding errors.

Message Details

Verilog

The following message appears at the location of vector/memory specification `<vector/memory-name>` where the LSB is non-zero:

Declare vector/memory '`<vector/memory-name>`' as `[MSB:0]`

VHDL

The following message appears at the location where an array `<array-`

STARCO5 Tags

name> is specified with a non-zero LSB:

Non-zero value used as LSB of array '<array-name>'

Severity

Recommended3

STARC05-2.1.6.5

Do not use x and z for an array index

When to Use

Use this Tag to detect array indexes using x and z characters.

Description

The STARC05-2.1.6.5 Tag reports violation for the array indexes using x and z characters.

Default Weight

10

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where an array index containing x and z characters:

[MANDATORY] For an array index x and z should not be used

Potential Issues

Violation may arise when x or z are used as an array index.

Consequences of Not Fixing

Using x and z for array indexes may not cause any error during simulation. However, it may cause an error with logic synthesis tool and an incorrect circuit may get generated.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the

STARCO5 Tags

line where a multiplier larger than the recommended size is described using the multiplication operator.

Scroll through the HDL Viewer window to check for the multiplication operator usage. The size of the multiplier calculated using the operator is more than the size specified by the Tag parameter.

To fix the violation, avoid using x and z as an array index.

Example Code and/or Schematic

Consider the following example:

```
module test(in1, in2, clk1, clk2, out1,out2);
  input[1:0] in1, in2, clk1, clk2;
  output out1, out2;
  reg[1:0] out1, out2;
  wire [1:0] a;
  assign a = in1[1'bx];
  always
  begin
    @ (posedge clk1)
      out1[1'bz] <= in2;
    @ (posedge clk2)
      out2[1'bx] <= ~in1;
  end
endmodule
```

In the above example, the STARCO5-2.1.6.5 Tag reports a violation as x and z are used for an array index.

Default Severity Label

Mandatory

Tag Group

quality1, RTL Description Technique

Reports and Related Files

No related reports or files.

STARCO5-2.2.1.2

Design should not have latches

Description

The STARCO5-2.2.1.2 Tag flags the latches inferred in the design.

Except where explicitly intended, latch inference is usually caused by a coding oversight like failure to define the default value for a variable defined in a conditional statement and may lead to incorrect behavior in a register-based design. Also, inferred latches in a register-based design may cause problems for the VC SpyGlass Lint DFT tools.

By default, this Tag does not report latches inferred from both .sglib and .lib libraries. Set the [reportLibLatch](#) parameter to `yes` to report latches inferred from libraries.

Message Details

The following message appears at the location where a latch has been inferred for the signal `<sig-name>` in module `<mod-name>`:

Latch inferred for signal '`<sig-name>`' in module '`<mod-name>`'

Severity

Reference

Verilog Examples

Consider the following example where a latch is inferred for `OUT[0]` as the case construct branch for `EN=2'b11` has not been defined:

```
module test (OUT, IN, EN);
  output [1:0] OUT;
  input [1:0] IN, EN;

  reg [1:0] OUT;

  always @ (EN or IN)
    case (EN)
      2'b00 : OUT[0] <= 1'b1;
      2'b01 : OUT[0] <= 1'b0;
```

```
        2'b10 : OUT[0] <= IN[1];  
    endcase  
endmodule
```

This example would have worked if there had been a default `case` branch. However, the `case` construct also does not have a default branch.

For this example, VC SpyGlass Lint generates the following message at the first branch of the `case` construct:

Latch inferred for signal 'OUT[0]'

To fix this message, either add the `case` branch for `EN=2'b11` or add the default branch.

STARC05-2.3.1.1

Blocking assignment must not be used in sequential always blocks. (Verilog)
Variable assignment statement must not be used in flip-flop description (VHDL)

Description

Verilog

The STARC05-2.3.1.1 Tag flags blocking assignments used in sequential always constructs to infer flip-flops or latches.

Use of blocking assignments in sequential always constructs may cause RTL logic simulation behavior and post-synthesis simulation behavior to differ. A blocking assignment must complete before the next behavioral step (for example, another assignment) is evaluated. However, assignments in a sequential always construct are normally synthesized as parallel logic that evaluates simultaneously.

The STARC05-2.3.1.1 Tag flags only those blocking assignments in an always construct which infer sequential logic (flip-flop or latch) in the design.

VHDL

The STARC05-2.3.1.1 Tag flags variable assignment statements in flip-flop description.

Inferring a flip-flop using variable assignment statement is not supported by some FPGA logic synthesis tools. Therefore, variable assignment must not be used for flip-flop descriptions and it should be used only where combinational logic is to be generated.

Message Details

Verilog

The following message appears at the location where a blocking assignment statement is encountered in a sequential always construct:

```
Do not use blocking assignments in sequential blocks
```

VHDL

The following message appears at the location where a variable assignment statement is used in a flip-flop description:

```
variable assignment statement used in flip-flop description
```

STARCO5 Tags

Severity

Mandatory

STARC05-2.3.2.2v

Variable assignment statement should not be used in a process block. (VHDL)

Description

The STARC05-2.3.2.2v Tag flags variable assignment statements in process constructs.

Using variables in process constructs is not recommended where flip-flop is generated since description method might go wrong if the order of assignment is not proper.

Therefore, it is recommended not to use variable assignment statement in process constructs.

Message Details

The following message appears at the location where a variable assignment statement is encountered in a process construct:

```
variable assignment used in process
```

Severity

Recommended1

STARC05-2.4.1.4

Avoid combinational loops that contain latches

When to Use

Use this Tag to detect combination loops that contain latches.

Description

The STARC05-2.4.1.4 Tag reports violation for combinational loops containing latches.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [*strict*](#): The default value is no. Set the value of the parameter to yes to report combinational loops through a multiplexer's select pin or through a flip-flop's set/reset pin.
- [*overlappingLoops*](#): The default value is yes. Set the value of the parameter to no to report only the first found combinational loop.
- [*effort_level*](#): The default value is 100. Set the value of the parameter to any positive integer value to enable the STARC05-2.4.1.4 Tag to traverse a net specified number of times.

Constraint(s)

- [*set_case_analysis*](#): The Tag performs combinational blocking and non-blocking path analysis when some constant value is propagated to the terminal of a combinational instance (for example, AND/OR gate) through `set_case_analysis` or supply net.

NOTE: *The Tag does not perform any blocking and non-blocking path analysis for sequential logic.*

Messages and Suggested Fix

The following message appears at the location where a latch is found within

a combinational loop:

[MANDATORY] Combinational loop containing latch exists at '<path>'

Where <path> is a hyphenated list of objects in the combinational loop.

Potential Issues

Violation is reported when an output of a cell loops back onto its inputs through one or more latches and other combinational circuits.

Consequences of Not Fixing

Feedback loops that contain latches create paths whose timing is difficult to analyze, much like combinational circuits. In fact, even if it were assumed that they were false paths and merely feedback, a timing analysis tool would not understand them. Such circuits may be acceptable on behavior but create problems in timing analysis and should not be used.

How to Debug and Fix

The violation message lists the elements forming the combinational loop.

View the Incremental Schematic of the violation message. The Incremental Schematic shows the loop formed with a few elements.

To fix the violation, search a place to break the combinational loop with a flip-flop, unless the loop is introduced by mistake.

Example Code and/or Schematic

Verilog

Consider the following example:

```
module m4 (in, out);

    input in;
    output out;
    reg a;
    wire b, c;

    assign c = ~in;
    assign out = (a & b) | c;
    always
    if(c)
```


STARC05 Tags

```
a = ~out;
```

```
endmodule
```

In the above example, the STARC05-2.4.1.4 Tag reports a violation as a combinational loop containing latches exist in the design.

VHDL

Consider the following example:

```
library IEEE;
use ieee.std_logic_1164.all;
```

```
entity my is
port(a,b,c,d :in bit; out1 : out bit);
end my;
```

```
architecture arc of my is
begin
process (d)
begin
if(a='1') then
out1 <= ((a and b) or c) xor d;
end if;
end process;
end arc;
```

```
entity top is
port( b,c,d :in bit; output : out bit);
end top;
```

```
architecture arc of top is
```

```
component my
port(a,b,c,d :in bit; out1 : out bit);
end component;
```

```
signal s1:bit;
begin
```

```
INST1: my port map (a=>c,b=>b,c=>s1,d=>d,out1=>s1);  
output <= s1;  
end arc;
```

In the above example, the STARCO5-2.4.1.4 Tag reports a violation as a combinational loop containing latches exist the design.

Default Severity Label

Mandatory

Tag Group

RTL Description Technique

Reports and Related Files

No related reports or files.

STARC05-2.10.1.3

Comparison should not be made with X, H, L, U, W, - or Z. (VHDL)

Description

The STARC05-2.10.1.3 Tag flags comparisons with X, H, L, U, W, -, or Z. The STARC05-2.10.1.3 Tag checks for signal comparisons in the following types of constructs:

- if-elsif condition
- while-loop condition
- Conditional signal assignment condition
- case statement choices
- selected signal assignment choices
- assert statement condition
- next statement condition
- exit statement condition
- return statement condition

The comparison with X, Z, U, -, W, H or L is evaluated by RTL simulation but the comparison with X is ignored by logic synthesis tool. The comparison with X, Z, U, -, W, H or L should be avoided.

Consider the following example:

```
If (A = "XXX") then
    Q <= 'X';
else
    Q <= '0';
end if;
```

Here, if the value A is XXX, it cannot be the cause of results differing between RTL and gate level simulations because 'X' is propagated to Q output. However, if RTL simulation initial value 'U' is input in the above description, else item is executed. The above description is not realistic to match RTL and gate level simulations completely and can lead to confusion. Therefore, it should be avoided.

Message Details

The following message appears at the location where a signal is compared with X, H, L, U, W, -, or Z:

Comparison made with X/Z/H/L/U/W/-

Severity

Recommended

STARC05-2.10.1.4a

Signals must not be compared with X or Z. (Verilog)

Description

The STARC05-2.10.1.4a Tag flags signal comparisons with X (unknown) or Z (high impedance). That is, the Tag flags violation when the value being compared contains pure X or Z values.

For example, consider the following scenario:

```
wire [2:0] w1;  
if (w1 == 3'bxxx)  
if (w1 == 3'b1x0)
```

In the above example, the Tag flags a violation as the value, 3'bxxx, contains all X values.

Comparison with X or Z is allowed in simulation but is treated as false during synthesis. Thus, the pre- and post-synthesis simulation results may not match.

Parameters

[*check_any_bit_for_xz*](#): Default value is no. Set the value of the parameter to yes to enable the *STARC05-2.10.1.4a* Tag to report violations for the cases where x and z have other bits also.

Message Details

The following message appears at the location where a signal is compared with X or Z:

signal compared with '<type>'

Where <type> is either X or Z respectively.

Severity

Mandatory

STARC05-2.10.1.4b

Signals must not be compared with values containing X or Z. (Verilog)

Description

The STARC05-2.10.1.4b Tag flags signal comparisons with values containing X (unknown) or Z (high impedance). That is, the Tag flags violation when the value being compared contain X or Z values.

For example, consider the following scenario:

```
wire [2:0] w1;  
if (w1 == 3'bxxx)  
if (w1 == 3'b1x0)
```

In the above example, the Tag flags a violation for both type of values, pure X or Z values and partial X or Z values.

Comparison with X or Z is allowed in simulation but is treated as false during synthesis. Thus, the pre- and post-synthesis simulation results may not match.

By default, the tag reports violation for case equality comparisons (==, !=). Set the value of the [ignore_case_compare_op](#) parameter to ignore violations for such case equality comparisons.

Message Details

The following message appears at the location where a signal is compared a value containing Xs or Zs:

signal compared with value containing x or z

Severity

Mandatory

STARC05-2.10.1.4c

Signals must not be compared with values containing X or Z in case statement. (Verilog)

Description

The STARC05-2.10.1.4c Tag flags signal comparisons with values containing X (unknown) or Z (high impedance) in case constructs.

Comparison with X or Z is allowed in simulation but is treated as false during synthesis. Thus, the pre- and post-synthesis simulation results may not match.

By default, the tag reports violation for case inside statements. Set the value of the [ignore_case_inside](#) parameter to yes to ignore violations for case inside statements.

Message Details

The following message appears at the location where a signal is compared a value containing Xs or Zs in a case construct:

```
signal compared with value containing x or z
```

Severity

Mandatory

STARC05-2.10.1.4v

Do not assign 'X' except for the others choice of case statements (VHDL)

Description

The STARC05-2.10.1.4v Tag flags signal assignments to X (unknown) except in the OTHERS clause of case constructs.

In situations except the OTHERS clause of case constructs, assigning X value is treated as don't-care and logic synthesis tools generate circuit as unknown value. This would lead to a simulation results mismatch between RTL and gate-level. As inconsistencies can occur in RTL and at gate-level with the propagation of X, this kind of description is risky and should be avoided.

Message Details

The following message appears at the location where a signal is assigned X value:

'X' is assigned in the assignment statement

Severity

Mandatory

STARC05-2.10.1.5

Do not use values including 'X','Z','U','-','W','H','L'. (VHDL)

Description

The STARC05-2.10.1.5 Tag flags values containing X, H, L, U, W, -, or Z. Such values is allowed in simulation but is ignored during synthesis. Thus, the pre- and post-synthesis simulation results may not match.

For example, the following construct is ignored during synthesis:

```
if(sig = "1xx1") then  
...
```

Message Details

The following message appears at the location where a constant containing X, H, L, U, W, -, or Z is used:

Constant including X/Z/U/W/H/L/- used

Severity

Mandatory

STARC05-2.10.1.6

Constants must not contain X or Z. (Verilog)

Description

The STARC05-2.10.1.6 Tag flags constants containing X (unknown) or Z (high impedance).

NOTE: *The STARC05-2.10.1.6 Tag does not flag constants containing X (unknown) or Z (high impedance) if they are used in the default clause of a case construct in Verilog designs.*

Constants containing Xs or Zs are allowed in simulation but are ignored during synthesis. Thus, the pre- and post-synthesis simulation results may not match.

No Tag checking is done for unused macro definitions and unused parameters.

Message Details

The following message appears at the location where a constant containing Xs or Zs is declared:

Constant containing x or z used

Severity

Mandatory

STARC05-2.10.2.3

Do not perform logical negation on vectors.

When to Use

Use this Tag to report instances of logical negation of vectors.

Description

The STARC05-2.10.2.3 Tag reports violation when logical negation is performed on vectors.

NOTE: *The Tag considers integers as 32-bit numbers.*

Default Weight

5

Language

Verilog

Parameter(s)

■ [*handle_negation_semantics*](#): By default this parameter is set to no.

If set to "yes", the STARC05-2.10.2.3 Tag does not report violations in following scenarios.

- ? Logical negation of a vector used in if condition - if(!vec)
 - ? Logical negation of a vector assigned to a bit - bit_a = !vec_b
- The STARC05-2.10.2.3 Tag reports additional violations in following scenarios:
- ? Bitwise negation used on a vector in a if condition - if(~vec)
 - ? Bitwise negation used on a vector and assigned to a non vector - bit_a = ~vec_b

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where logical negation is

performed on a vector:

[RECOMMENDED1] Logical negation used on a vector

Potential Issues

Violation may arise when logical negation is performed on a vector.

Consequences of Not Fixing

Logical negation should be used only on single-bit signals. If logical negation is performed to vector signals, the results are confusing.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where logical negation is performed on a vector signal.

To fix the violation, do not perform logical negation on a vector.

Example Code and/or Schematic

Example 1

Consider the following example:

```
module test1(a,z);  
  
    input [3:0] a;  
    output [3:0] z;  
    reg [3:0] z;  
    reg [3:0] a1;  
    always @(a)  
    begin  
        z = !(a1);  
    end  
  
endmodule
```

In the above example, the STARCO5-2.10.2.3 Tag reports a violation as a logical negation is used on a vector.

Example 2

Consider the following example:

```
module test1(a,b,z);
```

STARCO5 Tags

```

input [3:0] a,b;
output [3:0] z;
reg [3:0] z;
  always @(a)
    begin
      z = !(a+=1);
      z = !(a+b);
      z = !(a);
    end
end

```

endmodule

In the above example, the STARCO5-2.10.2.3 Tag reports a violation as a logical negation is used on a vector.

Example 3

Consider the following example:

```

module test1(a,z);
  input [3:0] a;
  output [3:0] z;
  reg [3:0] z;
  integer i = 1;
  always @(a)
    begin
      z = !(i); // Logical negation used on a vector
    end
end
endmodule

```

In the above example, the STARCO5-2.10.2.3 Tag reports a violation as a logical negation is used on a vector.

Default Severity Label

Recommended1

Tag Group

RTL_Description_Technique

STARC05-2.10.3.1v

**Bit-width of the operands of relational operator should match.
(VHDL)**

Description

The STARC05-2.10.3.1v Tag flags relational operations between operands of different bit-widths.

The STARC05-2.10.3.1v Tag checks mismatch in operands for the following relational operators:

>	<	<=	>=	/=	=

Using operands of different bit-widths in a relational operation may generate syntax error in some simulators.

Message Details

The following message appears at the location of a relational operation in hierarchy *<hier-path>* where the bit-width *<bit-widthl>* of the left operand *<opndl>* is different than the bit-width *<bit-widthr>* of the right operand *<opndr>*:

LHS: '*<opndl>*' (width *<bit-widthl>*) does not match RHS:
'*<opndr>*' (width *<bit-widthr>*) [Hierarchy: '*<hier-path>*']

Severity

Mandatory

STARC05-2.10.3.2b

LHS and RHS bit-widths in signal assignments must match. (Verilog)

Description

The STARC05-2.10.3.2b Tag flags bit-width mismatches between LHS and RHS expressions of signal assignments.

Bit-width mismatch between LHS and RHS of a signal assignment is allowed but may lead to inadvertent errors.

By default, integer constants are assumed to have natural width ($\log_2(N)+1$). To consider LRM width (32 bits) for integer constants, set the [use_lrm_width](#) Tag parameter to `yes`.

By default, the STARC05-2.10.3.2b Tag flags cases where LHS width is less than the RHS width. Set the value of the [strict](#) Tag parameter to `yes` or `STARC05-2.10.3.2b` to have the STARC05-2.10.3.2b Tag flag all cases where the LHS width is greater than the RHS width.

By default, the STARC05-2.10.3.2b Tag reports violation for disabled code in loops and conditional (if condition, ternary operator) statements. Set the value of the [disable_rtl_deadcode](#) parameter to `yes` to disable violations for disabled code in loops and conditional (if condition, ternary operator) statements.

By default, the STARC05-2.10.3.2b Tag does not report violation for the static expressions and non-static expressions containing static expressions. Set the value of the [check_static_value](#) parameter to `yes` or `<Tag_list>` to report violation for all cases with width mismatch, involving static expressions and non-static expressions having a static part. Other possible values are `only_const` and `only_expr`.

By default, the *STARC05-2.10.3.2b* Tag suppresses the overflow and does not report a violation when sign extension is used for unsigned signals in addition or subtraction operation.

If you set the [check_unsign_overflow](#) parameter to `yes` or `STARC05-2.10.3.2b`, the Tag does not suppress the overflow and reports a violation when sign extension is used for unsigned signals in addition or subtraction operation.

Consider the following examples:

```
wire [5:0] a,b;
wire [6:0] c;

assign c = {a[5],a} + {b[5],b}; //unsigned expression
```

NOTE: Irrespective of the value of the [check_unsigned_overflow](#) parameter, the Tag suppresses the overflow and does not report a violation when sign extension is used for signed signals in addition or subtraction operation.

By default, the STARCO5-2.10.3.2b Tag does not flag counters. Set the [strict](#) Tag parameter to 1 or STARCO5-2.10.3.2b to report counters. For example, the STARCO5-2.10.3.2b Tag reports violation in the following case if the `strict` Tag parameter is set to 1:

```
assign out1 = in1 + 6'b00_0001;
assign out1 = in1 + 1;
```

By default, the STARCO5-2.10.3.2b Tag checks all signal assignment statements. Set the value of the [assign_check](#) parameter to 1 to have the STARCO5-2.10.3.2b Tag check simple signal assignment statements only, to 2 to have the STARCO5-2.10.3.2b Tag check signal assignment statements using arithmetic operators only, or to 3 to check the RHS of assignments with all the bit-wise operators (OR, AND, Ex-OR, Ex-NOR and NOT) and simple assignments.

By default, the [control_sig_detection_nf](#) parameter is set to `no`. If you set this parameter to `yes`, the Tag treats a signal to be a control signal only if it is used in a conditional operator or control binary operator within the scope of currently processing statement. The Tag does not consider the signal to be a control signal even if it is used as a control signal in other places of the module (that is, the statements other than the current statement).

By default, the STARCO5-2.10.3.2b Tag does not check the bit-width as per LRM. Set the [nocheckoverflow](#) Tag parameter to `yes` to check the bit-width as per LRM. For constants, natural width is considered.

If the [nocheckoverflow](#) Tag parameter is set to `no` then width calculation is performed as per the following specification:

- For constant expressions, width is calculated based on value of expression, as given below:


```

out[2:0] = 5'd3 + 3'd4 ; //RHS Width = 3 (Value = 7)
out[9:0] = 100 << 4 ; //RHS Width = 11
out[2:0] = 100 >> 2 ; //RHS Width = 5

```

If the constant is not part of sub-expression, then the specified width is considered. For example:

```
out[11:0] = 12'b0 ;
```

If width of based number is not specified then natural width is considered. For example:

```
out[4:0] = 'b001; //LHS Width = 5, RHS Width = 1
```

- For subtraction operator, width will be calculated in a similar way as it is done for addition. For example:

```

assign out1[3:0] = a1[3:0] - a1[1:0]; //LHS 4, RHS 5
assign out1[3:0] = a2[3:0] - 4'b1010; //LHS 4, RHS 5

```

- For addition operator, width is calculated as width in which the maximum value of the expression is accommodated:

```

out[2:0] = in1[1:0] + in2[1:0] + in3[1:0]
//Max value RHS 3 + 3 + 3 = 9
//RHS Width = 4

```

```

out[1:0] = in1[1:0] + (3/3) ;
//Max Value RHS = 3 + 1 = 4
//RHS Width = 3

```

- For shift operator, width calculation is performed as follows:

- ? For right shift, if shifted or left operand width is matching LHS width then no violation is reported. For example:

```

out[1:0] = in[3:0] >> in1 ; //RHS Width = 4
out[1:0] = in[4:0] >> 4 //RHS width = 1
out[1:0] = 4 >> in1 ; //RHS width = 3 ( 4 = `100`)

```

- ? For left shift, if shifted or left operand width matches LHS width then no violation is reported. For example:

```

    out[2:0] = 4'b0001 << in1 ; //RHS Width = 4
    out[2:0] = 4 << in1 ; //RHS Width = 3
    out[2:0] = in1[3:0] << in2 ; //RHS Width = 4
    out[3:0] = in1[1:0] << 2 ; //RHS width = 4
    out[6:0] = 8 << in1[1:0]; //shifted width = 7

```

- Width of self-determined expression will be calculated as per LRM.

```

wire a,b,c;
assign a = {b+c} //RHS width = 1
assign b = {1'b1,b+c} //LHS width=1, RHS width=2

```

- For conditional operator, violation will be reported if there is width mismatch either in the left condition or in the right condition:

```

assign a[1] = a1[2] ? a2[0] : a3[2:0] ;
//LHS Width = 1, RHS Width = 3
assign a[1] = a1[2] ? a2[1:0] : a3[2] ;
//LHS Width = 1, RHS Width = 2
assign a[1] = a1[2] ? a2[0] : a3[2] ;

```

- Width of power operator is calculated as follows:

- ? If RHS expression of power operator is static then width of expression is calculated as LHS expression width * RHS expression value:

```

    assign out[4:0] = (in1[1:0] ** 12)
    //LHS Width = 5, RHS Width = 24

```

- ? If RHS expression of power operator is non-static then LHS expression width will be reported:

```

    assign out[4] = in1[1:0] ** 4;

```

- For concatenation operator, width of an expression is calculated in the following ways:

- ? Width calculated after considering the leading zeroes
- ? Width calculated after ignoring the leading zeroes

If any of the two widths calculated in the above manner matches with the LHS width, no violation is reported.

For example, no violation is reported in the following case:

```

A [10:0] = { 1'b0, b[10:0] } ;

```

Here, in the RHS expression, the width calculated after considering leading zeroes is 12 and the width calculated after ignoring leading zeroes is 11. In this case, the LHS width, 11, matches with the RHS width that is calculated after ignoring leading zeroes. Hence, no violation is flagged.

Consider another example, as given below:

```
A [10:0] = { 1'b0, b[9:0] } ;
```

Here, in the RHS expression, the width calculated by considering leading zeroes is 11 and the width calculated after ignoring leading zeroes is 10. In this case, the LHS width, 11, matches with the RHS width that is calculated after considering leading zeroes. Hence, no violation is flagged.

However, a violation is reported in the following case:

```
A [12:0] = { 1'b0, b[10:0] } ;
```

Here, in the RHS expression, the width calculated by considering leading zeroes is 12 and the width calculated after ignoring leading zeroes is 11. None of these values, 11 or 12, matches with the LHS width, 13. Hence, a violation is reported in this case.

The STARCO5-2.10.3.2b Tag preserves all leading zeros, if the padding element does not have all zero ('0') bits. Consider the following example:

```
assign out2 [8:0] = {3'b001, in1, 8'b00001111} ;
```

In the above example, the STARCO5-2.10.3.2b Tag preserves all leading zeros in case of out2 assignment (3 'b001). Therefore, the width of the assignment is calculated as 12 bits.

NOTE: *The Tag does not report violation if the RHS is numeric constant zero or any expression multiplied by a numeric constant zero or a based number zero. For example, violation will not be reported if the RHS is either of 0, 4 * 0, 4 * 1'b0, w1 * 4'b0, or (w1+w2) * 0.*

Message Details

The following message appears at the location where a bit-width mismatch between LHS (<LHS-bit-width> bits) and RHS (<RHS-bit-width> bits) of a signal assignment is encountered:

Bit-width mismatch in signal assignment (LHS: '<LHS-bit-width>' bits and RHS: '<RHS-bit-width>' bits)[Hierarchy: '<hier-path>']

Severity

Recommended1

STARC05-2.10.3.2b_sa

RHS width is greater than LHS width in assignment

When to Use

Use this Tag to check for width mismatch between the LHS and RHS expression of a signal assignment.

Description

The *STARC05-2.10.3.2b_sa* Tag reports violation for bit-width mismatches between LHS and RHS expressions of signal assignments.

The Tag will report violation for cases where width of the LHS expression is less than the width of the RHS expression.

When the value of the *nocheckoverflow* parameter is set to no, the *STARC05-2.10.3.2b_sa* Tag calculates the width as per the following cases:

- For constant expressions, width is calculated based on value of the expression.
- ? If the constant is not part of sub-expression, then the specified width is considered.
- ? If width of based number is not specified, then the natural width is considered.
 - For the addition operator, the width is calculated as the width in which the maximum value of the expression is accommodated.
 - For the subtraction operator, the width is calculated in a similar way as it is done for addition.
 - The width of a self-determined expression is calculated as per LRM.
 - For the conditional operator, a violation is reported, if there is a width mismatch in either the left or the right condition.
 - The width of the power operator is calculated as follows:
- ? If the RHS expression of the power operator is static, then the width of the expression is calculated as per the following formula:

Expression width = LHS expression width * RHS expression value

- ? If the RHS expression of the power operator is non-static, then the width of the expression equals the LHS expression width.

Language

Verilog

Default Weight

10

Parameter(s)

- [assign_check](#): By default, the value of the parameter is set to 0. Therefore, the STARCO5-2.10.3.2b_sa Tag checks all signal assignment statements. Set the value of the parameter to 1 to check simple signal assignment statements only. You can also set the value of the parameter to 2 to check signal assignment statements using arithmetic operators only.
- [nocheckoverflow](#): Default value is no. Set this parameter to yes or Tag name to check the bit-width as per LRM. Other possible value is the Tag name.
- [strict](#): Default value is no. Set the value of the parameter to yes or Tag name, to report the overflow for counter cases, such as, (`in[1:0] + 1'b1`).
- [use_lrm_width](#): Default value is no. Therefore, the default width of integer constants will be natural width, which is $\log_2(N)+1$. Set the value of this parameter to yes to consider the LRM width of integer constants, which is 32 bits.
- [check_pad_concat](#): The default value is no. This Tag provides the width of RHS expression by ignoring leading bits of calculated expression. Set the value of the parameter to yes or <Tag-name> to provide the width of RHS expression by considering leading bits as well.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where the LHS expression

`<lexpr>` of width `<widthl>` of an assignment is less than the RHS expression `<rexpr>` of width `<widthr>`:

[MANDATORY] LHS: '`<lexpr>`' width `<widthl>` is less than RHS: `<rexpr>` width `<widthr>` in assignment [Hierarchy: '`<hier-path>`']

Where, `<hier-path>` is the complete hierarchical path of the signal.

Potential Issues

A violation is reported when the LHS width is less than the RHS width in a signal assignment.

Consequences of Not Fixing

Bit-width mismatch between the LHS and RHS of a signal assignment is allowed but may lead to inadvertent errors.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the line where the width mismatch is detected.

To resolve the violation, explicitly truncate as necessary since it is more readable, instead of relying on the default behavior. Examine the logic to ensure that the truncation does not affect the behavior of the design. In case of counters specified as integers with range, a fix is not required since the user puts the check for avoiding overflow. It is considered a normal practice to use integer ranges for counters.

Examples

Example 1

Consider the following example code in which the Tag reports a violation when the value of the [strict](#) parameter is set:

```
assign out1=in1 + 6'b00_0001;
assign out1=in1 + 1 ;
```

Example 2

Consider the following example:

```
out[2:0] = 5'd3 + 3'd4 ; // RHS Width = 3 (Value = 7)
out[9:0] = 100 << 4 ; // RHS Width = 11
out[2:0] = 100 >> 2 ; // RHS Width = 5
```

In the above example, for constant expressions, the Tag calculates the

width based on the value of the expression.

Example 3

Consider the following example:

```
out[11:0] = 12'b0 ;
```

In the above example, since the constant is not part of the sub-expression, the Tag considers the specified width.

Example 4

Consider the following example:

```
out[1:0] = 'b00101; //LHS Width = 2, RHS Width = 3
```

In the above example, the width of the based number is not specified. Therefore, the Tag considers the natural width. The Tag reports a violation as the LHS width is less than the RHS width.

Example 5

Consider the following example:

```
out[2:0] = in1[1:0] + in2[1:0] + in3[1:0]
// Max value RHS 3 + 3 + 3 = 9
// RHS Width = 4

out[1:0] = in1[1:0] + (3/3) ;
// Max Value RHS = 3 + 1 = 4
// RHS Width = 3
```

In the above example, for the addition operator, the width is calculated as the width in which the maximum value of the expression is accommodated.

Example 6

Consider the following example:

```
assign out1[3:0] = a1[3:0] - a1[1:0]; //LHS 4, RHS 5
assign out1[3:0] = a2[3:0] - 4'b1010; //LHS 4, RHS 5
```

In the above example, for the subtraction operator, the width is calculated in a similar way as it is done for addition.

Example 7

Consider the following example:

```
wire a,b,c;
assign a = {b+c} // RHS width = 1
```



```
assign b = {1'b1,b+c} //LHS width=1, RHS width=2
```

In the above example, the width of a self-determined expression is calculated as per LRM.

Example 8

Consider the following example:

```
assign a[1] = a1[2] ? a2[0] : a3[2:0]; //LHS Width:1, RHS
Width: 3
assign a[1] = a1[2] ? a2[1:0] : a3[2]; //LHS Width:1, RHS
Width: 2
assign a[1] = a1[2] ? a2[0] : a3[2] ;
```

In the above example, for the conditional operator, a violation is reported if there is a width-mismatch in the left or right condition.

Example 9

Consider the following example:

```
assign out[4:0] = (in1[1:0]**12)
//LHS Width:5, RHS Width:24
```

In the above example, since the RHS expression of the power operator is static, the width of the expression is calculated as *LHS expression width * RHS expression value*.

Example 10

Consider the following example:

```
assign out[4] = in1[1:0] ** in2;
```

In the above example, since the RHS expression of the power operator is non-static, the width of the expression equals the LHS expression width.

Example 11

Consider the following example:

```
module top ();
    wire [3:0] a;
    assign a = 5'b10010 & 4'b1111;
endmodule
```

In the above example, the calculated value of RHS (5'b10010 & 4'b1111) is 5'b00010. Width is 5. So the Tag reports the following violation message as the [check_pad_concat](#) parameter is set to yes:

LHS: 'a' width 4 is less than RHS: '(5'b10010 & 4'b1111)' width 5 in assignment [Hierarchy:top]

Default Severity Label

Mandatory

Tag Group

RTL_Description_Technique

Reports and Related Files

None

STARC05-2.10.3.2b_sb

RHS width is lesser than LHS width in assignment

When to Use

Use this Tag to check for width mismatch between the LHS and RHS expression of a signal assignment.

Description

The STARC05-2.10.3.2b_sb Tag reports violation for bit-width mismatches between LHS and RHS expressions of signal assignments.

The Tag will report violation for cases where width of the RHS expression is less than the width of the LHS expression.

When the value of the [nocheckoverflow](#) parameter is set to no, the STARC05-2.10.3.2b_sb Tag calculates the width as per the following cases:

- For constant expressions, width is calculated based on value of the expression.
- ? If the constant is not part of sub-expression, then the specified width is considered.
- ? If width of based number is not specified then natural width is considered.
 - For the addition operator, the width is calculated as the width in which the maximum value of the expression is accommodated.
 - For the subtraction operator, the width is calculated in a similar way as it is done for addition.
 - The width of a self-determined expression is calculated as per LRM.
 - The width of the power operator is calculated as follows:
- ? If the RHS expression of the power operator is static, then the width of the expression is calculated as per the following formula:

$$\text{Expression width} = \text{LHS expression width} * \text{RHS expression value}$$
- ? If the RHS expression of the power operator is non-static, then the width of the expression equals the LHS expression width.

Language

Verilog

Default Weight

10

Parameter(s)

- ***assign_check***: By default, the value of the parameter is set to 0. Therefore, the STARC05-2.10.3.2b_sb Tag checks all signal assignment statements. Set the value of the parameter to 1 to check simple signal assignment statements only. You can also set the value of the parameter to 2 to check signal assignment statements using arithmetic operators only.
- ***nocheckoverflow***: Default value is no. Set this parameter to yes or Tag name to check the bit-width as per LRM. Other possible value is the Tag name.
- ***strict***: Default value is no. Set the value of the parameter to yes or Tag name, to report the overflow for counter cases, such as, (`in[1:0] + 1'b1`).
- ***use_lrm_width***: Default value is no. Therefore, the default width of integer constants will be natural width, which is $\log_2(N)+1$. Set the value of this parameter to yes to consider the LRM width of integer constants, which is 32 bits.
- ***check_pad_concat***: The default value is no. This Tag calculates the width of RHS expression by ignoring leading bits of calculated expression. Set the value of the parameter to yes or <Tag-name> to calculate the width of RHS expression by considering the leading bits as well.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where the width *<widthl>* of LHS expression *<exprl>* of an assignment is greater than the width *<widthr>* of the RHS expression *<exprr>*:

[MANDATORY] LHS: '<expr1>' width <width1> is greater than RHS: '<expr2>' width <width2> in assignment [Hierarchy: '<hier-path>']

Potential Issues

A violation is reported when the LHS width is greater than the RHS width in a signal assignment.

Consequences of Not Fixing

Bit-width mismatch between the LHS and RHS of a signal assignment is allowed but may lead to inadvertent errors.

How to Debug and Fix

Double-click the violation message. The HDL window highlights the line where the width mismatch is detected.

To resolve the violation, explicitly extend as necessary since it is more readable, instead of relying on the default behavior. Examine the logic to ensure that the truncation does not affect the behavior of the design.

In case of counters specified as integers with range, fix may not be required as user may have put the check to avoid overflow this is the normal practice to use integer range for counters.

Examples

Example 1

Consider the following example code in which the Tag does not report a violation when the value of the *strict* parameter is set:

```
assign out1[2:0] = in1[1:0] + 1'b1;
```

Example 2

Consider the following example:

```
out[2:0] = 5'd3 + 3'd4 ; // RHS Width = 3 (Value = 7)
out[12:0] = 100 << 4 ; // RHS Width = 11
out[12:0] = 100 >> 2 ; // RHS Width = 5
```

In the above example, for constant expressions, the Tag calculates the width based on the value of the expression.

Example 3

Consider the following example:

```
out[11:0] = 12'b0 ;
```

In the above example, since the constant is not part of the sub-expression, the Tag considers the specified width.

Example 4

Consider the following example:

```
out[4:0] = 'b00101; // LHS Width = 5, RHS Width = 3
```

In the above example, the width of the based number is not specified. Therefore, the Tag considers the natural width. The Tag reports a violation as the LHS width is less than the RHS width.

Example 5

Consider the following example:

```
out[5:0] = in1[1:0] + in2[1:0] + in3[1:0]
// Max value RHS 3 + 3 + 3 = 9
// RHS Width = 4

out[5:0] = in1[1:0] + (3/3) ;
// Max Value RHS = 3 + 1 = 4
// RHS Width = 3
```

In the above example, for the addition operator, the width is calculated as the width in which the maximum value of the expression is accommodated.

Example 6

Consider the following example:

```
assign out1[5:0] = a1[3:0] - a1[1:0]; //LHS 6, RHS 5
assign out1[5:0] = a2[3:0] - 4'b1010; //LHS 6, RHS 5
```

In the above example, for the subtraction operator, the width is calculated in a similar way as it is done for addition.

Example 7

Consider the following example:

```
wire a,b,c;
assign a = {b+c} // RHS width = 1
assign {a,b,b} = {1'b1,b+c} //LHS width=3, RHS width=2
```

In the above example, the width of a self-determined expression is calculated as per LRM.

Example 8

Consider the following example:

```
assign out[4:0] = (in1[1:0]**2)
//LHS Width:5, RHS Width:4
```

In the above example, since the RHS expression of the power operator is static, the width of the expression is calculated as *LHS expression width * RHS expression value*.

Example 9

Consider the following example:

```
assign out[4] = in1[1:0] ** in2;
```

In the above example, since the RHS expression of the power operator is non-static, the width of the expression equals the LHS expression width.

Example 10

Consider the following example:

```
module top ();
    wire [3:0] a;
    assign a = 5'b10010 & 4'b1111;
endmodule
```

In the above example, the calculated value of RHS (5'b10010 & 4'b1111) is 5'b00010. Ignoring the leading bits, it will be 10. The width is 2. So the Tag reports the following violation message as the [check_pad_concat](#) parameter is not set:

```
LHS: 'a' width 4 is greater than RHS: '(5'b10010 & 4'b1111)'
width 2 in assignment [Hierarchy:top]
```

Default Severity Label

Mandatory

Tag Group

RTL_Description_Technique

Reports and Related Files

None

STARCO5-2.10.3.7

Match the bit-width with the base number part. (Verilog)

Description

The *STARCO5-2.10.3.7* Tag reports based numbers where the bit-width definition and specified value do not match.

If the bit-width definition-part and the value part does not match like the examples below, it can easily lead to mistakes:

```
a = 3'b11111111;  
a = 3'b11;
```

When using a parameter, it is sometimes advantageous not to specify bit-widths so that any bit-width is allowed. Yet, bit-width should be specified for all constants and parameters unless there is a special reason. Then, the bit-width of the value should be matched with the based number part because it will not be an error if they do not match and that will lead to confusion.

For hexadecimal and octal based numbers, the Tag takes effective bit-width. For example, the Tag does not report a violation for `7'o123`, because effective bit-width for 123 is 7 (1+3+3), which matches with the based number part.

For binary based numbers, the Tag reports a violation if bit-width does not exactly match with the based number part. For example, a violation is reported for `1'b01`, although effective bit-width for '01' is 1.

When the *ignore_based_width* parameter is set to *yes*, the Tag does not report violations for binary, hexal, octal, and decimal based numbers, if the width specified is greater than the actual bit-width of the based number. By default, violations are reported for such cases.

For example, in the following cases, no violations are reported when the parameter is set to *yes* whereas violations are reported by default.

```
a = 11'd25;  
a = 13'h3ff;
```

Set the *strict* parameter to *yes* to report a violation for a based number that has all zeros. For example, the Tag reports a violation for `1'b00` if the *strict* parameter is set to *yes*.

Message Details

The following message appears at the location where a based number constant `<const-name>` is declared with mismatching the bit-width specification (`<num1>`) and value specification (`<num2>`):

Bit-width '`<num1>`' should match the based number width '`<num2>`' in '`<const-number>`'

Severity

Recommended2

Examples

Consider the following example:

```
module bit_width;
wire a;
assign a=6'o0; assign a=6'o2;//STARCO5-2.10.3.7 generates a
warning as bit width '3' mismatch the based number width '6'
in '6'o2'
```

```
assign a=6'o11;
assign a=6'o02;
```

```
assign a=10'o02;//STARCO5-2.10.3.7 generates a warning as
bit-width '6' mismatch the based number width '10' in
'10'o02'
```

```
assign a=6'o0011;
assign a=6'h11;
assign a=6'h02;
assign a=6'h0011;
```

```
assign a=3'h0011;//STARCO5-2.10.3.7 generates a warning as
bit-width '16' mismatch the based number width '3' in
'3'h0011'
```

```
assign a=6'd1;//STARCO5-2.10.3.7 generates a warning as bit
width '1' mismatch the based number width '6' in '6'd1'
```

STARC05 Tags

```
assign a=2'd0011;//STARC05-2.10.3.7 generates a warning as  
bit-width '4' mismatch the based number width '2' in  
'2'd0011'
```

```
assign a=6'b1;//STARC05-2.10.3.7 generates a warning as bit  
width '1' mismatch the based number width '6' in '6'b1'
```

```
assign a=6'b000;//STARC05-2.10.3.7 generates a warning when  
strict parameter is set as bit-width '3' mismatch the based  
number width '6' in '6'b000'
```

```
assign a=1'b01;//STARC05-2.10.3.7 generates a warning as bit  
width '2' mismatch the based number width '1' in  
'1'b01'
```

```
assign a=4'b00011;//STARC05-2.10.3.7 generates a warning as  
bit-width '5' mismatch the based number width '4' in  
'4'b00011'
```

```
endmodule
```

In the above example, the *STARC05-2.10.3.7* Tag reports violations because the bit-width definitions and specified values do not match.

STARC05-2.10.4.1

Do not use data types other than reg, wire, and integer. (Verilog)

Description

The STARC05-2.10.4.1 Tag flags data types other than reg, wire, and integer. The STARC05-2.10.4.1 Tag flags use of real, time, realtime, and array of time data types.

Verilog can assign among different data types. This is impossible among nearly all data types in the case of VHDL. The big difference is that VHDL must use type conversion function when assigning among different data types while Verilog can assign unlimitedly. Use only reg, wire, or integer data types.

Message Details

The following message appears at the location where object of a data type other than reg, wire, and integer is defined:

Data types other than reg, wire, or integer should not be used

Severity

Mandatory

STARC05-2.10.4.4

Negative value should not be assigned to integer. (VHDL)

Description

The STARC05-2.10.4.4 Tag flags integer objects that are assigned with a negative value.

While assigning negative value to a register, the sign bit is to be kept in mind as an integer is 32-bit long but the assigned register may be more than 32 bits. This situation can lead to erroneous value, that is, a positive value.

Message Details

The following message appears at the location where an integer *<name>* is assigned a negative value:

Negative value assigned to integer '*<name>*'

Severity

Recommended3

STARC05-2.10.4.5

Integer objects must not be assigned negative values. (Verilog)

Description

The STARC05-2.10.4.5 Tag flags negative value assignments to objects of type integer.

While it is possible to assign negative values to integer objects, you should be careful in using such objects.

For example, if you assign a negative integer object (32-bit wide) to a register that has a longer bit-width (more than 32-bits), the assigned value is positive (incorrect) instead of negative (correct).

NOTE: *The STARC05-2.10.4.5 Tag supports generate-for block.*

Message Details

The following message appears at the location where a negative value is assigned to an integer object `<name>`:

Negative value assigned to integer object '`<name>`'

Severity

Mandatory

STARC05-2.10.4.6

Signals should not be assigned negative values. (Verilog)

Description

The STARC05-2.10.4.6 Tag flags negative value assignments to signals.

While it is syntactically legal to assign negative values to signals, it is recommended to avoid such constructs as some EDA tools may not be able to interpret such designs correctly.

The Tag does not report violation for RHS expressions exceeding 32-bit capacity.

By default, the STARC05-2.10.4.6 Tag evaluates based number expressions as unsigned and violations are not reported for these expressions. When the value of the *strict* parameter is set to yes, the Tag evaluates these expressions as signed and violations are reported, if their value is negative.

NOTE: *The Tag reports a violation, if base is signed declared and, therefore, 1 is treated as negative.*

Message Details

The following message appears at the location where a signal *<sig-name>* is assigned a negative value:

Negative value assigned to signal '*<sig-name>*'

Severity

Recommended2

Example

The following example explains the STARC05-2.10.4.6 Tag behavior in default and when the *strict* parameter is set:

```
module sample(TOPOUT3);

wire [31:0] d32bit;
    assign d32bit[31:0] = -3534; //violation by default
    assign d32bit[31:0] = 3534-542212;//violation by
                                default
```

```
input [4:0]   TOPOUT3;
assign TOPOUT3 = 1'b1;
    assign TOPOUT3 = -1'b1; //violation when strict is set
    assign TOPOUT3 = -3'd3;  //violation when strict is set
    assign TOPOUT3 = 4'd4-4'd8; //violation when strict is
                                set
    assign TOPOUT3 = -3'd3-4'd4; //violation when strict is
                                set
    assign TOPOUT3 = 4'd8-4'd4;
    assign TOPOUT3 = -(4'd4+4'd8); //violation when strict
                                    is set

endmodule
```


STARC05-2.10.4.6v

Signed data type must be used in signed operation. (VHDL)

Description

The STARC05-2.10.4.6v Tag flags operations including objects of type integer that have range specification with negative numbers.

It is recommended to use only data type signed for signed operations since using negative number in range specification tends to lead to description mistakes.

Message Details

The following message appears at the location where a data object of type other than signed is used in a signed operation:

Signed data type not used in signed operation

Severity

Recommended3

STARC05-2.10.8.1

Do not use arithmetic and logical expressions at the right and left sides of the division or modulus operator.

Description

The STARC05-2.10.8.1 Tag flags when the LHS or RHS of the division or modulus operator contains arithmetic/logical expression.

To ensure a deterministic divider width, arithmetic and relational operators should not be used at the right and left sides of the division and modulus operators.

If both LHS and RHS contain arithmetic/logical expression, the STARC05-2.10.8.1 Tag reports violation for the LHS part only.

In case of multiple arithmetic/logical expressions present at the left or right side of the division/modulus operator, the STARC05-2.10.8.1 Tag reports only one violation.

In case of nested use of division/modulus operator, this Tag will report violation only for the outermost operator.

Arithmetic operators include unary plus, unary minus, addition, subtraction, multiplication, division, modulus, exponential, shift left arithmetic, shift right arithmetic.

Logical operators include and, or, not, shift left logical and shift right logical.

Consider the following scenarios where the STARC05-2.10.8.1 Tag reports violation:

- `a / (b + c);` // violation
- `(a + b) % (c + d)` // only one violation; for `(a + b)`
- `(a + (b * c)) / (a - d)` // only one violation for `/` operator
- `a % (b / (c + d))` // violation only for `%` operator

The STARC05-2.10.8.1 Tag ignores static expression

Message Details

The following message appears when an arithmetic/logical expression `<expr>` present at the LHS/RHS of the `<division/modulus>` operator:

STARCO5 Tags

<Arithmetic/Logical> expression '<expr>' present at the <left/right> side of the <division/modulus> operator

Severity

Mandatory

STARC05-2.11.3.1

Ensure that the sequential and combinational parts of an FSM description should be in separate always blocks. (Verilog)
Ensure that the sequential and combinational parts of an FSM description should be separated (VHDL)

When to Use

Use this Tag to report detect the sequential and combinational parts of an FSM that are in the same `always` or `sequential` blocks.

Description

Verilog

The STARC05-2.11.3.1 Tag flags combinational part of an FSM description that is in the same `always` construct as the corresponding sequential part of the FSM.

VHDL

The STARC05-2.11.3.1 Tag flags combinational part of an FSM description that is in the same `process` construct as the corresponding sequential part of the FSM.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

- **`ignore_fsm_counter`**: Default value is `no`. Set this parameter to `yes` to ignore violations for sequential block infer counter logic. This parameter is only applicable for Verilog.
- **`strict`**: Default value is `no` and the Tag does not report violation if combinational and sequential part of an FSM are present in the same `if-else` chain and comparison in combinational `if` condition is with constant or signal propagating constant value (for example, assigned to constant, based number, parameter or localparam before comparison). Set this parameter to `yes` to report violation for such cases. This parameter is applicable for **Verilog** only.

Constraint(s)

None

Messages and Suggested Fix

Verilog

The following message appears at the location of the combination part of an FSM that is in the same always construct as the sequential part of the FSM:

[RECOMMENDED3] Combinational and sequential parts of an FSM described in same always block

Potential Issues

Violation may arise when sequential and combinational parts of an FSM are described in the same always block.

Consequences of Not Fixing

Using the sequential part (flip-flop description) and combinational part (case construct) of an FSM in the same always block increases complexity of the design thereby reducing the readability of the code.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line of the FSM description whose sequential and combinational parts are described in the same always block.

To fix the violation, ensure that the sequential and combinational parts of an FSM are described in separate always blocks.

VHDL

The following message appears at the location of the combination part of an FSM that is in the same process as the sequential part of the FSM:

[RECOMMENDED3] Combinational and sequential parts of a FSM described in the same process block

Potential Issues

Violation may arise when sequential and combinational parts of an FSM are described in the same process block.

Consequences of Not Fixing

Using the sequential part (flip-flop description) and combinational part (case construct) of an FSM in the same process block increases complexity of the design thereby reducing the readability of the code.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line of the FSM description whose sequential and combinational parts are described in the same process block.

To fix the violation, ensure that the sequential and combinational parts of an FSM are described in separate process blocks.

Example Code and/or Schematic

Verilog

Example 1

Consider the following example:

```
always @(posedge clk or posedge reset)
begin
    if (reset)
        C_STATE <= ST0;
    else
        C_STATE <= N_STATE;
    case(C_STATE)
        ST0 : begin
            FSMOUT = 2'b00;
            N_STATE = ST1;
        end
        ST1 : begin
            FSMOUT = 2'b11;
            N_STATE = ST2;
        end
        ST2 : begin
            FSMOUT = FSMIN1;
            N_STATE = ST3;
        end
        ST3 : begin
            FSMOUT = FSMIN2;
```

STARC05 Tags

```

        N_STATE = ST0;
    end
endcase
end

```

In the above example, the STARC05-2.11.3.1 Tag reports a violation as the combinational and sequential parts of an FSM are described in same `always` block.

Example 2

Consider the following example:

```

localparam N = 0;
reg a;
always @(posedge clk or posedge ares) begin
    if (ares)
        a <= 1'b0;
    else if (a == N)//violation by default
        a <= b;
    else
        a <= 1'b1;
end

```

In the above example, the STARC05-2.11.3.1 Tag reports a violation by default.

Example 3

Consider the following example:

```

reg a;
always @(posedge clk or posedge ares) begin
    if (ares)
        a <= 1'b0;
    else if (a == 3'b0)//violation by default
        a <= b;
    else
        a <= 1'b1;
end

```

In the above example, the STARC05-2.11.3.1 Tag reports a violation by default.

Example 4

Consider the following example:

```
reg a;
always @(posedge clk or posedge ares) begin

    if (ares)
        a <= 1'b0;
    else if (a == 5)
        a <= b;
    else
        a <= 1'b1;
end
```

In the above example, the STARC05-2.11.3.1 Tag reports a violation when the *strict* parameter is set to yes.

Example 5

Consider the following example:

```
wire N;
    reg a;
assign N = 0;
always @(posedge clk or posedge ares) begin
    if (ares)
        a <= 1'b0;
    else if (a == N)
        a <= b;
else
    a <= 1'b1;
end
```

In the above example, the STARC05-2.11.3.1 Tag reports a violation when the *strict* parameter is set to yes.

VHDL

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all
```



```

process(clk)
    variable state : std_logic_vector(2 downto 0);
begin
    if(clk'event and clk='1') then
        case state is
            when state_a =>
                inst_out <= "001";
                if(inst_in="000") then
                    state := state_b;
                elsif(inst_in="001" ) then
                    state := state_e;
                else
                    state := state_a;
                end if;

            when state_b =>
                inst_out <= "011";
                state := state_c;

            when state_c =>
                inst_out <= "111";
                state := state_d;

            when state_e =>
                inst_out <= "001";
                if( inst_in= "111")then
                    state := state_a;
                else
                    state := state_e;
                end if;
            when others =>
                inst_out <= "111";
                state := state_a;
            end case;
        end if;
    end process;

```

In the above example, the STARC05-2.11.3.1 Tag reports a violation as the combinational and sequential parts of an FSM are described in the same `always` block.

Default Severity Label

Recommended3

Tag Group

Creating Functional Libraries

Reports and Related Files

No related report or files

STARC05-2.2.2.3a

Multiple event Expressions should not be described with always constructs. (Verilog)

Description

The STARC05-2.2.2.3a Tag flags always constructs with multiple event expressions.

Do not describe multiple event expressions "@" in always constructs for both sequential and combinational circuits. Logic synthesis with this kind of description might be possible but not in many cases.

By default, the STARC05-2.2.2.3a Tag flags the first occurrence of multiple event expressions inside always block. Set the value of the [strict](#) Tag parameter to `yes` or `STARC05-2.2.2.3a` to report all occurrences of multiple event expressions in always block.

Message Details

The following message appears at the location of the sensitivity list of an always construct that has multiple event constructs:

Multiple event Expressions described with always block

Severity

Mandatory

STARC05-2.2.3.3

Do not assign over the same signal in an always construct for sequential circuits

When to Use

Use this Tag to detect initial value assignments to flip-flops in `always` constructs.

Description

The *STARC05-2.2.3.3* Tag reports initial value assignments to flip-flops in `always` constructs.

The Tag considers that a flip-flop output is assigned an initial value in an `always` construct, if any of the following conditions hold true:

- If there exists a direct assignment to the flip-flop output followed by a conditional assignment (inside `if-else` branch) to the signal in the same `always` construct.
- If there exists multiple assignments to the flip-flop output in the same block in an `always` construct.
- If a signal is assigned multiple times in different `if` blocks within an `always` block.

Default Weight

5

Language

Verilog

Parameter(s)

- *strict*: Default value is `no`. Set this parameter to `yes` to enable the *STARC05-2.2.3.3* Tag to:
 - ? report a violation for a variable initialization within an `if` statement, which does not have a corresponding `else` statement.
 - ? report a violation for an initialization variable present in any of the `if-else-if` branch or inside any of the case labels.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where the output of the flip-flop `<flop-name>` is assigned an initial value in an `always` construct at line number `<num>`:

[MANDATORY] Flipflop '`<flop-name>`' is assigned over the same signal in an `always` construct for sequential circuits (at line '`<num>`')

Potential Issues

The *STARC05-2.2.3.3* Tag considers that a flip-flop output is assigned an initial value in an `always` construct, if any of the following conditions hold true:

- If there exists a direct assignment to the flip-flop output followed by a conditional assignment (inside `if-else` branch) to the signal in the same `always` construct.
- If there exists multiple assignments to the flip-flop output in the same block in an `always` construct.
- If a signal is assigned multiple times in different `if` blocks within an `always` block.

Consequences of Not Fixing

Multiple assignments to the same signal overwrite the initial value and may result in unexpected functionality.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the a signal is re-assigned inside the same `always` construct.

Scroll through the HDL Viewer window for the first declaration of the signal in the `always` construct.

To fix the violation, avoid assigning the same variable multiple times. If possible, use `if-else-if` coding style instead of parallel `if` statements.

Example Code and/or Schematic

Example 1

Consider the following example:

```
module test (out, in1, in2, clk, reset);
    output out;
    input in1, in2, clk, reset;

    reg out, out1;

    always @ (posedge clk)
        begin
            out <= 1'b0;
            out <= in1;
        end

    always @ (posedge clk or negedge reset)
        begin
            out1 <= 1'b1;
            if (!reset)
                out1 <= 1'b0;
            else
                out1 <= in2;
        end
    end
endmodule
```

The flip-flop output out is being directly assigned two values (1'b0 and in1). However, the second assignment overwrites the first assignment and the *STARCO5-2.2.3.3* Tag is violated.

The flip-flop output out1 is being directly assigned a value (1'b1) and then being conditionally assigned a value (1'b0 and in2). Thus, the direct assignment is assumed to be an initial value assignment and the *STARCO5-2.2.3.3* Tag is violated. For this example, VC SpyGlass Lint generates the following message at the location of the conditional assignment (else statement):

Flipflop 'out1' is assigned initial value in always block (at line '15')

Example 2

Consider the following example:

```
always @ (posedge clk or negedge reset)
begin
    out1 <= 1'b1; // Direct assignment
    if (!reset)
        out1 <= in1; //Violation only when the strict parameter is set to yes
```

In the above example, the initialization to a variable is not treated as the initial value, if there is no corresponding else condition to an if condition. Therefore, the Tag reports a violation for the conditional assignment when the value of the *strict* parameter is set to yes.

Example 3

Consider the following example:

```
always @ (posedge clk or negedge reset)
begin
    if (!reset)
    begin
        out1 <= 1'b0; // Initialization inside if block
        if (en)
            out1 <= in; //Violation only when the strict parameter is set to yes
    end
end
```

In the above example, the variable initialization is inside the nested if block. In addition, the Tag does not report a violation for this case.

Example 4

Consider the following example:

```
always @ (posedge clk)
begin
    out <= 1'b0; // First assignment
    out <= in1; // Second assignment in same
                // block
end

always @(posedge clk) begin
```

```

if (rst) begin
    q<=1'b0; // First assignment
    q<=1'b1; // Second assignment inside same
              // block (if)
end
else begin
    q<=data; //First assignment in the else block
end
end

```

In the above example, the *STARCO5-2.2.3.3* Tag reports two violations as there exists multiple assignments to the flip-flop output in the same block in an `always` construct.

Example 5

Consider the following example:

```

always @ (posedge clk or negedge reset)
begin
    out <= 1'b1; // Direct assignment
if (! reset)
    out1 <= in1; //Conditional assignment
if (! set)
    out1 <= in2; //Conditional assignment
end

```

In the above example, the *STARCO5-2.2.3.3* Tag reports a violation as the signal `out1` is assigned multiple times in different `if` blocks within an `always` block.

Example 6

Consider the following example:

```

always @ (posedge clk or negedge reset)
begin
    out1 <= 1'b0; //Initialization inside if block
    if (!reset)
        out1 <= 1'b1; //Violation only when strict is set to yes
    else
        out1 <= in1; // default violation
end

```


In the above example, the *STARC05-2.2.3.3* Tag reports a violation for the conditional assignment inside the `else` condition. In addition, the Tag reports a violation for the conditional assignment inside `if` condition, if the *strict* parameter is set to `yes`.

Example 7

Consider the following example:

```
always @ (posedge clk or negedge reset)
begin
    out1 <= 1'b1; //Direct assignment
    case (!reset)
    0: begin
        out1 <= in1; //Conditional assignment
    end
    1: begin
        out1 <= in2; //Conditional assignment
    end
    endcase
end
```

In the above example, the signal `out1` is assigned under different conditions of the case label. The *STARC05-2.2.3.3* Tag does not report violations for such signals between the direct and conditional assignments. However, the Tag reports violations in the above case if the *strict* parameter is set to `yes`.

Default Severity Label

Mandatory

Tag Group

RTL Description Technique

Reports and Related Files

No related reports or files.

STARC05-2.3.1.2a

Do not use quasi-continuous assignments. (Verilog)

Description

The STARC05-2.3.1.2a Tag flags flip-flop descriptions that use assign statements in always blocks (quasi-continuous assignments).

Also, this tag reports violations for quasi-continuous assignments in initial blocks by default.

Such descriptions are not synthesizable.

By default, the STARC05-2.3.1.2a tag reports violation for initial blocks. Set the value of the [ignore_initial_blocks](#) parameter to yes to ignore STARC05-2.3.1.2a violations in initial blocks.

Message Details

The following message appears at the location where an assign statement is encountered in an always construct:

Quasi-continuous assignment used

Severity

Mandatory

STARCO5-2.3.1.2b

Do not use deassign statements. (Verilog)

Description

The STARCO5-2.3.1.2b Tag flags flip-flop descriptions that use deassign statements.

Such descriptions are not synthesizable.

Message Details

The following message appears at the location where a deassign statement is encountered:

```
deassign statement used
```

Severity

Mandatory

STARC05-2.3.1.2c

Do not use unsynthesizable User-Defined Primitives (UDPs).

When to Use

Use this Tag to identify unsynthesizable UDP descriptions.

Description

The STARC05-2.3.1.2c reports violation for only unsynthesizable UDP descriptions.

NOTE: *This Tag flags at the last line of the UDP definition.*

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where an unsynthesizable UDP description is encountered:

[MANDATORY] User-Defined Primitive declared

Potential Issues

A violation is reported when an unsynthesizable UDP is used in the design.

Consequences of Not Fixing

This type of coding style is not supported by most synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the last line of the User-defined primitive declaration.

Scroll through the HDL Viewer window to search for the declaration of the UDP.

To fix the violation, avoid using unsynthesizable UDPs in the design.

Example Code and/or Schematic

Consider the following example:

```
primitive dff (QN, D, CP, R, S);
output QN;
input D, CP, R, S;
reg QN;
table
  // D CP R S : Qtn : Qtn+1
  1 (01) 0 0 : ? : 1;
  1 (01) 0 x : ? : 0;
  //? ? 0 x : 0 : 0;
  0 1 0 0 : ? : 1; // clocked data
  //0 (01) x 0 : ? : 1; // pessimism
  ? ? x 0 : 1 : 1; // pessimism
  //1 (x1) 0 0 : 0 : 0;
  //0 (x1) 0 0 : 1 : 1;
  //1 (0x) 0 0 : 0 : QN;
  0 (0x) 0 0 : 1 : 1;
  //? ? 1 ? : ? : 1; // asynch clear
  //? ? 0 1 : ? : 0; // asynchronous set
  ? n 0 0 : ? : -;
  * ? ? ? : ? : -;
  ? ? (?0) ? : ? : -;
  ? ? ? (?0): ? : -;
  ? ? ? ? : ? : x;
endtable
endprimitive
module top();
wire QN,D,CP,R,S;
dff inst1(QN, D, CP, R, S);
endmodule
```

In the above example, STARCO5-2.3.1.2c Tag reports a violation as an

unsynthesizable user-defined primitive, `dff`, is defined in the design.

Default Severity Label

Mandatory

Tag Group

Flip-Flop Inference, quality1

Reports and Related Files

No related reports or files.

STARC05-2.3.1.2va

(Do not use `rising_edge` or `falling_edge` of clock for flip-flop description. (VHDL))

Description

The STARC05-2.3.1.2va Tag flags `rising_edge` and `falling_edge` package functions used in flip-flop descriptions.

Flip-flop description using `rising_edge` and `falling_edge` package functions should be avoided as ATPG tools do not support such descriptions.

It is recommended to use 'event attribute instead.

Example:

```
if (clk'event and clk = '1') --{Good}
if (rising_edge(clk)) then    --{Bad}
```

Message Details

The following message appears at the location where a `rising_edge` or `falling_edge` package function is used in flip-flop description:

`rising_edge` used to describe flip-flop

`falling_edge` used to describe flip-flop

Severity

Mandatory

STARC05-2.3.1.2vb

Do not use stable attribute for flip-flop description. (VHDL)

Description

The STARC05-2.3.1.2vb Tag flags 'stable attribute used in flip-flop descriptions.

The 'stable attribute in flip-flop description is not supported by all synthesis tools.

The description, where 'event and 'stable are mixed, is difficult to read and logic negation is required for clock description using 'stable attribute. Therefore, clock description must be made using 'event attribute only.

For example:

```
if (clk'event and clk = '0') then ... {Good}
if (not clk'stable and clk = '1') then ...{Bad}
```

Message Details

The following message appears at the location where a 'stable attribute is used in a flip-flop description:

stable attribute used to describe flip-flop

Severity

Mandatory

STARC05-2.3.1.2vc

Do not use non-synthesizable clocking styles for flip-flop description. (VHDL)

Description

The STARC05-2.3.1.2vc Tag flags non-synthesizable clocking styles used to describe flip-flops.

The STARC05-2.3.1.2vc Tag considers the following clocking styles as non-synthesizable clocking styles:

```
library ieee;
use ieee.std_logic_1164.all
if (clk'event) then
if (not clk'stable) then
if (clk'stable) then
if (clk'stable and clk = '1') then
if (clk'event and clk'stable) then
    -- If edge is not specified
if (not clk'event and clk = '0') then
if (clk'event and clk = "00") then
    -- Multi-bit clock-signal
elsif (clk'event) then
elsif (not clk'stable) then
elsif (clk'stable) then
elsif (clk'stable and clk = '1') then
elsif (clk'event and clk'stable) then
    -- If edge is not specified
elsif (not clk'event and clk = '0') then
elsif (clk'event and clk = "00") then
    -- Multi-bit clock-signal
wait until clk'event;
wait until not clk'stable;
wait until clk'stable;
wait until clk'stable and clk = '1';
```

Multi-bit clock signals are considered non-synthesizable.

A clocking style is considered to be synthesizable if the clock signal is of any of the following types:

- Simple Name
- Selected Name
- Qualified Expression
- Type Conversion
- Index Name (Single dimension)
- Attribute Name

The STARCO5-2.3.1.2vc Tag considers the following clocking styles as synthesizable clocking styles:

```
wait until clk = '1';
if (clk'event and clk = '1') then
if (not clk'stable and clk = '1') then
wait until clk'event and clk = '1';
if (falling_edge(clk)) then
if (rising_edge(clk)) then
if ((clk'event and not clk'stable) and clk = '0') then
wait until (clk'event and not clk'stable) and clk = '0';
```

The STARCO5-2.3.1.2vc Tag does not check conditional signal and select signal assignments.

The STARCO5-2.3.1.2vc Tag checks only the if-elsif and the wait constructs and does not check the while-loop constructs.

Message Details

The following message appears at the location where a non-synthesizable clocking style is encountered:

Non-synthesizable clocking style used to describe flip-flop

Severity

Mandatory

STARC05-2.3.1.5b

Ensure that the delay values are non-negative.

When to Use

Use this Tag to detect negative delay values in the design.

Description

The STARC05-2.3.1.5b Tag reports negative delay values in the design.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a negative delay value is encountered:

[MANDATORY] Negative delay value used

Potential Issues

Violation may arise when a negative delay value is used in the design.

Consequences of Not Fixing

A negative delay value indicates a step back in time which is confusing and therefore error-prone. Also, negative delay values may cause efficiency problems in some simulators.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where negative delay is used.

Scroll through the HDL Viewer window for the parameter/local parameter/enum declarations, if used in delay.

To fix the violation, use negative delay values only to model special behavior in cell libraries.

Example Code and/or Schematic

Consider the following example:

```
module mod_2_3_1_5test2(in1, clk, out1);  
    input [3:0] in1;  
    input clk;  
    output [3:0] out1;  
    reg [3:0] out1;  
    always @(posedge clk)  
        begin  
            out1 = #(-10) in1  
        end  
  
endmodule
```

In the above example, the STARCO5-2.3.1.5b Tag reports a violation as a negative delay value is used.

Default Severity Label

Mandatory

Tag Group

RTL Description Technique

Reports and Related File

No related reports or files

STARC05-2.3.1.6

Check the logic level of the reset signal as specified in the sensitivity list of the always block.

When to Use

Use this Tag to check for logic level of the reset signal as specified in the always construct's sensitivity list.

Description

The STARC05-2.3.1.6 Tag flags those uses of reset signals that are different from the reset line in the always construct's sensitivity list.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a reset signal *<rst-name>* is being checked for a logic level that is different from that described in the sensitivity list:

[MANDATORY] Checking for logic level of reset '*<rst-name>*' different from the edge specified in sensitivity list

Potential Issues

Violation may arise when logic level or edge of reset signal used is different from the logic level or edge specified in the sensitivity list.

Consequences of Not Fixing

Using reset signals in variance of the reset line in the always construct's

sensitivity list may lead to generation of unintentional synchronous flip-flop during synthesis.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where a reset signal is used with logic level different from that in the sensitivity list.

To fix the violation, use the same logic level in both the usage and the sensitivity list.

Example Code and/or Schematic

Consider the following example where the reset line in the always construct has the negedge of the reset signal RESET and the logic that follows checks for the positive value of the same signal:

```
module test(RESET,Q);  
output Q;  
input RESET;  
reg Q,CLK,D;  
always @(posedge CLK or negedge RESET)  
    if(RESET==1)  
        Q<= 1'b0;  
    else  
        Q<=D;  
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Checking for logic level of reset 'RESET' different from the edge specified in sensitivity list

Default Severity Label

Mandatory

Tag Group

RTL Description Technique

Reports and Related File

No related reports or files

STARC05-2.3.3.1

Do not use edges of multiple clocks in same always block. (Verilog)
Do not use multiple clock signals in a single process block (VHDL)

Description

Verilog

The STARC05-2.3.3.1 Tag flags always constructs with edges of multiple clocks.

An always construct with edges of multiple clocks may not be synthesizable. Some synthesis tools can handle limited forms of implicit Finite-state machine. However, using such constructs is not considered good design practice.

VHDL

The STARC05-2.3.3.1 Tag flags process constructs using edges of multiple clocks.

A process construct with edges of multiple clocks may not be synthesizable. Some synthesis tools can handle limited forms of implicit Finite-state machine. However, using such constructs is not considered good design practice.

Message Details

Verilog

The following message appears at the location of the always construct in hierarchy *<hier-path>* that uses edges of multiple clocks:

Edges of multiple clocks used in the same always block.[Hierarchy: '*<hier-path>*']

VHDL

The following message appears at the location of the process construct that uses edges of multiple clocks *<sig-list>*:

Multiple clocks *<sig-list>* used in the same process block

Severity

Mandatory

Verilog Examples

Consider the following examples where the always construct has edges of two clocks CLK1 and CLK2 to describe sequential elements:

```
...
always
begin
    @(posedge CLK1)
        OUT1 = IN1;
    @(posedge CLK2)
        OUT3 = IN3;
end
always @(posedge CLK1 or posedge CLK2)
begin
    OUT1 = IN1;
end
```

VC SpyGlass Lint generates the STARC05-2.3.3.1 message for these examples.

To fix this message, separate the description of each clock in to a separate always constructs. Example :

```
always
begin
    @(posedge CLK1)
        OUT1 = IN1;
end
always
begin
    @(posedge CLK2)
        OUT3 = IN3;
end
```

VHDL Examples

Consider the following example where the always construct has edges of two clocks CLK1 and CLK2 to describe sequential elements:

```
library ieee;
use ieee.std_logic_1164.all
```



```
process (CLK1,CLK2)
begin
  if(CLK1' event and CLK1='1') then
    q1<='0'
  end if;
  if(CLK2'event and CLK2='1') then
    q1<=d
  end if;
```

STARC05-2.3.3.2b

Do not describe multiple clock edges in a single process/always block for the same edge of a single clock

Description

The STARC05-2.3.3.2b Tag flags cases where a clock is described with the same edge multiple times in the same always construct or process construct.

In such a case, the logic synthesis tool generates implicit Finite-state machine circuits that control the edge change order. However, this Finite-state machine circuit becomes an implicit Finite-state machine circuit that is not defined in the HDL description, so it can no longer be claimed to be a concise RTL description.

Logic synthesis of such designs would be possible but such usage should be avoided.

Message Details

Verilog

The following message appears at the location where clock *<clk-name>* is described with an edge when it has already been described with the same edge earlier in the same always construct:

Same edge of a single clock '*<clk-name>*' used multiple times in same always block.

VHDL

The following message appears at the location where clock *<clk-name>* is described with an edge when it has already been described with the same edge earlier in the same process construct:

Same edge of a clock '*<clk-name>*' described more than once in same process block

Severity

Reference

STARC05-2.3.4.2

Reports initial construct used.

When to Use

Use this Tag to report initial constructs used in the design.

Description

The STARC05-2.3.4.2 Tag flags initial constructs in the design.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where an initial construct is encountered:

[CAUTION] initial construct used

Potential Issues

Violation may arise when an initial construct is used in the design.

Consequences of Not Fixing

Initial constructs are unsynthesizable and can cause simulation synthesis mismatch.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the initial construct is used.

The initial constructs have no physical counterpart and should only be used

for simulation setup. Use reset signals to control the behavior of the actual circuit.

Example Code and/or Schematic

Consider the following example:

```
module mod_2_3_3_1test2(in1,clk,out1,out2);
    input in1,clk;
    output out1,out2;
    reg out1,out2;
    initial //violation
        begin
            out1 = 1'b0;
            out2 = 1'b0;
        end
    always @(posedge clk or negedge clk or in1)
        begin
            out1 <= in1;
            out2 <= ~in1;
        end
endmodule
```

In the above example, the STARCO5-2.3.4.2 Tag reports a violation as an `initial` construct is used in the design.

Default Severity Label

Caution

Tag Group

RTL Description Technique

Reports and Related Files

No related reports or files

STARC05-2.3.6.2a

Multiple sets/resets should not be used in an always block.

(Verilog)

Multiple asynchronous resets should not be used in a process block. (VHDL)

Description

Verilog

The STARC05-2.3.6.2a Tag flags always constructs that have multiple sets/resets. While some synthesis tools may be able to handle designs having always constructs with multiple sets/resets, such designs may not be portable.

VHDL

The STARC05-2.3.6.2a Tag flags process constructs that have multiple asynchronous resets.

Having a block with multiple resets is not advisable. It may be synthesized by synthesis tools but it is unlikely to be portable.

An example of such a process block is as follows:

```
library ieee;
use ieee.std_logic_1164.all

process(clk1, rst1, rst2)
begin
    if (rst1 = '1') then
        t <= '0';
    elsif (rst2 = '1') then
        t <= '0';
    elsif (clk1'event and clk1 = '1') then
        t <= not t;
    end if;
end process;
```

Message Details

Verilog

Following message appears when an always block contain multiple sets/

resets:

always block contains multiple sets/resets

VHDL

Following message appears when a process block contains multiple reset
<sig-list>:

Process block contains multiple resets <sig-list>

Severity

Recommended3

Verilog Examples

Consider the following example where the always construct has multiple resets (RST1 and RST2):

```
always @(posedge CLK or negedge RST1 or negedge RST2)
begin
    if (!RST1 || !RST2)
        Q = 0;
    else
        Q = DATA;
end
```

For this example, VC SpyGlass Lint generates the following message:
always block contains multiple resets

STARC05-2.3.6.2b

Asynchronous reset description should use negedge only.

Description

Verilog

The STARC05-2.3.6.2b Tag flags asynchronous reset descriptions defined using posedge.

All ASIC vendors use inverted logic for asynchronous reset pins. If your design uses posedge based reset description, the synthesis tools insert inverter cells in asynchronous reset line to bring the design in line with the design convention.

VHDL

The STARC05-2.3.6.2b Tag flags asynchronous reset descriptions defined using positive edge.

All ASIC vendors use inverted logic for asynchronous reset pins. If your design uses positive edge based reset description, the synthesis tools insert inverter cells in asynchronous reset line to bring the design in line with the design convention.

Message Details

Verilog

The following message appears at the location where an asynchronous reset description *<sig-name>* defined using posedge is encountered:

Asynchronous reset '*<sig-name>*' uses posedge

VHDL

The following message appears at the location where an asynchronous reset description *<sig-name>* defined using positive edge is encountered:

Asynchronous reset '*<sig-name>*' uses positive edge

Severity

Recommended3

Verilog Examples

Consider the following example where the posedge of the reset signal RESET has been used in the flip-flop description:

```
module test(IN1, RESET, CLK, OUT1);
    input [3:0] IN1;
    input RESET, CLK;
    output [3:0] OUT1;

    reg [3:0] OUT1;

    always @(posedge RESET or posedge CLK)
    begin
        if(RESET) OUT1 = 3'b000;
        else OUT1 = IN1;
    end
endmodule
```

For this example, VC SpyGlass Lint generates the following message:
Asynchronous reset 'RESET' uses posedge

STARC05-2.4.1.2

Latch descriptions should not be mixed with other descriptions in same module

Description

The STARC05-2.4.1.2 tag flags latch description that is in the same design unit with other descriptions.

Latch descriptions are not easily distinguishable from combinational circuits. In some cases, unintentional latches may be generated for combinational logic where case construct clauses are incomplete.

Hence, it is recommended to include latch descriptions in to a separate design unit.

NOTE: *Buffers and inverters will be allowed with latch descriptions.*

Message Details

The following message appears at the location where a latch description `<latch-name>` is encountered in a design unit that also has other logic descriptions:

Latch description '`<latch-name>`' mixed with other descriptions in same module

Severity

Recommended

Verilog Examples

Consider the following example where a latch description OUT is present with other logic descriptions:

```
module test (IN1, IN2, SET, OUT);  
  input IN1, IN2, SET;  
  output OUT;  
  reg OUT;  
  wire anded, net1;  
  
  assign anded = IN2 & net1;
```

```
always @(SET)
  if (SET)
    OUT = IN1;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Latch description 'OUT' mixed with other descriptions in same module

STARC05-2.5.1.1

Tristate descriptions must not be mixed with other descriptions in same module

Description

The STARC05-2.5.1.1 tag flags tristate buffers that are in the same design unit with other descriptions.

Separating tristate buffer descriptions from other descriptions helps in using timing-related synthesis directives only on the tristate buffers without affecting the timing analysis of the other descriptions.

Message Details

The following message appears at the location of the tristate buffer *<sig-name>* is encountered in a design unit that also has other logic descriptions:

Tristate description '*<sig-name>*' mixed with other descriptions in same module

Severity

Recommended3

Verilog Examples

Consider the following example where the tristate buffer description `tso` is in the same module with other descriptions:

```
module test (in1, in2, enb, out, tso);
  input in1, in2;
  input enb;
  output out, tso;
  reg out, tso;

  always@(enb or in1 or in2)
  begin
    tso = enb ? in2 : 1'bz;
    out = in1 && in2;
  end
end
```

```
endmodule
```

For this example, VC SpyGlass Lint generates the following message at the location where the tristate buffer is described:

```
Tristate description 'tso' mixed with other descriptions in  
same module
```

STARC05-2.6.2.2

A variable in sensitivity list is modified inside the same always block. (Verilog)

A variable in sensitivity list is modified inside the same process block. (VHDL)

Description

Verilog

Signal present in sensitivity list and modified inside the same always block indicates a potential error. A signal in sensitivity list should not be updated in the same always block as it may result in erroneous simulation results.

By default, the *STARC05-2.6.2.2* Tag does not report violation for always @* and always_comb. Set the value of the [check_implicit_senselist](#) parameter to yes to report violation for such cases

NOTE: *The STARC05-2.6.2.2 Tag supports generate-for block.*

VHDL

Signal present in sensitivity list and modified in the same process block indicate a potential error. A signal in sensitivity list should not be updated in the same always block as this may result in erroneous simulation results.

Message Details

Verilog

The following message appears when a signal/variable *<name>* is modified inside an always block in the hierarchy *<hier-path>*:

The signal/variable '*<name>*' is modified inside always block
[Hierarchy: '*<hier-path>*']

VHDL

The following message appears when a signal *<sig-name>* is modified inside a process block in the hierarchy *<ent-name>* *<arch-name>*:

signal '*<sig-name>*' is modified inside process block
[Hierarchy: '*<ent-name>* *<arch-name>*']

Severity

Mandatory

STARC05-2.7.1.3b

An "if" clause must have a corresponding "else" clause in describing sequential circuits. (Verilog)
An IF clause must have a corresponding ELSE clause in sequential circuits (VHDL)

Description

The STARC05-2.7.1.3b Tag flags if statements in sequential circuits that do not have corresponding else statements.

The Tag also reports violation when an `else` statement corresponding to an `if` statement is empty.

The else statement covers conditions that are not met by any of the preceding if or else if statements. A missing else statement may lead to inference of enabled flip-flops during synthesis. This Tag provides information to the user about all the sequential blocks where else is missing so that user can see, if the block is inferring an unwanted flip-flop.

NOTE: *The STARC05-2.7.1.3b Tag supports generate-for block.*

Message Details

Verilog

Message 1

The following message appears at the location of an if statement that does not have a corresponding else statement in a sequential block:

```
""if"" statement with condition ""<condition>"" in module
""<module-name>"" does not have a corresponding ""else""
statement in sequential block
```

Message 2

The following message appears when an if statement has an empty else block:

```
""if"" statement with condition ""<condition>"" in module
""module-name>"" have a corresponding ""empty-else"" statement
in sequential block
```

VHDL

The following message appears at the location of an if statement that does not have a corresponding else statement in a sequential block:

```
IF statement does not have a corresponding ELSE statement in  
sequential block
```

Severity

Recommended3

STARC05-2.7.2.2

Avoid describing case constructs

Description

The STARC05-2.7.2.2 tag flags case constructs that will never execute. Violation would be reported when

- Case label size exceeds the size of case select expression, where significant bit contains '1'.
- Case label completely overlaps with previous labels.
- In VHDL, Case label contains x, z. In Verilog, case label contains x, z or ? or Casez contains x.

If you include lines that will never be executed, you might make mistakes in coding other lines. Condition expressions should be coded carefully.

Message Details

The following message appears when a case construct is never executed:

<case/if> <label/branch> will <not execute/overlap> [Reason: <reason>] [Hierarchy: <hierarchy-name>]

Where, <reason> can be any of the following:

- size mismatch between case-label and case-select
- overlaps with previous case-labels
- case-label contains x, z, or ?

Severity

Recommended1

STARC05-2.7.2.3

Avoid null condition expressions.

Description

The STARC05-2.7.2.3 Tag flags all null condition expressions present in if or case block.

The STARC05-2.7.2.3 Tag checks the statement associated with every case constructs, if, and else-if statements, and reports violation if the statement is a null statement.

An always block can be used to code combinational logic, but if the if or case statement has any null condition expression, a latch is most likely to be generated. Even if an always block causes flip-flops to be inferred and contains any null condition expression, mistakes may occur in coding other condition expressions.

Message Details

The following message appears at the locations where an if or a case block contains a null condition expression:

Avoid using null condition expression

Severity

Recommended1

Examples

Example 1

Consider the following example containing a null condition expression:

```
always @ (a or b) begin
  case (a)
    4'h1: c = b[0];
    4'h2: ; // Do not leave a condition expression null
    4'h4: c = b[2];
    4'h8: c = b[3];
    default: c = 1'bx;
  endcase
end
```

Example 2

Consider the following example:

```
module testRtl167 (out0,out1,out2,out3,in,s0,s1);
output out0,out1,out2,out3;
reg out0,out1,out2,out3;
input in,s0,s1;
reg [3:0] encoding;
integer state;
always @(s1 or s0 or in or encoding)
begin
    case({s1,s0})
    2'b00 : begin
        out0 = in;
        out1 = 1'bz;
        out2 = 1'bz;
        out3 = 1'bz;
    end
    2'b01 : ;
    2'b10 : begin
        ;
    end
    endcase
end
endmodule
```

The Tag report violations for the above example because NULL statements are used within a case statement.

STARC05-2.8.1.5

Do not use the full_case directive. (Verilog)

Description

The STARC05-2.8.1.5 Tag flags `full_case` directives used in the design.

The `full_case` directive forces the synthesis tool to assume that all clauses including the default clause of the `case` construct have been specified. Thus, the pre- and post-synthesis simulation results of missing case clauses may be different.

Message Details

The following message appears at the location where a `full_case` directive is encountered:

"full_case" directive used

Severity

Mandatory

STARC05-2.8.3.3

Do not use `//synopsys full_case pragma` when all conditions are not described as case clause or the default clause is missing. (Verilog)

Description

The STARC05-2.8.3.3 Tag flags case statements when `synopsys full_case pragma` is specified for case statements when all the case conditions are not described or when default clause is missing.

This Tag reports violation when all conditions of a case statement are not described as case clauses and `//synopsys full_case` is specified for this case statement. This Tag will not report violation when default clause is described for the case statement even in the presence of the pragma `//synopsys full_case`.

Message Details

Following message appears when case select expression `<expr>` is specified with `synopsys full_case pragma` and the case statement does not have a default `<default>` case specified for the signal `<sig-name>`:

Case select expression '`<expr>`' detected with pragma `synopsys full_case` does not have `<default>` specified for signal '`<sig-name>`'

Severity

Reference

STARC05-2.8.3.4

A signal that is assigned don't care value in a case default clause should not be used in conditional constructs (if, if-elsif, case, ternary).

The STARC05-2.8.3.4 Tag runs the [STARC05-2.8.3.4a](#) and [STARC05-2.8.3.4b](#) Tags.

STARC05-2.8.3.4a

A signal that is assigned don't care value in a case default clause should not be used in if conditions. (Verilog)

A signal or variable, which is assigned don't-care value in an OTHERS clause should not be used in if-elsif conditions. (VHDL)

Description

Verilog

The STARC05-2.8.3.4a Tag flags `if` constructs where the condition expression uses signals that are assigned a don't care value in the default clause of case constructs.

When a signal that is assigned value `x` in the default clause of a case construct is used in the condition expression of an `if` construct, the `if` statement is assumed to be `FALSE` during RTL simulation and the corresponding `else` statement, if any is executed. However, the value of this signal may be fixed to be either 0 or 1 after synthesis. Thus, results of gate-level simulation may not match the results of RTL simulation.

NOTE: *The STARC05-2.8.3.4a Tag supports generate-for block.*

VHDL

The STARC05-2.8.3.4a Tag reports `if-elsif` constructs where the condition expression uses signals that are assigned a "don't care" value in the `others` clause of a case construct. When a signal, which is assigned value `x` in the `others` clause of a case construct, is used in the condition expression of an `if` construct, the `if` statement is assumed to be `false` during the RTL simulation and the corresponding `else` statement, if any, is executed. However, the value of this signal may be fixed to be either 0 or 1 after synthesis. Thus, results of gate-level simulation may not match the results of RTL simulation.

When a signal that is assigned value `X` in the `OTHERS` clause of a case construct is used in the condition expression of an `if` construct, the `if` statement is assumed to be `FALSE` during the RTL simulation and the corresponding `else` statement, if any is executed. However, the value of this signal may be fixed to be either 0 or 1 after synthesis. Thus, results of gate-level simulation may not match the results of the RTL simulation.

NOTE: *Records and multi-dimensional arrays are not handled in this Tag.*

NOTE: *The STARC05-2.8.3.4a Tag cannot be flagged beyond hierarchy boundary*

regardless of language (Verilog or VHDL).

Message Details

Verilog

The following message appears at the location of an `if` statement where the condition expression uses a signal `<sig-name>` that has been assigned a don't care value in the default clause of a case construct at line `<num>` in hierarchy `<hier-path>`:

Signal '`<sig-name>`' used in if condition was assigned value x in a case default clause (line: `<num>`)[Hierarchy: '`<hier-path>`']

VHDL

The following message appears at the location of an `if` statement where the condition expression uses a signal `<sig-name>` that has been assigned a don't care value in the OTHERS clause of a case construct:

Signal/Variable '`<name>`' (or some of its bits) used in if condition was assigned value X in a Case OTHERS clause

Severity

Recommended1

VHDL Example

Following example contains signal SIG that is assigned value x in the case OTHERS clause and is used in the if condition expression:

```
library ieee;
use ieee.std_logic_1164.all

PROC_1 : process
begin
    case expr is
        when "10" => ....
        when others => SIG <= 'X';
    end case;
end process PROC_1;
```


STARCO5 Tags

```
PROC_2: process
begin
    if (SIG = '1') then
        ....
    end if;
end process PROC_2;
```

STARC05-2.8.3.4b

A signal that is assigned don't care value in a case default clause should not be used in "ternary" and "case" constructs. (Verilog)
A signal that is assigned a don't care value in others clause should not be used in the case construct. (VHDL)

Description

Verilog

The STARC05-2.8.3.4b Tag flags ternary (?:) and case constructs checking signals that are assigned a don't care value in the default clause of a case construct.

When a signal that is assigned value x in the default clause of a case construct is used in a ternary or case construct, the ternary or case statement is assumed to be FALSE during RTL simulation. However, the value of this signal may be fixed to be either 0 or 1 after synthesis. Thus, results of gate-level simulation may not match the results of RTL simulation.

VHDL

The STARC05-2.8.3.4b Tag reports violation when the case select expression uses the signal that is assigned a don't care value 'x' in the others clause of the same/other case construct.

When a signal that is assigned value x in the others clause of a case construct is used in case construct, case statement is assumed to be false during RTL simulation. However, the value of this signal may be fixed to be either 0 or 1 after synthesis. Thus, results of gate-level simulation may not match the results of RTL simulation.

Message Details

Verilog

The following message appears at the location of a ternary construct or a case construct where the condition expression uses a signal `<sig-name>` that has been assigned a don't care value in the default clause of a case construct at line `<num>` in hierarchy `<hier-path>`:

```
Signal '<sig-name>' used in ternary or case condition was  
assigned value x in a case default clause (line:  
'<num>')[Hierarchy: '<hier-path>']
```

VHDL

The following message appears at the location of a case construct where the condition expression uses a signal *<name>* that has been assigned a don't care value in the OTHERS clause of a case construct:

Signal/Variable '<name>' (or some of its bits) used in case condition was assigned value X in Case OTHERS clause

Severity

Recommended1

STARC05-2.8.3.5

default clause in case constructs must be the last clause. (Verilog)

Description

The STARC05-2.8.3.5 Tag flags case constructs where the select expression uses a signal that is assigned don't care and the default clause does not assign don't care to a signal.

Do not use signals to which don't care conditions are assigned, for case construct selection expressions if the default clause does not assign don't care to signals. Because the don't care condition does not propagate without assigned in the default clause. Thus, the pre- and post-synthesis simulation results may differ.

Message Details

The following message appears at the location of a case construct clause that follows the default clause:

default clause is not the last clause in case construct

Severity

Mandatory

STARC05-2.8.5.1

Do not use the `parallel_case` directive. (Verilog)

Description

The STARC05-2.8.5.1 Tag flags `parallel_case` directives used in the design.

The `parallel_case` directive forces the synthesis tool to assume that all clauses of the `case` construct are mutually exclusive, even if there are overlapping clauses. Thus, the pre- and post-synthesis simulation results of the overlapping clauses may be different.

Message Details

The following message appears at the location where a `parallel_case` directive is encountered:

"`parallel_case`" directive used

Severity

Recommended1

STARC05-2.8.6.1

'case' statements and nested 'if' statements co-exist, better to have fewer conditionals with multiple matches than fewer matches with multiple conditionals.

Tag Description

The tag will report violation when if-statements are present in the case items of a case statement.

Tag Exceptions

- The tag will not check generate-case, generate-if combinations
- The tag will treat casex and casez statements as the general case statements

Message Details

Small conditions present in large case table '<case_selector>' [Hierarchy: '<hierarchy>']

Example Code and/or Schematic

Example 1

```
module test (sel, in1, in2, in3, a, result);
    input in1, in2, in3, a;
    input [1:0] sel;
    output reg result;

    always @ (*)
        case (sel) // violation
            2'b00: if (a) result = in1 & in2;
                    else result = in1 | in2;
            2'b01: if (a) result = !(in1 & in2);
                    else result = !(in1 | in2);
```

```

2'b10: if (a) result = !in1 & in2;
        else result = in1 | !in3;
2'b11: if (a) result = !(in1 & in2);
        else result = !(in1 | !in3);
endcase
endmodule

```

Example 2

```

module test (sel, in1, in2, in3, a, b, result);
    input in1, in2, in3, a, b;
    input [1:0] sel;
    output reg result;

    always @ (*)

```

case (sel) // no violation. the step count of the if-statements is the same, but their corresponding conditional expressions do not coincide.

```

2'b00: if (a) result = in1 & in2;
        else result = in1 | in2;
2'b01: if (b) result = !(in1 & in2);
        else result = !(in1 | in2);
2'b10: if (a) result = !in1 & in2;
        else result = in1 | !in3;
2'b11: if (a) result = !(in1 & in2);
        else result = !(in1 | !in3);
endcase
endmodule

```

Example 3

```
module test (sel, in1, in2, in3, a, b, result);  
    input in1, in2, in3, a, b;  
    input sel;  
    output reg result;
```

```
    always @ (*)
```

case (sel) // no-violation. corresponding conditional expressions of if-statements have coincided, and the count of their steps are the same. but that step count is higher than the case item count.

```
        1'b0: if (a) result = in1 & in2;  
            else if (b) result = in1 | in2;  
            else result = in1 | in3;  
        1'b1: if (a) result = in1 & ~in2;  
            else if (b) result = ~in1 | in2;  
            else result = !(in1 | in3);  
    endcase  
endmodule
```

Example 4

```
module test (sel, in1, in2, in3, a, b, result);  
    input in1, in2, in3, a, b;  
    input [1:0] sel;  
    output reg result;
```

```
    always @ (*)
```

case (sel) // no-violation. corresponding conditional expressions of if-statements have coincided, and the count of their steps are the same. but that step count is higher than the case item count.


```

2'b00, 2'b10:
    if (a) result = in1 & in2;
    else if (b) result = ~in1 & in2;
    else result = in1 | in2;
2'b01, 2'b11:
    if (a) result = !(in1 & in3);
    else if (b) result = ~in1 & in3;
    else result = !(in1 | in2);
endcase
endmodule

```

Example 5

```

module test (sel, in1, in2, in3, a, result, out);
    input in1, in2, in3, a;
    input [1:0] sel;
    output reg result;
    output reg out;

```

```

always @ (*)

```

case (sel) // no violation. the case items contain elements other than an isolated if-statement.

```

2'b00:
begin
    out = in1 & in3;
    if (a) result = in1 & in2;
    else result = in1 | in2;
end
2'b01:

```

```
begin
    if (a) result = !(in1 & in2);
    else result = !(in1 | in2);
end
2'b10:
begin
    if (a) result = !in1 & in2;
    else result = in1 | !in3;
end
endcase
endmodule
```

Example 6

```
module test (sel, in1, in2, in3, a, b, result);
    input in1, in2, in3, a, b;
    input [1:0] sel;
    output reg result;
```

```
    always @ (*)
```

case (sel) // no violation. the if-statements do not have the same number of steps.

```
2'b00: if (a) result = in1 & in2;
      else if (b) result = ~in1 & in2;
      else result = in1 | in2;
2'b01: if (a) result = !(in1 & in2);
      else result = !(in1 | in2);
2'b10: if (a) result = !in1 & in2;
      else result = in1 | !in3;
```

```

        2'b11: if (a) result = !(in1 & in2);
            else result = !(in1 | in3);
        endcase
    endmodule

```

Example 7

function my_func(input [1:0]i, input i2, a, b, c); //

similarly, the rule also will support for tasks, interfaces...

```

    case (i) // violation
        2'b00: if (i2) my_func = a & b;
            else my_func = a | b;
        2'b01: if (i2) my_func = !(a & b);
            else my_func = !(a | b);
        2'b10: if (i2) my_func = !a & b;
            else my_func = a | !c;
        2'b11: if (i2) my_func = !(a & b);
            else my_func = !(a | !c);
    endcase
endfunction

module test (sel, in1, in2, in3, a, result);
    input in1, in2, in3, a;
    input [1:0] sel;
    output reg result;
    always@(*)
        result = my_func(sel, in1, in2, in3, a); // function
call
endmodule

```

Severity

Warning

STARC05-2.9.1.2e

Do not use loop variable outside of the "for" loop. (Verilog)

Description

The STARC05-2.9.1.2e flags loop variables used outside the for loop.

NOTE: *The STARC05-2.9.1.2e Tag supports generate-for block.*

Parameter(s)

[ignore_auto_function](#): Default value is no. Set the value of the parameter to yes to ignore tag checking for automatic functions.

Message Details

The following message appears when a loop variable `<var-name>` is used outside the for loop at line `<num>`:

Loop variable '`<var-name>`' used outside of the for loop present at line no `<num>`

Severity

Mandatory

STARC05-3.1.4.4

Multiple assignments should not be made in one line. (Verilog)
Multiple assignment statements should not be made in one line (VHDL)

Description

The STARC05-3.1.4.4 Tag flags all lines that contain more than one assignment statement.

For VHDL designs, the STARC05-3.1.4.4 Tag checks the following types of assignment statements:

- Sequential signal assignment statements
- Sequential variable signal assignment statements
- Concurrent signal assignment statements
- Conditional signal assignment statements
- Selected signal assignment statements

By default, the STARC05-3.1.4.4 Tag flags VHDL design code lines that have more than one assignment statement. Set the value of the [strict](#) Tag parameter to `yes` or `STARC05-3.1.4.4` to check that more than one waveform of an assignment statement does not appear on the same code line.

A second assignment statement on a line can easily be overlooked and can create confusion.

Message Details

Verilog

The following message appears at the location where more than one assignment statement appears in the same line:

Multiple assignments in one line

VHDL

The following message appears at the location where more than one assignment statement appears in the same line:

Multiple assignment statements in one line

STARCO5 Tags

Severity

Recommended

STARC05-3.2.2.2b

Define macros should be read using include files. (Verilog)

Description

The STARC05-3.2.2.2b Tag flags ``define` declarations in source files.

To avoid excessive messages, the STARC05-3.2.2.2 Tag reports a message only for the first ``define` declaration in a source file and ignores other ``define` declarations in the source file.

It is recommended that ``define` directives are in a separate file and then include this file in the source files.

By default, the STARC05-3.2.2.2 Tag reports only one violation per file. The maximum violations per file can be specified by setting the [starc2005_define_viol](#) Tag parameter.

Message Details

The following message appears at the location where a ``define` directive is encountered:

Macro(s) declared in source file

Severity

Recommended3

STARC05-3.2.2.4

Include files must be specified with more than 1 level higher relative path. (Verilog)

Tag Description

The STARC05-3.2.2.4 tag reports violation when the include path is specified as follows:

- path is specified as full path name
- path is not specified (current directory)
- path is specified down the current directory

Message Details

The following message appears if an include file <file-name> has been specified with absolute path:

Include path \"<path-name>\" does not return to directory that is more than 1 level higher

Severity

Recommended1

STARC05-3.2.3.1

Port connections in instantiations must be made by named association rather than positional association. (Verilog)

Description

The STARC05-3.2.3.1 Tag flags positional port associations.

It is recommended to use port mapping by name. Port mapping by position or order may cause connection errors and is difficult to maintain in case the type, number, or name of ports change.

By default, the STARC05-3.2.3.1 Tag does not report violation for parameters in instantiation which are positional association. Set the value of the [check_param_association](#) parameter to `yes` to check for parameters in instantiation.

Message Details

The following message appears at the location where a port/parameter mapping by position is encountered:

Use instance port connections by name rather than by position

Here, `<port>` can have one of the following values:

- Port (Default value)
- Parameter (when the [check_param_association](#) parameter is set to `yes`)

Severity

Mandatory

Verilog Examples

Consider the following example where the port connection is by port order or position:

```
clkGen cg(IN1, CLK1);
```

The same example can be rewritten for port mapping by name as follows:

```
clkGen cg(.IN(IN1), .CLK(CLK1));
```

STARC05-3.2.3.1v

Port/Generic connections in instantiations must be made by named association rather than positional association. (VHDL)

Description

The STARC05-3.2.3.1v Tag flags positional port or generic associations.

The following examples indicate the requirement:

```
Foo_1 : Foo (out1 <= out1, in1 <= in1);
```

```
Foo_1 : Foo (out1, in1);
```

It is recommended to use port or generic mapping by name. Port or generic mapping by position or order may cause connection errors and is difficult to maintain in case the type, number, or name of ports or generics change.

Message Details

The following message appears at the location where a port mapping by position is encountered:

```
Use named association rather than positional association in  
instance port/generic maps
```

Severity

Mandatory

STARC05-3.2.3.2

Bit-width of the component port and its connected net must match. (Verilog)

Description

The STARC05-3.2.3.2 Tag flags bit-width mismatches between component ports and their connected nets.

The STARC05-3.2.3.2 Tag checks connections of module instances, gate cell instances, and direct port connections.

EDA tools handle bit-width mismatches in a way that may not match the original design intent.

The STARC05-3.2.3.2 Tag calculates the width of port expressions on the basis of the following conditions:

- If the [nocheckoverflow](#) Tag parameter is set to `yes`, width is calculated as per LRM. However, for constants, natural width is considered.
 - If the [nocheckoverflow](#) Tag parameter is set to `no`, width is calculated according to following methods:
- ? For plus/minus operator, value based width is considered. For example:


```
inst IN1 (.in1(a[2:0]+b[2:0]));
// Width of expression, a[2:0]+b[2:0], will be 4
(7+7=14, 4 bits wide)
```
 - ? For multiplication operator, width will be sum of operand widths. For example:


```
inst IN1 (.in1(a[2:0]*b[2:0]));
// Width of expression, a[2:0]*b[2:0], will be 6

inst IN2 (.in1(a1[2:0]*b1[2]));
// Width of expression, a1[2:0]*b1[2], will be 3
```
 - ? For division operator, LHS width will be considered. For example:


```
inst IN1 (.in1(a[2:0]/b[2]));
// Width of expression, a[2:0]/b[2], will be 3
```
 - ? For concat operator, width will be calculated as the sum of all operands. For example:

STARCO5 Tags

```

inst IN1 (.in1({1'b0,a[2:0]}));
// Width=3 bits (leading 0 ignored)

inst IN2 (.in1({1'b1,a[2:0]}));
// Width=4 bits (1+3 bits)

```

Also, set the value of the [use_lrm_width](#) Tag parameter to `yes` to consider LRM width, which is 32 bits.

By default, the STARCO5-3.2.3.2 Tag ignores leading zeros in concat/multiconcat operator while width calculation. Set the value of the [check_pad_concat](#) parameter to `yes` or `<Tag-name>` to consider leading zeros in concat/multiconcat operator while width calculation.

Message Details

The following message appears at the location of a module/gate instance `<inst-name>` where there is a width mismatch between the connected net and the port `<port-name>` of the parent module:

Bit width mismatch between port '`<port-name>`' of instance '`<inst-name>`' and its connected net [Hierarchy: '`<module-hier-name>`']

The following message appears at the location where there is a width mismatch between port `<port-name>` and its connected net:

Bit width mismatch between port '`<port-name>`' and its connected net [Hierarchy: '`<module-hier-name>`']

Severity

Mandatory

Verilog Examples

Example 1

Consider the following example where the bit-width of the port OUT of module `hier1` is different from that of net OUT of module `test`:

```

module test (OUT, IN);
  output [3:0] OUT;
  input IN;

```

```

    hier1 hier1(OUT, IN);
endmodule

```

```

module hier1(OUT, IN);
    output OUT;
    input IN;

```

```

    assign OUT = IN;
endmodule

```

For this example, VC SpyGlass Lint generates the following message:

Bit width mismatch between port 'OUT' of instance 'hier1' and its connected net [Hierarchy: 'test'].

To fix the message, rewrite the example to match bit-widths as follows:

```

module test (OUT, IN);
    output [3:0] OUT;
    input IN;

```

```

    hier1 hier1(OUT[0], IN);
endmodule

```

```

module hier1(OUT, IN);
    output OUT;
    input IN;

```

```

    assign OUT = IN;
endmodule

```

Example 2

Consider the following example:

```

inst IN1 (.in1({2'b0,a[2:0]}))

```

In the above example, the Tag calculates the width for {2'b0,a[2:0]} as 5 bits when the value of the [check_pad_concat](#) parameter is set to yes.

STARC05-3.2.4.3

Do not use defparam statements. (Verilog)

Description

The STARC05-3.2.4.3 Tag flags `defparam` statements in the design.

The `defparam` statements may not be synthesizable.

Message Details

The following message appears at the location where a `defparam` statement is encountered:

```
Defparam is used for parameter "<parameter-name>" in module  
"<module-name>" and it is not supported at Synthesis level
```

Severity

Recommended1

STARC05-3.5.3.1

A standard header should be provided in every source file

Description

The STARC05-3.5.3.1 Tag flags lines in the file header that do not conform to the recommended structure.

The exact format of the header should be adjusted to company standards.

By default, VC SpyGlass Lint checks for Verilog source file headers in the following format:

```
////////////////////////////////////
// FILE NAME : <file_name>
// FUNCTION : <function>
// AUTHOR : <author_name>
// CREATION DATE : <date>
// MODIFICATION HISTORY : <your comments>
// VERSION      Date      AUTHOR      DESCRIPTION
// <version> <date>      <name>        <comments>
// <version> <date>      <name>        <comments>
////////////////////////////////////
```

By default, VC SpyGlass Lint checks for VHDL source file headers in the following format:

```
-----
-- FILE NAME : <file_name>
-- FUNCTION : <function>
-- AUTHOR : <author_name>
-- CREATION DATE : <date>
-- MODIFICATION HISTORY : <your comments>
-- Version      Date      Author      Description
-- <version> <date>      <name>        <comments>
-- <version> <date>      <name>        <comments>
-----
```

VC SpyGlass Lint checks whether the file header has values for all the variables in exactly the same format as shown above.

Message Details

The following message appears for each file header line that does not conform to the default file header format:

Non-standard file header: <message>

Where <*message*> indicates the type of message.

Severity

Recommended

VHDL Examples

If a file header does not have the AUTHOR line, VC SpyGlass Lint generates the following message:

Non-standard file header: Expecting -- AUTHOR :

STARC05-1.2.1.2

Do not create an RS latch using primitive cells such as AND, OR

When to Use

Use this Tag to detect RS latches modeled using primitive cells.

Description

The STARC05-1.2.1.2 Tag reports violation for RS latches modeled using primitive cells.

Tag Exceptions

The STARC05-1.2.1.2 Tag does not check for flip-flops modeled using primitive cells as it is very difficult to detect. Also, it is unusual to infer flip-flops using primitive cells.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a latch `<latch>` has been modeled using primitive cells:

[MANDATORY] RS Latch '`<latch>`' inferred using primitive cells

Potential Issues

Violation may arise when a RS latch is described using primitive cells.

Consequences of Not Fixing

This type of coding style is treated as feedback to a combinational circuit

by timing analysis tool, which may create problem during Static Timing Analysis (STA).

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the location where a latch is modeled using primitive cells. Check the surrounding code that is used to model a latch.

Avoid using primitive cell for RS latch description. Else, use the `set_disable_timing` setting to avoid the effect of a feedback loop during timing analysis.

Example Code and/or Schematic

Consider the following example:

```
module RSLATCH1(R, S, Q, Qbar);
  input R,S;
  output Q,Qbar;
  wire w1,w2,w3;
  or MyOr(w3, R, S);
  or OR1(w1, R, Qbar);
  not N1(Q, w1);
  or OR2(w2, S, Q);
  not N2(Qbar, w2);
endmodule
```

In the above example, the STARCO5-1.2.1.2 Tag reports a violation as an `RSlatch` is inferred using primitive cells.

Default Severity Label

Mandatory

Tag Group

Basic Design Constraint

Reports and Related Files

No related reports or files

STARC05-1.3.1.3

Do not use asynchronous reset/preset signals as non-reset/preset or synchronous reset/preset signals

When to Use

Use this Tag to detect asynchronous reset/preset signals that are used as non-reset/preset signals or as synchronous reset/preset signals.

Description

The STARC05-1.3.1.3 Tag reports violation for asynchronous reset/preset signals that are used as non-reset/preset signals or as synchronous reset/preset signals.

The STARC05-1.3.1.3 Tag identifies all asynchronous reset/preset sources of a flip-flop (sources that are feeding the flip-flop reset/preset pin directly or through buffers or inverters) and then forward traverses the other paths from these reset/preset sources to determine whether the reset source is driving any signal other than the asynchronous reset/preset pin of a flip-flop. The forward traversal ignores any intermediate combinational logic.

Other non-reset/preset uses that are checked by STARC05-1.3.1.3 Tag are using a reset/preset signal as data, clock, or synchronous reset/preset pin of flip-flops, as latch inputs, as enable inputs of tristates, as select input of a mux.

When `enable_Tag_checking_per_ip` option is specified through the command line the Tag reports multiple violations, which are in different IPs, corresponding to an asynchronous reset/preset signal.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- `allow_combo_logic_base`: Default value is `no`. Set this parameter to `yes` to skip all VC SpyGlass Lint-generated combinational elements in the fan-in cone while finding the reset/preset source net.

- ***handle_latch_setreset***: Default value is no and this Tag considers the flip-flop's set/reset only as a start point. It reports a violation if the flip-flop's set/reset is going to another latch's set/reset pin. If you set to this parameter to yes, the *STARC05-1.3.1.3* Tag does not give violation starting from the flip-flop's set/reset and reaching to another latch's set/reset. The Tag also honors latch's set/reset as a start point in this case.
- ***reset_synchronizer_modname***: Set the value of this parameter to a comma separated list of module names. The specified modules are considered as reset synchronizers, and flip flops falling down the hierarchy of these modules are ignored during Tag checking.
- ***waiver_compat***: Default value is no. If you set the value of this parameter to yes or <Tag-name>, it ensures that the Tag does not generate the line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.
- ***report_all_rst***: Default value is no. If you set the value of this parameter to yes, the Tag reports all occurrences of non-reset/preset signals or synchronous reset/preset signals of asynchronous reset/preset signals in the RTL.
- ***allow_combo_logic_base***: Default value is no. Set this parameter to yes to skip all VC SpyGlass Lint-generated combinational elements in the fan-in cone while finding the reset/preset source net.
- ***ignore_cell***: The default value is not set. Set this parameter to a comma separated list of IP blocks that you want to ignore while finding reset signal used as non-reset signal respectively in the design. When this parameter is set, the Tag checks in fan-out of ignored IP blocks for a reset signal used as a non-reset signal.
- ***ignore_memory***: The default is set to no. Set this parameter to yes to ignore violations for reset signal usages inside or after memory cells.
- ***REPORT_UNDRIVEN_RESET***: The default value is yes. Set the value of the parameter no to avoid reporting violations for undriven reset source.
- ***REPORT_NONUSERMUX***: The default value is yes. Set the value of the parameter no to avoid reporting violations when a non-user defined RTL mux as non-reset usage.

- **REPORT_FILE_LINE_ON_SOURCE**: The default value is 0. Set the value of the parameter to 1 to report violations on back reference on RTL data net of fanout flop.

Constraint(s)

- **set_case_analysis**: Use this constraint to set the initial value for block path analysis.
- **assume_path**: Since the STARCO5-1.3.1.3 Tag supports block path analysis and does not traverse on the blocked path, use this constraint to traverse further in case of black boxes.

Messages and Suggested Fix

The following message appears at the location where a reset/preset signal *<rst-name>* of flip-flop *<flop-name>* is used as a non-reset/preset signal or as synchronous reset/preset in logic driving the instance *<inst-name>* at the input terminal *<inp-ter-name>* in file *<file-name>* at line *<num>*:

[RECOMMENDED1] Asynchronous *<reset/preset>* signal '*<rst-name>*' (flop: '*<flop-name>*') used as non-*<reset/preset>*/synchronous-*<reset/preset>* at instance '*<inst-name>.<inp-ter-name>*' (File Name: '*<file-name>*', Line no.: '*<num>*')
Suggested Fix: Use synchronous reset/preset signal.

Potential Issues

A violation is reported when an asynchronous reset/preset signals that are used as non-reset/preset signals or as synchronous reset/preset signals.

Consequences of Not Fixing

Design teams usually prefer a dominant usage of one of the reset/preset schemes (synchronous or asynchronous). Mixing the two schemes may result in partial resetting of the system and may not conform to the design intent of SoC.

There might be cases where there may not be any choice in assembly of IPs. However, most designers would want to follow a common scheme or at least be aware of synchronous and asynchronous reset/preset zones in the SoC. Additionally, there might be an impact on STA tasks. For asynchronous resets, the recovery times (de-assertion) need to be satisfied. For synchronous resets, set-up times need to be satisfied. Therefore, using an asynchronous reset/preset signal of a flip-flop as non-

reset/preset signal or synchronous reset/preset, makes the timing analysis of the path that passes through the asynchronous set/reset difficult.

How to Debug and Fix

The violation message contains the file-name and line number where signal is used as a non-reset signal. Double-click the violation message. The HDL Viewer window highlights the signal being used as reset.

Alternatively, view the Incremental Schematic of the violation message. The Incremental Schematic shows the signal being used as reset and as non-reset. Right-click on the signal and click *Cross-Reference to RTL*. The HDL Viewer window highlights the use of signal as a non-reset signal.

Example Code and/or Schematic

Verilog

Consider the following example where the asynchronous reset signal reset of flip-flop out1 is used as a non-reset signal in the flip-flop out2:

```
module test (out1, out2, in1, clk, reset);
  output out1, out2;
  input in1, clk, reset;
  reg out1, out2;
  always @ (posedge clk or negedge reset)
    if (!reset)
      out1 <= 1'b0;
    else
      out1 <= in1;
  always @ (posedge clk)
    out2 <= reset;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Asynchronous reset signal 'test.reset' (flop: 'test.out1') used as non-reset/synchronous-reset at instance 'test.out2_reg.D' (File Name: 'check.v' ,Line no.: '11')

The schematic for this example is given below:

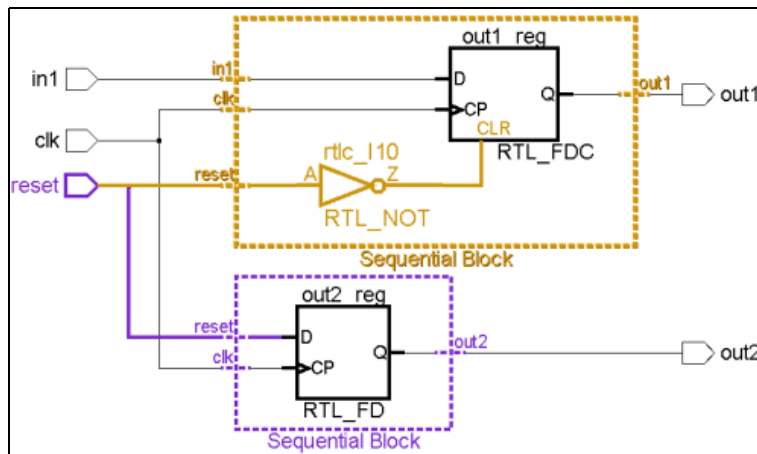


FIGURE 20. Incremental schematic

VHDL

Consider the following example where the asynchronous reset signal `reset` of flip-flop `output1` is also used as the data signal (non-reset use) of flip-flop `output2`:

```
library ieee;
use ieee.std_logic_1164.all

library ieee;
use ieee.std_logic_1164.all;
entity top is
  port(
    ck          : in std_logic;
    reset,data1  : in std_logic;
    output1,output2 : out std_logic
  );
end top;
architecture arc_top of top is
begin
  process ( ck, reset)
  begin
```



```

    if rising_edge(ck) then
        output2 <= reset;
    end if;
end process;

process( ck , data1, reset)
begin
    if (reset ='0') then
        output1<='0';
    elsif rising_edge(ck) then
        output1<=data1;
    end if;
end process;
end arc_top;

```

For this example, VC SpyGlass Lint generates the following message:

Asynchronous reset signal 'top.reset' (flop: 'top.output1') used as non-reset/synchronous-reset at instance 'top.output2_reg.D' (File Name: '1.3.1.3_S_Test1_pos.vhd' ,Line no.: '15')

The schematic for this example is given below:

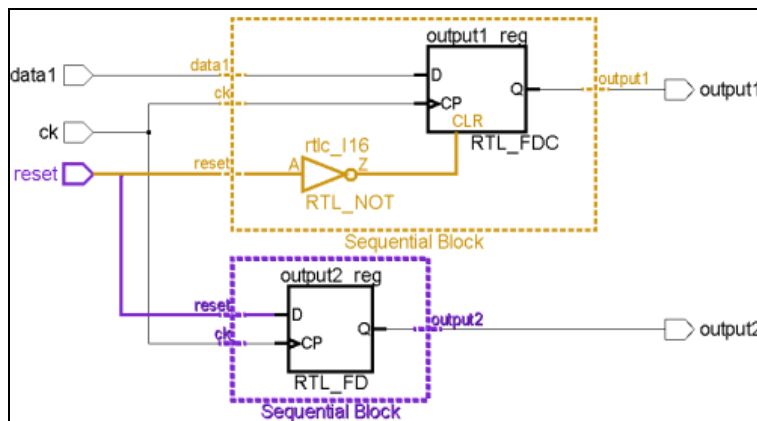


FIGURE 21. Incremental schematic

Default Severity| Label

Recommended1

Tag Group

Basic Design Constraint

Reports and Related Files

No related reports or files

STARC05-1.4.3.1a

Clocks should not be inverted

Description

The *STARC05-1.4.3.1a* Tag reports clocks that are inverted.

Inverted clocks have skew issues during layout. If required, clocks should be inverted in a separate top-level clock generation layer.

By default, the Tag reports only one violation for a clock source. If you set the [report_allclk](#) parameter to `yes`, the Tag reports all violations for a clock source.

By default, the [ignoreCellName](#) parameter is not set. Set this parameter to a PERL regular expression to specify the cell names that need to be ignored, through which the clocks are inverted.

Parameters

- [DISABLE_CLOCK_SORTING](#): By default, this parameter is set to 0. This parameter when set to 1 stops sorting of clocks from CDC API.
- [ignoreModName](#): By default, this parameter is not set. Set this parameter to a PERL regular expression to specify the module names to be ignored through which clocks are inverted.

The following message appears at the location where the flip-flop `<flop-name>` is inferred by using the inverted clock `<clk-name>` at the line number `<line_number>` in the file `<file_name>`:

Inverted clock `<clk-name>` drives flop `<flop-name>` (Line `<line_number>`, File: `<file_name>`)

Examples

Example 1:

Consider the following example, where the Tag reports a violation because the lib cell `CK_NOT` is not caught by the default regular expression value `""` as the [ignoreCellName](#) parameter is not set:

```
module top(input clk, a, b, sel, output reg y);
    wire w;
    always @ (posedge w)
```

```
begin
  y = 0;
  if (sel == 0) begin
    y = a;
  end else begin
    y = b;
  end
end
end
CK_NOT n(clk, w); //Violation
endmodule
```

Example 2:

Consider the following example, where the Tag does not report a violation because the lib cell CK_NOT is caught by the regular expression value "CK.*" as specified through the *ignoreCellName* parameter:

```
always @ (posedge w)
begin
  y = 0;
  if (sel == 0) begin
    y = a;
  end else begin
    y = b;
  end
end
end
CK_NOT n(clk, w); //No violation
endmodule
```

Severity

Recommended2

STARC05-1.4.3.1b

Clocks should not be gated

Description

The *STARC05-1.4.3.1b* Tag reports clocks that are locally gated.

By default, the [check_internally_generated_clk](#) parameter is set to no and the Tag does not check for internally generated clock nets.

Set the [check_internally_generated_clk](#) parameter to yes to enable the Tag to consider internally generated nets also during the Tag checking.

By default, the [ignoreCellName](#) parameter is not set. Set this parameter to a PERL regular expression to specify the clocks gated through lib cell instances that need to be ignored.

Parameters

- [ignoreModName](#): By default, the parameter is not set. Set this parameter to a PERL regular expression to specify the clocks gated through module instances that should be ignored. [Message Details](#)

The following message appears at the location where the flip-flop `<flop-name>` is inferred by using the gated clock `<clk-name>` at the line number `<line_number>` in the file `<file_name>`:

Gated clock `<clk-name>` drives flop `<flop-name>` (Line `<line_number>`, File: `<file_name>`)

Examples

Example 1

Consider the following example with the [ignoreCellName](#) parameter set as "CK.*":

```
module top0(input clk0, clk1, a, b, sel, output reg y);
    wire w;
    always @ (posedge w)
    begin
```

```
y = 0;
if (sel == 0) begin
    y = a;
end else begin
    y = b;
end
end
end

AN2 an2(clk0, clk1, w); //Violation, lib cell AN2 is not
//caught by the default regular expression value "CK*"
endmodule
```

Example 2

Consider the following example with the *ignoreCellName* parameter set as "CK.*":

```
module top1(input clk0, clk1, a, b, sel, output reg y);
    wire w;
    always @ (posedge w)
    begin
        y = 0;
        if (sel == 0) begin
            y = a;
        end else begin
            y = b;
        end
    end
end

CK_AN2 an2(clk0, clk1, w); //No Violation, lib cell CK_AN2 is caught by
// the regular expression value "CK.*"
endmodule
```

STARCO5 Tags

Severity

Recommended2

STARC05-1.4.3.2

Do not use flip-flop outputs as clocks of other flip-flops

When to Use

Use this Tag to report flip-flop outputs that are used as clocks of other flip-flops.

Description

The STARC05-1.4.3.2 Tag reports violation for the flip-flop outputs that are used as clock signals for other flip-flops in the design.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location where a flip-flop output `<flop1-name>` is used as the clock for another flip-flop `<flop2-name>`:

[MANDATORY] Flip-flop output '`<flop1-name>`' is used as clock of flip-flop '`<flop2-name>`'

Potential Issues

Violations may arise when there is a direct path from a flip-flop output to the clock input of another flip-flop in the design. There must be some intervening logic in such a path.

NOTE: *Buffers and inverters are not considered as intervening logic.*

Consequences of Not Fixing

Connecting flip-flop outputs to clock of other flip-flops may result in clock synchronization issues.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the flip-flop output is described. View the Incremental Schematic of the violation message. The Incremental Schematic shows the flip-flop output being used as clock.

Right-click on the clock terminal and click Cross-Reference to RTL. The HDL Viewer window highlights the use of signal as a clock.

To fix the violation, re-write the code so that the output of a flip-flop is not used as a clock signal of the other flip-flop.

Example Code and/or Schematic

Verilog

Consider the following example where the flip-flop output `tmp1` is used as the clock signal for flip-flop `OUT1`:

```
module test (IN1, IN2, CLK, OUT1);
    input IN1, IN2, CLK;
    output OUT1;

    reg OUT1, tmp1;

    always@(posedge CLK)
        tmp1 = IN1;

    always@(posedge tmp1)
        OUT1 = IN2;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:

Flip-flop output 'tmp1' is used as clock of flip-flop 'OUT1'

To fix this message, rewrite the `test` module so that the `tmp1` signal is not the output of a flip-flop.

VHDL

Consider the following example where the flip-flop output `ck1` is used as

the clock signal for flip-flop output2:

```
library ieee;
use ieee.std_logic_1164.all

process (ck, data1)
begin
    if rising_edge(ck) then
        ck1 <= data1;
    end if;
end process;

process (ck1, data2)
begin
    if rising_edge(ck1) then
        output2 <= data2;
    end if;
end process;
```

For this example, VC SpyGlass Lint generates the following message:

Flip-flop output 'ck1' used as clock of flip-flop 'output2'

Default Severity Label

Mandatory

Tag Group

Clocks

Reports and Related Files

No related reports or files.

STARC05-1.4.3.4

Do not use flip-flop clock signals as non-clock signals

When to Use

Use this Tag to detect flip-flop clock signals that are used as non-clock signals in other descriptions.

Description

The STARC05-1.4.3.4 Tag reports violation for the flip-flop clock signals that are being used as non-clock signals in other descriptions.

The Tag identifies all clock sources of a flip-flop (sources that are feeding the flip-flop clock pin directly or through buffers or inverters) and then forward traverses the other paths from these clock sources to determine whether the clock source is driving any signal other than the clock pin of a flip-flop. The forward traversal ignores any intermediate combinational logic.

For example, if a clock signal of one flip-flop is connected to the data pin of another flip-flop, the timing analysis becomes difficult. Other non-clock uses that are checked by the STARC05-1.4.3.4 Tag are using a clock signal as asynchronous sets and resets of flip-flops, as latch inputs, as enable inputs of tristates, or as select signals of multiplexers.

If `enable_Tag_checking_per_ip` option is specified through the command line, the Tag reports multiple violations out of the full set of all the occurrences of the clock as non-clock. Multiple violations, which are in different IPs, corresponding to a particular clock signal are reported if the `enable_Tag_checking_per_ip` option is specified.

Tag Exceptions

This Tag does not flag a violation for the clock driven by a constant value, which can be propagated through an assignment or using an SGDC file.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

- [*allow_combo_logic_base*](#): Default value is `no`. Set this parameter to `yes` to skip all VC SpyGlass Lint-generated combinational elements in the fan-in cone while finding the reset/preset source net.
- [*clk_EnableLatch*](#): The default value is `yes`. Set the value of the parameter to `no` to report the flip-flop clock signals that are also used as latch enable signals.
- [*report_allclk*](#): The default value is `no` and the STARCO5-1.4.3.4 Tag reports only the first occurrence of the non-clock usage of a clock source. Set the value of the parameter to `yes` to report all occurrences where a clock is being used as non-clock.
- [*check_clock_cell*](#): The default value is not set. Set this parameter to a comma separated list of IP blocks for which you want this Tag to consider clock signals as a non-clock signals. No violation is reported if the clock signals of specified IP blocks are used as non-clock signals in different logic.
- [*ignore_cell*](#): The default value is not set. Set this parameter to a comma separated list or regular expression of IP blocks that you want to ignore while finding clock signal used as non-clock signal in the design. When this parameter is set then Tag checks in fan-out of ignored IP blocks for a clock signal used as a non-clock signal.
- [*ignore_clk_generator*](#): The default value is `no`. Set the value of the parameter to `yes` to ignore violations for clock generator instances.
- [*REPORT_UNDRIVEN_CLOCK*](#): The default value is `yes`. Set the value of the parameter to `no` to avoid reporting violations when non-user defined RTL mux as non-clock usage.
- [*REPORT_UNDRIVEN_CLOCK*](#): The default value is `yes`. Set the value of the parameter to `no` to avoid reporting violations for undriven clock source.
- [*REPORT_FILE_LINE_ON_SOURCE*](#): The default value is `no`. Set the value of the parameter to `yes` to report violations on back reference on RTL data net of fanout flop.

Constraint(s)

- [*set_case_analysis*](#): Use this constraint to set the initial values for block path analysis.

- ***assume_path***: Use this constraint to specify the paths that exist between the input pins and the output pins of black boxes.

Messages and Suggested Fix

The following message appears at the location where a signal `<clk-name>` is declared that is being used both as a clock and signal and as a non-clock signal:

[MANDATORY] Clock signal '`<clk-name>`' used as a non-clock (Used with name '`<name>`')

Potential Issues

Violation may arise when clock signal of a flip-flop drives any signal other than clock pin of a flip-flop through some combinational logic.

Consequences of Not Fixing

In general, it is not advisable to use clock signals as non-clock signals. This can lead to race condition in the resulting logic and add to complexity of STA tasks. This can also impact portability and reusability of the design adversely. In some cases, the resulting logic can also be asynchronous adding to the complexity of functional verification.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic shows the clock signal used as non-clock. The clock path and the non-clock path are highlighted with different colors.

To fix the violation, ensure that you do not use clock signals as non-clock signals.

Example Code and/or Schematic

Verilog

Consider the following example where the clock signal CLK1 is also used as a non-clock signal:

```
module test (OUT1, IN1, CLK1, CLK2, RESET);
    input IN1, CLK1, CLK2, RESET;
    output OUT1;

    reg OUT1, out2;
```

```
always @ (posedge CLK1)
    OUT1 <= IN1;

always @ (posedge CLK2)
    out2 <= CLK1;
endmodule
```

For this example, VC SpyGlass Lint generates the following message:
clock signal 'CLK1' used as a non-clock signal

VHDL

Consider the following example where the signal `input` is used as a clock signal for flip-flop `output1` and is also used as a reset (non-clock use) signal for flip-flop `output2`:

```
library ieee;
use ieee.std_logic_1164.all

process(input, data1, reset)
begin
    if(reset = '1') then
        output1 <= "00000000";
    elsif(input'event and input='1') then
        output1 <= data1;
    end if;
end process;

process(clk, input, data2)
begin
    if(input = '1') then
        output2 <="00000000";
    elsif(clk'event and clk='1') then
        output2 <= data2;
    end if;
end process;
```

For this example, VC SpyGlass Lint generates the following message at the location where the signal `input` is declared:

STARCO5 Tags

clock signal 'input' used as a non-clock signal

Default Severity Label

Mandatory

Tag Group

Clocks

Reports and Related Files

No related reports or files

STARC05-2.1.1.1

Combinational logic should consistently be described using either always constructs or function and assign constructs. (Verilog)

Description

The STARC05-2.1.1.1 Tag flags inconsistent combinational logic modeling. There are two methods for describing combinational logic — using function and assign constructs and using always constructs. Only one of these styles should be consistently used in the design.

By default, the STARC05-2.1.1.1 Tag flags always constructs used for describing combinational logic. Use the [comb_logic_style](#) Tag parameter to specify the preferred modeling style.

By default, the [suggest_always_also](#) parameter is set to no and the Tag suggests only function+assign for function style in the violation message. Set this parameter to yes to enable the Tag to suggest always_comb and always@(*) also for function style in the violation message in addition to the default suggestion of function+assign.

Message Details

The following message appears at the location of an always construct used to describe combinational logic when the [comb_logic_style](#) Tag parameter is not specified or is specified to allow function and assign constructs:

'<always>' construct used to describe combinational logic rather than '<function+assign>' construct

The following message appears at the location of an assignment statement that uses a function call to describe combinational logic when the [comb_logic_style](#) Tag parameter is specified to allow always constructs:

'<function+assign>' construct used to describe combinational logic rather than '<always>' construct

The following message appears at the location of an assignment statement that uses a function call to describe combinational logic when the [suggest_always_also](#) parameter is set to yes:

'<always>' construct used to describe combinational logic

rather than '<function+assign+always_comb+always@(*)>' construct

Examples

Example 1

Consider the following example:

```
module top(input encoded_masks);
    reg decoded_masks;
    always @(encoded_masks)
    begin
        case (encoded_masks)
            1'b0: decoded_masks = 1'b0;
            default: decoded_masks = 1'b1;
        endcase
    end
endmodule
```

For the above example, the [comb_logic_style](#) parameter is set to function and the [suggest_always_also](#) is set to yes. The Tag reports the following violation message:

'always' construct used to describe combinational logic rather than 'function+assign+always_comb+always@(*)' construct

Example 2

Consider the following example:

```
module top(input encoded_masks);
    reg decoded_masks;
    always @(*)
    begin
        case (encoded_masks)
```

```
        1'b0: decoded_masks = 1'b0;  
        default: decoded_masks = 1'b1;  
    endcase  
end  
endmodule
```

For the above example, the [comb_logic_style](#) parameter is set to function and the [suggest_always_also](#) is set to yes. The Tag does not report any violation because the function is always @(*) .

Severity

Reference

STARC05-2.2.3.1

Non-blocking assignment should not be used in combinational always blocks (may lead to shoot through). (Verilog)

Description

The STARC05-2.2.3.1 Tag flags nonblocking assignment statements in always constructs of combinational logic.

Use of nonblocking assignment statement in always constructs of combinational logic may cause mismatch in pre- and post-synthesis behavior. During synthesis of combinational logic, statements are processed in the order as they appear. This processing order is different from the order intended by nonblocking assignment statements.

Set the [check_dependency](#) parameter to yes to not report violations for dependent assignment statements. By default, the value of the check_dependency parameter is set to no.

Message Details

The following message appears at the location where a nonblocking assignment statement is encountered in an always construct of a combinational circuit:

nonblocking assignment used in combinational always block

Severity

Recommended

STARC05-2.3.1.7a

A flip-flop should not have two asynchronous resets

Description

The STARC05-2.3.1.7a Tag flags flip-flops description having two asynchronous resets.

Flip-flops with two asynchronous resets lead to multiple gates in path of asynchronous reset to the corresponding flip-flop.

The STARC05-2.3.1.7a Tag reports only in those cases where the flip-flop does not have any set described, and its sensitivity list contains more than one signal besides the clock.

Message Details

The following message appears at the location where a block `<block-name>` (always or process block) has multiple asynchronous reset `<rst-name>`:

`<block-name>` block has multiple asynchronous reset
where `<block-name>` can be always or process block.

Severity

Mandatory

Verilog Example

Following flip-flop description has two asynchronous resets, RESET_X and LD_X:

```
always @ (posedge CLK or negedge RESET_X or negedge LD_X)
begin
    if (!RESET_X) Q <= #DELAY 8'h00;
    else if (!LD_X) Q <= #DELAY 8'h28;
    else Q <= #DELAY D;
end
```

STARC05-2.3.4.1v

Ensure that a flip-flop does not have an initial value in signal/variable description

When to Use

Use this Tag to identify signal or variable declarations that contain initial value and infer flip-flop

Description

The STARC05-2.3.4.1v Tag reports violation for signal or variable declarations, which have initial value and infer flip-flop.

Tag Exceptions

The STARC05-2.3.4.1v Tag does not flag a message if the flip-flop output is assigned to initial values in port declaration.

Default Weight

5

Language

VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location of a signal or variable declaration *<sig-name>* with initial value that is the output of a flip-flop:

[MANDATORY] Flip-flop '*<sig-name>*' has initial value in declaration

Potential Issues

A violation is reported when a flip-flop has an initial value in declaration.

Consequences of Not Fixing

Initial values in signal or variable declaration are ignored by synthesis tools.

How to Debug and Fix

Double-click the violation message. The HDL Viewer window highlights the line where the flip-flop output signal is assigned initial value.

View the Incremental Schematic of the violation message to see the corresponding flip-flop for the signal.

To fix the violation, do not use an initial value in a signal or variable declaration.

Example Code and/or Schematic

Example 1

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
    port ( in1, CLK : in std_logic;
          out1 : out std_logic );
end test;

architecture arch_test1 of test is
    signal sig : std_logic := '1';
begin
    process( in1, CLK )
    begin
        if( CLK'event and CLK = '1') then
            sig <= in1;
        end if;
    end process;
    out1 <= sig;
end arch_test1;
```

In the above example, the STARC05-2.3.4.1v Tag reports a violation as the flip-flop, sig, has an initial value in signal declaration.

Example 2

Consider the following example:

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
    port ( in1, CLK : in std_logic;
          out1 : out std_logic );
end test;

architecture arch_test1 of test is
    signal sig : std_logic;
begin
    process( in1, CLK )
    begin
        if( CLK'event and CLK = '1') then
            sig <= in1;
        end if;
    end process;
    out1 <= sig;
end arch_test1;

```

In the above example, the STARCO5-2.3.4.1v Tag does not report a violation as the flip-flop, sig, does not have an initial value in signal declaration.

Example 3

Consider the following example:

```

library ieee;
use ieee.std_logic_1164.all;

entity test1 is
    port ( in1, CLK : in std_logic;
          out1 : out std_logic := '1');
end test1;

architecture arch_test1 of test1 is
begin
    process( in1, CLK )
    begin

```

```
        if( CLK'event and CLK = '1') then
            out1 <= in1;
        end if;
    end process;
end arch_test1;
```

In the above example, the *STARCO5-2.3.4.1v* Tag does not report a violation as the flip-flop, out1 is assigned an initial value in port declaration.

Default Severity Label

Mandatory

Tag Group

Flip-Flop Inference

Reports and Related Files

No related reports or files.

STARC05-2.3.5.1

Flip-flops should not have fixed inputs

Description

The STARC05-2.3.5.1 Tag flags flip-flop descriptions that have a fixed data input.

The STARC05-2.3.5.1 Tag checks for flip-flop descriptions whose D input is connected to a supply source directly or via buffers and inverters.

NOTE: *The STARC05-2.3.5.1 Tag will not report a violation for the library cell flip-flop instances whose data pin is fixed but output pin is open.*

By default, the STARC05-2.3.5.1 Tag flags enabled flip-flops only when the data pin is held to a constant value. Set the [fixedinputenableflop](#) Tag parameter to `yes` to have the STARC05-2.3.5.1 Tag flag enabled flip-flops only when both enable pin and data pin are connected to supply.

A flip-flop with fixed input value cannot be tested properly.

Message Details

The following message appears at the location of a flip-flop description `<flop-name>` that has a fixed data input value `<value>`:

Flip-flop '`<flop-name>`' has fixed input value '`<value>`'

Severity

Recommended

STARC05-2.3.6.1

Do not mix descriptions of flip-flops with asynchronous reset and flip-flops without asynchronous reset in the same process/always block

Description

The STARC05-2.3.6.1 Tag flags those always or process constructs that have a flip-flop description with a reset line and another flip-flop description without a reset line.

The STARC05-2.3.6.1 Tag does not report violation for hanging flip-flops.

Mixing the descriptions of flip-flops with and without reset lines in the same always or process construct has the following issues:

- Such description may be confusing and difficult to debug
- Such description may also cause problems for synthesis tools.

When the *fast* parameter is set to yes, the RTL version of the Tag is run.

The following are the details of the RTL version of the Tag:

- All the signals present on the left hand side of the assignments belonging to a sequential block are assumed as flopped signals.
- Async reset/set detection over RTL is heuristic in nature, so flopped signals that are guarded by reset/set if condition is also heuristic.
- Tag reports one violation per always block over RTL.
- Reset detection over RTL is supported through if-else block only.
- RTL version of the Tag reports temporary variables also.
- The RTL version of the Tag checks and reports the violation even if module has a synthesis error.

By default, the STARC05-2.3.6.1 tag reports violation for internal signals. Set the value of the *ignore_internal_signal* parameter to yes to ignore violation for internal signals.

Message Details

The following message appears at the location where a flip-flop description *<flop-name>* without a reset is encountered in an always or process construct that also has a flip-flop description with asynchronous reset at line *<num>*:

uninitializable flip-flop '<flop-name>' coexists with flip-flops having asynchronous reset/set description in the same block (line: <line-number>). [Hierarchy: '<hier-path>']

If the *fast* parameter is set, the following message appears at the location where a flip-flop description <flop-name> without a reset is encountered in an always or process construct that also has a flip-flop description with asynchronous reset:

uninitializable flip-flop '<flop-name>' coexists with flip-flop '<flop-name>' having asynchronous reset/set description in the same block

For a description of a flip-flop with or without asynchronous reset, the previous respective description of the flip-flop without or with asynchronous reset is also highlighted.

Severity

Recommended1

Verilog Examples

Consider the following example that has a flip-flop description out1 with asynchronous reset res and flip-flop description out2 without reset, when the *fast* parameter is not set:

```
always @(negedge res or posedge clk)
begin
    if(!res)
        out1 = 1'b0;
    else
        begin
            out1 = in1;
            out2 = in2;
        end
    end
end
```

For this example, VC SpyGlass Lint generates the following message:

uninitializable flip-flop 'out2' coexists with flip-flops having asynchronous reset/set description in the same block (line: 7)

To fix this message, separate the two flip-flop descriptions in to different always constructs as follows:

```
always @(negedge res or posedge clk)
    if(!res)
        out1 = 1'b0;
    else
        out1 = in1;
always @( posedge clk)
    out2 = in2;
```

VHDL Examples

Consider the following example where the flip-flop out2 does not have a set/reset description while flip-flop out1 has an asynchronous reset description res, when the *fast* parameter is not set:

```
library ieee;
use ieee.std_logic_1164.all

process (clk,res)
begin
    if(res = '0') then
        out1 <= '0';
    elsif (clk'event and clk='1') then
        out1 <= in1;
        out2 <= in2;
    end if;
end process;
```

STARC05-1.1.1.10

Instance or module name must be different from the cell names of the ASIC library used. (Verilog)
Instance/Entity name should be different from those in the ASIC library (VHDL)

Description

The STARC05-1.1.1.10 Tag flags instance or module/entity names that are same as the cell names of the ASIC library specified using the `read_file -type gateslib <file>` command.

SpyGlass uses a default ASIC library (internal target library after synthesis) which is part of the SpyGlass installation. Please refer to the `rtl.c.prim.slf.lib` file located in the `SPYGLASS_HOME/auxiliary/gates` directory for the names of these internal cells. The STARC05-1.1.1.10 Tag also flags those instance or module/entity names that are same as the cell names in the internal target library even if you have not specified the `read_file -type gateslib <file>` command to pass a library.

It is recommended not to name instances or modules/entities in the design same as the cell names used in ASIC library to avoid confusion. In VHDL designs, this overlap causes a simulation failure.

Message Details

Verilog

The following message appears at the location where an instance or module name `<name>` is used that matches a cell name in the ASIC library specified using the `read_file -type gateslib <file>` command:

Instance/module name '`<name>`' matches with a ASIC library cell name

VHDL

The following message appears at the location where an instance or entity name `<name>` is used that matches an entity name in the ASIC library specified using the `read_file -type gateslib <file>` command:

`<type>` name '`<name>`' matches with cell name in ASIC library

Where `<type>` can be Instance or Entity.

Severity

Mandatory

STARC05-1.1.1.3

Do not use the reserved Verilog, SystemVerilog, or VHDL keywords. (Verilog)

Description

The STARC05-1.1.1.3 Tag flags names that are:

- Verilog and SystemVerilog reserved keywords and their case variants.
As Verilog-HDL is case sensitive, names that are case variants of Verilog reserved keywords are allowed. However, such names can be confusing.
- VHDL reserved keywords and their case variants.
Using VHDL keywords or their case variants in a Verilog design can hinder future migration to VHDL.
- EDIF, SDF, and Windows keywords and their case variants.

NOTE: *This Tag may flag duplicate violations when a UDP is declared within the v/y file after a module declaration.*

Following is the list of Verilog reserved keywords:

always	and	assign	automatic
begin	buf	bufif0	bufif1
case	casex	casez	cell
cmos	config	deassign	default
defparam	design	disable	edge
else	end	endcase	endconfig
endfunction	endgenerate	endmodule	endprimitive
endspecify	endtable	endtask	event
for	force	forever	fork
function	generate	genvar	highz0
highz1	if	ifnone	incdir
include	initial	inout	input
instance	integer	join	large
liblist	library	localparam	macromodule
medium	module	nand	negede
nmos	nor	noshowcancelled	not

notif0	notif1	or	output
parameter	pmos	posedge	primitive
pull0	pull1	pulldown	pullup
pulstyle_ondetect	pulstyle_on event	rcmos	real
realtime	reg	release	repeat
rnmos	rpmos	rtran	rtranif0
rtranif1	scalared	showcancelled	signed
smal	specify	specparam	strong0
strong1	supply0	supply1	table
task	time	tran	tranif0
tranif1	tri	tri0	tri1
triand	trior	trireg	unsigned
use	vectored	wait	wand
weak0	weak1	while	wire
wor	xnor	xor	

Following is the list of SystemVerilog reserved keywords:

alias	always_comb	always_ff	always_latch
assert	assume	before	bind
bins	binsof	bit	break
byte	chandle	class	clocking
constraint	const	context	continue
cover	covergroup	coverpoint	cross
dist	do	endclass	endclocking
endgroup	endinterface	endpackage	endprogram
endproperty	endsequence	enum	expect
export	extends	extern	final
first_match	foreach	forkjoin	iff
ignore_bins	illegal_bins	import	inside
int	interface	intersect	join_any
join_none	local	logic	longint

STARC05 Tags

matches	modport	new	null
package	packed	priority	program
property	protected	pure	rand
randc	randcase	randsequence	ref
return	sequence	shortint	shortreal
solve	static	string	struct
super	Tagged	this	throughout
timeprecision	timeunit	type	typedef
union	unique	uwire	var
virtual	void	wait_order	wildcard
with	within		

Following is the list of VHDL reserved keywords:

abs	access	after	all
architecture	array	attribute	block
body	buffer	bus	component
configuration	constant	disconnect	downto
elsif	entity	exit	file
generic	group	guarded	impure
in	inertial	is	lable
linkage	library	literal	loop
map	mod	next	of
on	open	others	out
port	postponed	procedure	process
pure	range	record	register
reject	rem	report	rol
ror	select	severity	shared
signal	sla	sll	sra
srl	subtype	then	to
transport	unaffected	units	until
variable	when		

Following is the list of EDIF, SDF, and Windows keywords:

ABSOLUTE	cell	celltype	edif
DELAY	HOLD	IOPATH	NET
VIEW	SETUP	CON	AUX
COM1	COM2	COM3	LPT1
LPT2	PRN	NUL	

Set the value of the [disable_vhdl_keywords](#) parameter to `yes` to ignore VHDL keywords. By default, this parameter is set to `no`.

Set the value of the [ignore_sv_keywords](#) parameter to `yes` to ignore the usages of SystemVerilog keywords or their case variants when SystemVerilog is not enabled. By default, this parameter is not set.

Message Details

The following message appears at the location where a Tag-violating object name `<name>` is first encountered:

verilog/VHDL reserved keyword `<name>` used

Severity

Mandatory

STARC05-1.1.1.3va

Do not use reserved keywords of Verilog-HDL and SystemVerilog. (VHDL)

Description

The STARC05-1.1.1.3va Tag flags names that are:

- Verilog and SystemVerilog reserved keywords and their case variants.

The use of Verilog and SystemVerilog reserved keywords as names hinder future migration to Verilog, which may be required for Verilog-based end-users.

- EDIF, SDF, and Windows keywords and their case variants.

Following is the list of Verilog reserved keywords:

always_comb	always_ff	always_latch	assume
before	bind	bins	binsof
break	byte	chandle	class
clocking	constraint	context	continue
cover	covergroup	coverpoint	cross
dist	do	endclass	endclocking
endgroup	endinterface	endprogram	endproperty
endsequence	enum	expect	export
extends	extern	final	first_match
foreach	forkjoin	iff	ignore_bins
illegal_bins	import	inside	int
interface	intersect	join_any	join_none
local	logic	longint	matches
modport	packed	priority	program
property	protected	rand	randc
randcase	randsequence	ref	sequence
shortint	shortreal	solve	static
struct	super	Tagged	this
throughout	timeprecision	timeunit	typedef
union	unique	uwire	var

virtual	void	wait_order	wildcard
within			

Following is the list of SystemVerilog reserved keywords:

always	assign	automatic	buf
bufif0	bufif1	casex	casez
cell	cmos	config	const
deassign	default	defparam	design
disable	edge	endcase	endconfig
endfunction	endgenerate	endmodule	endprimitive
endspecify	endtable	endtask	force
forever	fork	genvar	highz0
highz1	ifnone	incdir	include
initial	input	instance	join
large	liblist	localparam	macromodule
medium	module	negedge	nmos
noshowcancelled	notif0	notif1	output
parameter	pmos	posedge	primitive
pull0	pull1	pulldown	pullup
pulstyle_ondet ect	pulstyle_oneve nt	rcmos	realtime
reg	release	repeat	rnmos
rpmos	rtran	rtranif0	rtranif1
scalared	showcancelled	small	specify
specparam	strong0	strong1	supply0
supply1	table	task	tran
tranif0	tranif1	tri	tri0
tri1	triand	trior	trireg
vectored	wand	weak0	weak1
wire	wor		

Following is the list of EDIF, SDF, and Windows keywords:

STARCO5 Tags

ABSOLUTE	cell	celltype	DELAY
edif	HOLD	IOPATH	NET
VIEW	SETUP	CON	AUX
COM1	COM2	COM3	LPT1
LPT2	PRN	NUL	

Message Details

The following message appears at the location where a Tag-violating object name *<name>* is first encountered:

Reserved Verilog keyword '*<name>*' used

Severity

Mandatory

STARC05-1.1.1.3vb

Do not use standard VHDL names (VHDL)

Description

The STARC05-1.1.1.3vb Tag flags VHDL standard names used as object names.

While using the VHDL reserved keywords as objects names is syntax error, it is also recommended not to use the following as object names since they are standard VHDL names:

<code>std_logic</code>	<code>std_ulogic</code>
<code>std_logic_vector</code>	<code>bit</code>
<code>bit_vector</code>	<code>Boolean</code>
<code>character</code>	<code>conv_std_logic_vector</code>
<code>conv_integer</code>	<code>conv_unsigned</code>
<code>conv_signed</code>	<code>integer</code>
<code>string</code>	<code>signed</code>
<code>to_integer</code>	<code>to_stdlogicvector</code>
<code>unsigned</code>	

Message Details

The following message appears at the location where a Tag-violating object name *<name>* is first encountered:

standard VHDL keyword '*<name>*' used

Severity

Mandatory

STARC05-1.1.1.4

Do not use names starting with the user specified pattern (specified by the parameter 'starc2005_compare_str')(Verilog).

Description

The STARC05-1.1.1.4 Tag flags object names containing user-specified patterns.

By default, the STARC05-1.1.1.4 Tag flags object names containing VDD, VSS, GND, VCC, or VREF. Use the [starc2005_compare_str](#) Tag parameter to specify your patterns. The default value of the `starc2005_compare_str` Tag parameter is VDD, VSS, GND, VCC, VREF.

For VHDL designs, the STARC05-1.1.1.4 Tag checks object names of the following constructs:

entity	architecture	package
configuration	signal	port
variable	constant	attribute
generic	types	sub-types
library	function	component
process labels	instance	procedure

VDD, VSS, GND, or VCC are standard keywords used by many EDA tools with specific meanings. Hence, it is recommended not to use object names containing any of these keywords to avoid confusion and compatibility issues.

Consider the following example of the `starc2005_compare_str` Tag parameter being set as follows:

```
configure_lint_tag_parameter -tag STARC05-1.1.1.4 -parameter
starc2005_compare_str -value "Clock,reset"
```

The STARC05-1.1.1.4 Tag flags object names that start with Clock or reset.

Message Details

The following message appears at the location where an object name *<name>* is used that contains any of the patterns specified using the `starc2005_compare_str` Tag parameter:

Name '*<name>*' starting with the pattern `$starc2005_compare_str`

Severity

Mandatory

STARC05-1.1.1.6

Primary port names or module names must follow recommended naming convention. (Verilog)

Description

The STARC05-1.1.1.6 Tag flags primary port names or module names that end with an underscore or have consecutive underscores.

As per VHDL conventions, names must not end in an underscore. Such names may cause problems with other EDA tools.

Message Details

The following message appears at the location where a Tag-violating port name or module name *<name>* is declared:

Primary port/module '*<name>*' does not follow naming convention

Severity

Recommended1

STARC05-1.1.1.6v

Component name should be same as its corresponding entity name. (VHDL)

Description

The STARC05-1.1.1.6 Tag flags component names that are not same as their corresponding entity's name.

Following the recommended naming convention allows easier understanding of the design.

Message Details

The following message appears at the location where a Tag violating component name *<name>* of entity *<entity-name>* is declared:

Component name '*<name>*' not same as its corresponding entity name '*<entity-name>*'

Severity

Recommended1

STARCO5-1.1.1.7

Negative logic signals must follow recommended naming convention

Description

Verilog

The STARCO5-1.1.1.7 Tag flags the negative logic signal names that do follow recommended naming convention.

All scalar signals that are used inside conditional expressions in the format `!a` , `~a` , `a==1'b0`, or `a!=1'b1` are candidates for active low signals. Then, the STARCO5-1.1.1.7 Tag ignores those candidate signals that have been used in situations other than the following:

- In LHS of continuous assignments
- In RHS of continuous assignments as simple scalar expression
- In RHS of wire declarations as simple scalar expression
- In event control expressions
- In port connection expressions
- As module ports

Signals that are buffered (using direct assignment statements) to candidate signals meeting the above criteria are also candidates for active low signals. If any such buffered signal does not meet the criteria for being a valid active low signal, then both the original signal and the buffered signal are ignored.

Remaining candidate signals are Tag-checked.

Following the convention checked by this Tag ensures that such signals are clearly identifiable from other signals.

By default, the STARCO5-1.1.1.7 Tag flags those negative logic signal names that do not end with `_X` or `_N`. Use the [starc2005_negative_str](#) Tag parameter to specify your pattern.

By default, the STARCO5-1.1.1.7 Tag does not report a violation for an active low reset branch having non-static assignment in the RHS of an assign statement. Set the [report_nonstatic_actlow](#) parameter to `yes` to report violations for such cases.

For example, a violation is reported if the `report_nonstatic_actlow`

parameter is set to `yes` in the following expression:

```
if(reset == 0)
  q = din;
```

VHDL

The STARCO5-1.1.1.7 Tag flags the negative logic signal names that do follow recommended naming convention.

By default, the STARCO5-1.1.1.7 Tag flags those negative logic signal names that do not end with `_X` or `_N`. Use the [starc2005_negative_str](#) Tag parameter to specify your pattern.

The STARCO5-1.1.1.7 Tag checks for active low signals in the following types of conditional expressions:

- if statements

For example:

```
if (cond = '0') then ...
```

- Conditional signal assignments

For example:

```
q <= d when cond = '0';
```

- while constructs

For example:

```
while cond = '0' loop
  ...
end loop;
```

The following types of signals are considered active low signals:

- All scalar signals, which are used in conditional expression in the form of zero.

In the following example, the signal `cond` is considered an active low signal:

```
if (cond = '0') then ...
```

- Signals used with unary operator `not`.

In the following example, the signal `sig1` is considered an active low signal:

For example

```
signal sig1 : boolean;
if (not sig1) then ...
```

The following types of signals are not considered active low signals:

- Signals that are used as both active low and active high.

In the following example, signal `cond` is not considered an active low signal as it is used in conditional expressions in the forms one and zero:

```
if (cond = '0') then
...
end if;
```

```
if (cond = '1') then
...
end if;
```

- Signals that are used in conditional expression in the form of zero and the else statement also exists.

In the following example, the signal `cond` is not considered an active low signal:

```
if (cond = '0') then ...
else ...
end if;
```

- Clock signals that are driving on negative edge

Following the convention checked by this Tag ensures that such signals are clearly identifiable from other signals.

Message Details

The following message appears at the location where a Tag-violating signal `<sig-name>` is first encountered:

Active low signal name '`<sig-name>`' must end with `$starc2005_negative_str`

Severity

Recommended2

Verilog Examples

Consider the following example that has an active low signal SET:

```
module test (DIN, SET, Q, CLK);
  input DIN, SET, CLK;
  output Q;

  reg Q;

  always @(posedge CLK or negedge SET)
  begin
    if(!SET)
      Q = 1'b0;
    else
      Q = DIN;
    end
endmodule
```

For this example, SpyGlass generates the following message:

Active low signal name 'SET' must end with _X or _N.

To fix this message, rename the signal SET to SET_X or SET_N as follows:

```
module test (DIN, SET_X, Q, CLK);
  input DIN, SET_X, CLK;
  output Q;

  reg Q;

  always @(posedge CLK or negedge SET_X)
  begin
    if(!SET_X)
      Q = 1'b0;
    else
      Q = DIN;
    end
endmodule
```

STARC05-1.1.1.8

Module instance names should be same as or based on the module name. (Verilog)
Instance names should follow recommended naming convention. (VHDL)

Description

Verilog

The STARC05-1.1.1.8 Tag flags those module instance names that are not based on the module name.

The module instances should be named same as the module name or in `<modulename>_<num>` format.

Following this naming convention ensures that the instances of a module are clearly and easily located.

VHDL

The STARC05-1.1.1.8 Tag flags instance names that do not follow recommended naming convention.

It is recommended that instance names must start with U and must contain the master entity's name. Thus, the instance name must be `U<entity-name>` where `<entity-name>` is the name of the master entity and there is only one instance of the master entity. In case, there are more than one instances of the master entity, the instance names must be `U<entity_name>_<num>` where `<num>` is an integer.

NOTE: *The STARC05-1.1.1.8 Tag will not check in case of black box instantiations.*

Following the recommended naming convention allows easier understanding of the design.

NOTE: *The STARC05-1.1.1.8 Tag supports generate-for block.*

Message Details

Verilog

The following message appears at the location where a Tag violating module instance `<inst-name>` of module `<module-name>` is found:

Instance name '`<inst-name>`' should be in '`<module_name>[_<value>]`' format

VHDL

The following message appears at the location where a Tag-violating instance name *<inst-name>* is encountered:

Instance name '*<inst-name>*' doesn't follow the naming convention

Severity

Recommended3

Verilog Examples

Consider the following example where the instance name FF is not same as or based on the parent module Flop:

```
module test (IN, Q, CLK);
    input IN, CLK;
    output Q;

    wire Q;
    wire reset;

    Flop FF (.CLK(CLK), .DIN(IN1), .RES_X(reset), .QOUT(Q));
endmodule

module Flop (CLK, DIN, RES_X, QOUT);
    ...
endmodule
```

For this example, SpyGlass generates the following message:

Instance name 'FF' should be in 'Flop[_<value>]' format

To fix the message, change the instance name to be same as the module name (since there is only one module instance) as follows:

```
Flop Flop (.CLK(CLK), .DIN(IN1), .RES_X(reset), .QOUT(Q));
```

If there were more than one module instance, the instances should be named as Flop_1, Flop_2 and so on.

STARC05-1.1.1.8v

Instance names should follow recommended naming convention. (VHDL)

Description

The STARC05-1.1.1.8v Tag flags instance names that do not follow recommended naming convention.

It is recommended that instance names must start with U and must contain the master entity's name. Thus, the instance name must be `U<entity-name>` where `<entity-name>` is the name of the master entity and there is only one instance of the master entity. In case, there are more than one instances of the master entity, the instance names must be `U<entity_name>_<num>` where `<num>` is an integer.

You can specify a different naming convention for instances by using the [starc_inst_name](#) Tag parameter.

Following the recommended naming convention allows easier understanding of the design.

NOTE: *The STARC05-1.1.1.8v Tag will not check in case of black box instantiations.*

Message Details

The following message appears at the location where a Tag-violating instance name `<inst-name>` is encountered:

Instance name '<inst-name>' does not follow the naming convention. [Hierarchy: '<hier-path>']

Severity

Recommended3

STARC05-1.1.1.9a

Top-level module names should follow recommended naming convention. (Verilog)
Top-level entity names should follow recommended naming convention. (VHDL)

Description

Verilog

The STARC05-1.1.1.9a Tag flags those top-level module names that do not follow the recommended naming convention.

The recommended naming convention for top-level module names is as follows:

- The module names must use only alphanumeric characters and underscore.
- The module names must be all lowercase or all uppercase characters.

The naming convention of top-level module names can be modified using the [starc2005_top_mod_ent_name](#) Tag parameter.

VHDL

The STARC05-1.1.1.9a Tag flags those top-level entity names that do not follow the recommended naming convention.

The recommended naming convention for top-level entity names is as follows:

- The entity names must use only alphanumeric characters.
- The entity names must be all lowercase or all uppercase characters.

The naming convention of top-level entity names can be modified using the [starc2005_top_mod_ent_name](#) Tag parameter.

Message Details

Verilog

The following message appears at the location where a Tag violating module name *<mod-name>* is defined:

Top level module '*<mod-name>*' does not follow naming convention

STARCO5 Tags

VHDL

The following message appears at the location where a Tag violating entity name *<entity-name>* is defined:

Top level entity '*<entity-name>*' does not follow naming convention

Severity

Recommended1

STARC05-1.1.1.9b

Top-level module name should be less than or equal to 16 characters in length (Verilog)
Top-level entity name should be less than or equal to 16 characters in length (VHDL)

Description

Verilog

The STARC05-1.1.1.9b Tag flags top-level modules when length of the top-level module name is greater than the value specified by the [starc2005_max_mod_ent_char_length](#) Tag parameter.

VHDL

The STARC05-1.1.1.9b Tag flags top-level entity names when the length of the top-level entity name is greater than the value specified by the [starc2005_max_mod_ent_char_length](#) Tag parameter.

Message Details

Verilog

The following message appears when the top-level module name *<mod-name>* is not less than or equal to the value specified for \$starc2005_max_mod_ent_char_length Tag parameter:

Top-level module name '*<mod-name>*' should be less than or equal to \$starc2005_max_mod_ent_char_length characters in length

VHDL

The following message appears when the top-level entity name *<ent-name>* is not less than or equal to the value specified for \$starc2005_max_mod_ent_char_length Tag parameter:

Top-level entity name '*<ent-name>*' should be less than or equal to \$starc2005_max_mod_ent_char_length characters in length

Severity

Recommended1

STARC05-1.1.1.9c

Top-level port names should follow recommended naming convention

Description

The STARC05-1.1.1.9c Tag flags those top-level port names that do not follow the recommended naming convention.

The recommended naming convention for top-level port names is as follows:

- The port names should use only alphanumeric characters and the underscore.
- The port names should be all lowercase or all uppercase characters.

The naming conventions can be modified using the [starc2005_top_port_name](#) Tag parameter.

Message Details

The following message appears at the location where a Tag violating port name `<port-name>` is declared:

Top level port '<port-name>' does not follow naming convention

Severity

Recommended1

STARC05-1.1.1.9d

Top-level port name should be less than or equal to 16 characters in length.

Description

The STARC05-1.1.1.9d Tag flags top-level port names when the length of the top-level port name is greater than the value specified by the [starc2005_max_port_char_length](#) Tag parameter.

Message Details

The following message appears at the location where the top-level port name `<port-name>` is greater than the value specified by `starc2005_max_port_char_length` Tag parameter:

Top-level port name '<port-name>' should be less than or equal to \$starc2005_max_port_char_length characters in length

Severity

Recommended1

STARC05-1.1.2.1a

Module names should be between number of characters in length specified by parameters `starc2005_min_mod_ent_length` and `starc2005_max_mod_ent_length`. (Verilog)
Entity names should be between number of characters in length specified by parameters `starc2005_min_mod_ent_length` and `starc2005_max_mod_ent_length`. (VHDL)

Description

The STARC05-1.1.2.1a Tag flags those module names (Verilog) and entity names (VHDL) that are not within the specified limits.

By default, the STARC05-1.1.2.1a Tag flags those module names (Verilog) and entity names (VHDL) that are not between 2 and 32 characters. Use the [starc2005_min_mod_ent_length](#) and [starc2005_max_mod_ent_length](#) Tag parameters to set different minimum and maximum limits respectively.

A length of 16 or fewer characters is recommended.

By default, the STARC05-1.1.2.1a Tag checks the length of entity names only for VHDL designs. Set the value of the [strict](#) Tag parameter to `yes` or `STARC05-1.1.2.1a` to check the length of the architecture, package, package body, and configuration names.

NOTE: *The STARC05-1.1.2.1a Tag checks the size of all module or entity names including top-level module or entity names whereas [STARC05-1.1.1.9a](#) Tag checks the size of only top-level module or entity names (besides other checks).*

Message Details

Verilog

The following message appears at the location where a Tag violating module name `<name>` is defined:

```
Module name '<name>' is not between
$starc2005_min_mod_ent_length and
$starc2005_max_mod_ent_length characters
```

VHDL

The following message appears at the location where a Tag violating design unit name `<name>` is defined:

```
Design unit name '<name>' is not between
$starc2005_min_mod_ent_length and
```

\$starc2005_max_mod_ent_length characters

Severity

Mandatory

STARC05-1.1.2.1b

Instance names should be between number of characters in length specified by parameters `starc2005_min_inst_length` and `starc2005_max_inst_length`.

Description

The STARC05-1.1.2.1b Tag flags those instances that are not within the specified limits.

By default, the STARC05-1.1.2.1b Tag flags those instance names that are not between 2 and 32 characters. Use the [*starc2005_min_inst_length*](#) and [*starc2005_max_inst_length*](#) Tag parameters to set different minimum and maximum limits respectively.

A length of 16 or fewer characters is recommended.

Message Details

The following message appears at the location where a Tag violating instance name `<inst-name>` is encountered:

Instance name '`<inst-name>`' is not between `$starc2005_min_inst_length` and `$starc2005_max_inst_length` characters

Severity

Mandatory

STARC05-1.1.2.2

Top layer instance names should start with the layer identification character

Description

The STARC05-1.1.2.2 Tag flags those top-level layer instance names that do not start with the top-level layer identification character.

The first character of the name of the top-level module is the top-level layer identification character. All module instances in the top-level layer should start with this top-level layer identification character.

NOTE: *The STARC05-1.1.2.2 Tag is case-insensitive for VHDL designs and case-sensitive for Verilog designs.*

Following a hierarchical naming convention for instance names improves readability.

Message Details

The following message appears at the location where a top-level layer instance name `<inst-name>` does not start with the top-level layer identification character `<char>`:

Verilog

Top layer instance name '`<inst-name>`' should start with layer identification character '`<char>`'

VHDL

Top layer instance name '`<inst-name>`' does not start with layer identification character '`<char>`'

Severity

Recommended

Verilog Examples

Consider the following example where the module `test` contains an instant `isMod_1` of module `newMod_1` and the name of the instant does not start with `t` that is the starting character of the containing module `test`:

STARCO5 Tags

```
module test (out1, out2, in1, in2);  
    output out1, out2;  
    input in1, in2;  
  
    assign out1 = in1;  
    newMod_1 isMod_1 (out2, in2);  
endmodule  
  
module newMod_1 (out, in);  
    output out;  
    input in;  
  
    assign out = ~in;  
endmodule
```

For this example, SpyGlass generates the following message:

Top layer instance name 'isMod_1' should start with layer identification character 't'

STARC05-1.1.2.3

Instance names should start with the first character of the parent module name. (Verilog)
Instance name should start with layer identification character (VHDL)

Description

Verilog

The STARC05-1.1.2.3 Tag flags module, gate, or black box instance names that do not start with the first character of the parent module name (also known as the layer identification character).

The module that contains an instance is called the parent module of that instance.

Following this guideline improves the readability of the design.

VHDL

The STARC05-1.1.2.3 Tag flags instance names that do not start with the first character of the parent design unit name (also known as the layer identification character).

The parent design unit of an instance is the entity corresponding to the architecture that contains the instance. All instances in a layer should start with the layer identification character.

Following this guideline improves the readability of the design.

NOTE: *The STARC05-1.1.2.3 Tag is case-insensitive for VHDL designs and case-sensitive for Verilog designs.*

Message Details

Verilog

The following message appears at the location where an instance name `<inst-name>` does not start with the layer identification character (first letter of the parent module name `<module-name>`):

Instance name '`<inst-name>`' does not start with layer identification character (first character of parent module name '`<module-name>`')

STARCO5 Tags

VHDL

The following message appears at the location of a Tag-violating instance *<inst-name>*:

Instance name '*<inst-name>*' does not start with layer identification character '*<char>*'

Severity

Recommended

STARC05-1.1.2.4

Output port names should start with the first character of the module name. (Verilog)
Output pin name of each entity should start with the layer identification character followed by '_' (VHDL)

Description

Verilog

The STARC05-1.1.2.4 Tag flags output port names that do not start with the first character of the module name (also known as the layer identification character).

Following this guideline improves the readability of the design.

VHDL

The STARC05-1.1.2.4 Tag flags output pin names that do not start with the first character of the corresponding entity name (also known as the layer identification character) followed by an underscore.

Following this guideline improves the readability of the design.

NOTE: *The STARC05-1.1.2.4 Tag is case-insensitive for VHDL designs and case-sensitive for Verilog designs.*

Message Details

Verilog

The following message appears at the location where an output port name `<port-name>` does not start with the layer identification character (first letter of the module name `<module-name>`):

Output port name '`<port-name>`' does not start with layer identification character (first character of module name '`<module-name>`')

VHDL

The following message appears at the location where a Tag-violating output pin `<pin-name>` is declared:

Output pin name '`<pin-name>`' does not start with '`<char>_`'

STARCO5 Tags

Severity

Recommended

STARC05-1.1.2.5

Port names for each block should be in uppercase. (Verilog)
Port names for each entity should be in uppercase (VHDL)

Description

Verilog

The STARC05-1.1.2.5 Tag flags port names that are not in uppercase.

The recommended naming convention is to name ports in uppercase only so that they can easily be distinguished from other signals.

VHDL

The STARC05-1.1.2.5 Tag flags port names that are not in uppercase.

By default, the STARC05-1.1.2.5 Tag checks the port names in entity declarations only. Use the [strict](#) Tag parameter to check the port names in block constructs also.

The recommended naming convention is to name ports in uppercase only so that they can easily be distinguished from other signals.

Message Details

The following message appears at the location where a Tag violating port name `<port-name>` is declared:

Port name '`<port-name>`' should be in UPPERCASE

Severity

Recommended

STARC05-1.1.2.6

Connected net name of a module instance must be same as or based on the master module input/output port name. (Verilog)
Connected net name of a instance must be same as or based on the corresponding master entity input/output port name. (VHDL)

The STARC05-1.1.2.6 Tag runs the [STARC05-1.1.2.6a](#) and [STARC05-1.1.2.6b](#) Tags.

STARC05-1.1.2.6a

**Connected net name of a module instance must be same as or based on the master module output port name. (Verilog)
Connected net name in an instance must be same as or based on the corresponding master entity output port name (VHDL)**

Description

Verilog

The STARC05-1.1.2.6a Tag flags those nets connected to output ports of an instance that are not named the same as or are not based on the corresponding output port names of the master module.

The recommended practice is to name the connected net same as the corresponding output port of the master module or in

`<port-name>_<num>` format where `<port-name>` is the name of the output port.

NOTE: *The STARC05-1.1.2.6a Tag does not flag concatenated formals or actuals used in the instantiation.*

Example,

```
mod inst (in1, {out1, out2});
```

By default, the STARC05-1.1.2.6a Tag flags a maximum of 500 messages per design. Set the [allviol](#) Tag parameter to `yes` or `<Tag-name>` to report all violation messages.

Set the [checkFirstInstance](#) Tag parameter to have the STARC05-1.1.2.6a Tag to the following:

- Ignore I/O ports that are directly connected to the instance as in the following example:

```
module top(in1, out1);
    input in1;
    output out1;

    m2 m2_u(.a(in1), .b(out1));
endmodule

module my(a, b);
    input a;
```

```
output b;
...
endmodule
```

- Flag only one message for each module even if the module is instantiated multiple times in the design.

NOTE: *The STARC05-1.1.2.6a Tag supports generate-for block.*

VHDL

The STARC05-1.1.2.6a Tag flags those net names of an instance that are not same as or based on the corresponding output port names of the master entity.

The recommended practice is to name the connected net same as the corresponding output port of the master entity or in

`<port-name>_<num>` format where `<port-name>` is the name of the output port.

Set the [checkFirstInstance](#) Tag parameter to have the STARC05-1.1.2.6a Tag to the following:

- Ignore I/O ports that are directly connected to the instance
- Flag only one message for each design unit even if the design unit is instantiated multiple times in the design.

Message Details

Verilog

The following message appears at the location of a module instance where a net name `<net-name>` connected to an output port does not have the same name as the corresponding output port name `<port-name>` of the master module or is not in `<port-name>_<num>` format:

Net name '`<net-name>`' of module instance is not based on corresponding master module port name '`<port-name>`'

VHDL

The following message appears at the location of an instance where a net name `<net-name>` does not have the same name of the corresponding output port name `<port-name>` of the master entity or in `<port-name>_<num>` format:

Net name '<net-name>' in instance is not based on corresponding master module port name '<port-name>'

Severity

Mandatory

Verilog Examples

Consider the following example of a module instance where the name of the net `Q` connected to output port `OUT` of the instance `Foo_1` is not based on the output port name `OUT` of the master module `Foo`:

```
Foo Foo_1 (.OUT(Q), .IN(IN1))
```

For this example, SpyGlass generates the following message:

Net name 'Q' of module instance is not based on corresponding master module port name 'OUT'

To fix this message, change the name of the net `Q` to `OUT` as follows:

```
Foo Foo_1 (.OUT(OUT), .IN(IN1))
```

STARC05-1.1.2.6b

Connected net name of a module instance must be same as or based on the master module input port name. (Verilog)
Connected net name of a instance must be same as or based on the corresponding master entity input port name. (VHDL)

Description

Verilog

The STARC05-1.1.2.6b Tag flags those nets connected to input ports of an instance that are not named the same as or are not based on the corresponding input port names of the master module.

The recommended practice is to name the connected net same as the corresponding input port of the master module or in `<port-name>_<num>` format where `<port-name>` is the name of the input port.

By default, the STARC05-1.1.2.6b Tag flags a maximum of 500 messages per design. Set the [allviol](#) Tag parameter to `yes` or `<Tag-name>` to report all violation messages.

NOTE: *The STARC05-1.1.2.6b Tag does not flag nets connected to input ports that are used as clocks or resets in sequential always constructs.*

NOTE: *The STARC05-1.1.2.6b Tag does not flag concatenated formals or actuals used in the instantiation. For example:*

```
mod inst ({in1, in2}, out1, out2);
```

Set the [checkFirstInstance](#) Tag parameter to have the STARC05-1.1.2.6b Tag to the following:

- Ignore I/O ports that are directly connected to the instance as in the following example:

```
module top(in1, out1);
    input in1;
    output out1;

    m2 m2_u(.a(in1), .b(out1));
endmodule

module my(a, b);
    input a;
```

```
output b;
...
endmodule
```

- Flag only one message for each module even if the module is instantiated multiple times in the design

VHDL

The STARCO5-1.1.2.6b Tag flags those net names of an instance that are not same as or based on the corresponding input port names of the master entity.

The recommended practice is to name the connected net same as the corresponding input port of the master entity or in `<port-name>_<num>` format where `<port-name>` is the name of the input port.

Set the [checkFirstInstance](#) Tag parameter to have the STARCO5-1.1.2.6b Tag to the following:

- Ignore I/O ports that are directly connected to the instance
- Flag only one message for each design unit even if the design unit is instantiated multiple times in the design

Message Details

Verilog

The following message appears at the location of a module instance where a net name `<net-name>` connected to an input port does not have the same name as the corresponding input port name `<port-name>` of the master module or is not in `<port-name>_<num>` format:

Net name '`<net-name>`' of module instance is not based on corresponding master module port name '`<port-name>`'

VHDL

The following message appears at the location of an instance where a net name `<net-name>` does not have the same name of the corresponding input port name `<port-name>` of the master entity or in `<port-name>_<num>` format:

Net name '`<net-name>`' in instance is not based on corresponding master module port name '`<port-name>`'

STARCO5 Tags

Severity

Mandatory

STARCO5-1.1.3.1

**Internal signal names should have at least one lowercase alphabet.
(Verilog)**
**Internal signal names should have at least one lowercase character
(VHDL)**

Description

The STARCO5-1.1.3.1 Tag flags the internal signal names that do not have at least one lowercase alphabet.

For VHDL designs, the STARCO5-1.1.3.1 Tag checks for the signals inside entity, architecture, package, package body, and configuration.

Since input and output signals are recommended to be only in uppercase alphabets, the internal signal names must not be all uppercase alphabets to distinguish between the two. Thus, the signal names are recommended to have at least one lowercase alphabet.

Message Details

The following message appears at the location where a Tag violating internal signal name *<sig-name>* is found:

Verilog

Internal signal name '*<sig-name>*' should have at least one lowercase alphabet

VHDL

Internal signal name '*<sig-name>*' should have at least one lowercase character

Severity

Recommended

STARC05-1.1.3.3

Signal names, port names, parameter names, generic names, function names, and define names should be between 2 and 40 characters in length

The STARC05-1.1.3.3 Tag runs the [STARC05-1.1.3.3a](#), [STARC05-1.1.3.3b](#), [STARC05-1.1.3.3c](#), [STARC05-1.1.3.3d](#), and [STARC05-1.1.3.3e](#) Tags.

STARC05-1.1.3.3a

Signal name should be between 2 and 40 characters in length

Description

The STARC05-1.1.3.3a Tag flags signal names when the signal name length is not within the specified limits.

The signal name length should be more than the value specified by the parameter [starc2005_signal_min](#) Tag parameter and less than the value specified by the parameter [starc2005_signal_max](#) Tag parameter.

Message Details

The following message appears when the length of the signal name *<sig-name>* is not between the value specified by the `starc2005_signal_min` and `starc2005_signal_max` Tag parameters:

Signal name '*<sig-name>*' should be between \$starc2005_signal_min and \$starc2005_signal_max characters in length

Severity

Mandatory

STARC05-1.1.3.3b

Port name should be between 2 and 40 characters in length

Description

The STARC05-1.1.3.3b Tag flags port names when the port name length is not within the specified limits.

The port name length should be more than the value specified by the [starc2005_port_min](#) Tag parameter and less than the value specified by the [starc2005_port_max](#) Tag parameter.

Message Details

The following message appears when the port name <port-name> is not between the values specified for `starc2005_port_min` and `starc2005_port_max` Tag parameters:

Port name '<port-name>' should be between \$starc2005_port_min and \$starc2005_port_max characters in length

Severity

Mandatory

STARC05-1.1.3.3c

**Parameter name should be between 2 and 40 characters in length.
(Verilog)**
**Generic name should be between 2 and 40 characters in length.
(VHDL)**

Description

Verilog

The STARC05-1.1.3.3c Tag flags parameter names that are not within the specified limits.

The STARC05-1.1.3.3c Tag reports violation when parameter name length is less than the value specified by the [starc2005_parameter_min](#) Tag parameter or more than the value specified by the [starc2005_parameter_max](#) Tag parameter.

VHDL

The STARC05-1.1.3.3c Tag flags generic names when the generic name length is not within the specified limits.

The STARC05-1.1.3.3c Tag reports violation when the generic name length is less than the value specified by the [starc2005_parameter_min](#) Tag parameter or more than the value specified by the [starc2005_parameter_max](#) Tag parameter.

Message Details

Verilog

The following message appears when the length of the parameter name *<param-name>* is not between the values specified for the [starc2005_parameter_min](#) and [starc2005_parameter_max](#) Tag parameters:

Parameter name '*<param-name>*' should be between \$starc2005_parameter_min and \$starc2005_parameter_max characters in length

VHDL

The following message appears when the length of the generic name *<generic-name>* is not between the values specified for the [starc2005_parameter_min](#) and [starc2005_parameter_max](#) Tag

STARC05 Tags

parameters:

Generic name '<generic-name>' should be between
\$starc2005_parameter_min and \$starc2005_parameter_max
characters in length

Severity

Mandatory

STARC05-1.1.3.3d

Function name should be between 2 and 40 characters in length

Description

The STARC05-1.1.3.3d Tag flags function names whose length is not within the specified limits.

The STARC05-1.1.3.3d Tag reports violation when function name length is less than the value specified by the [starc2005_function_min](#) Tag parameter or more than the value specified by the [line_length_max](#) Tag parameter.

Message Details

The following message appears when the function name *<function-name>* is not between the values specified for `starc2005_function_min` and `starc2005_function_max` Tag parameters:

Function name '<function-name>' should be between \$starc2005_function_min and \$starc2005_function_max characters in length

Severity

Mandatory

STARC05-1.1.3.3e

Define macro names should be between 2 and 40 characters in length. (Verilog)

Description

The STARC05-1.1.3.3e Tag flags define macro names whose length is not within the specified limits.

The STARC05-1.1.3.3e Tag reports violation when the define macro name length is less than the value specified by the [starc2005_define_min](#) or more than the value specified by the [starc2005_define_max](#) Tag parameter.

By default, the STARC05-1.1.3.3e Tag reports only up to 500 violations. Set the [allviol](#) Tag parameter to `yes` or `<Tag-name>` to report all violation messages.

Message Details

The following message appears when the length of define macro name `<macro-name>` is not between the values specified for `starc2005_define_min` and `starc2005_define_max` Tag parameters:

Define macro name '<macro-name>' should be between \$starc2005_define_min and \$starc2005_define_max characters in length

Severity

Mandatory

STARC05-1.1.4.1

Include files must have extensions that match any of the extensions specified by the parameter `starc2005_ext_string_vlog`. (Verilog)

Description

The STARC05-1.1.4.1 Tag flags those ' INCLUDE statements that specify include files with non-recommended extensions.

By default, the STARC05-1.1.4.15 Tag flags those ' INCLUDE statements that specify include files having extensions other than `.h`, `.vh`, or `.inc`. Use the [starc2005_ext_string_vlog](#) Tag parameter to specify your preferred extensions.

Following the recommended naming convention helps file management.

Message Details

Verilog

The following message appears at the location of a ' INCLUDE statement that specifies a Tag-violating include file `<file-name>`:

```
Include file name '<file-name>' does not have
$starc2005_ext_string_vlog as extension
```

Severity

Recommended2

STARC05-1.1.4.1vb

Package and entity descriptions must not be mixed in same file (VHDL)

Description

The STARC05-1.1.4.1vb Tag flags source files that contain both package descriptions and entity descriptions.

Package descriptions must be placed in separate files to distinguish them from other VHDL descriptions.

Message Details

The following message appears for a Tag-violating file *<file-name>*:
Package and entity mixed in same file '*<file-name>*'

Severity

Recommended

STARC05-1.1.4.2a

Parameter names should follow recommended naming convention. (Verilog)

Description

The STARC05-1.1.4.2a Tag flags parameter names that do not follow the recommended naming convention.

By default, the STARC05-1.1.4.2a Tag flags those parameter names that do not start the name with `P_`. Use the [param_string](#) Tag parameter to specify your preferred naming convention.

Following the convention checked by this Tag ensures that parameters are clearly identifiable from other signals.

Message Details

Verilog

The following message appears at the location where a Tag-violating parameter `<name>` is defined:

Parameter name '`<name>`' should match pattern `$param_string`

Severity

Recommended3

STARC05-1.1.4.2b

Define macro names should start with "D_". (Verilog)

Description

The STARC05-1.1.4.2b Tag flags define macro names that do not follow the recommended naming convention.

The STARC05-1.1.4.2b Tag reports violation when the define macro name does not match the regular expression specified by the [starc2005_define_string](#) Tag parameter. This parameter specifies a regular expression, which should be used to name define macros.

Default value for the `starc2005_define_string` parameter is `/^D_[a-zA-Z0-9]*[a-zA-Z0-9]$/`.

Message Details

The following message appears when the define macro name *<macro-name>* does not follow naming convention:

Define macro name *<macro-name>* does not follow naming convention

Severity

Recommended3

STARC05-1.1.4.2v

**Constant names should follow recommended naming convention.
(VHDL)**

Description

The STARC05-1.1.4.2v Tag flags constant names that do not follow the recommended naming convention.

By default, the recommended naming convention for constant names is as follows:

- Names should start with C_
- All characters of the names should entirely be in uppercase
- Names should use only alphanumeric characters or underscore

Use the [const_string](#) Tag parameter to specify your preferred naming convention.

The default value of the `const_string` parameter is the regular expression `"/^C_/ and /^[A-Z0-9_]+$/"`.

Following the convention checked by this Tag ensures that constants are clearly identifiable from other signals.

Message Details

The following message appears at the location where a Tag-violating constant *<name>* is defined:

Constant name '*<name>*' does not follow naming convention

Severity

Recommended3

STARC05-1.1.4.3

**Do not use parameters with same name for different modules.
(Verilog)**

Description

The STARC05-1.1.4.3 Tag flags same parameter used in different modules.

Parameters used for the overall design must be distinguished from parameters used only under particular hierarchies.

The STARC05-1.1.4.3 Tag ignores parameters included in the module through include files.

Message Details

The following message appears at the location where a parameter `<name>` is defined in a module when a parameter with the same name was already defined in module `<du-name>` in file `<file-name>` at line `<num>`:

Same parameter '`<name>`' name is already used in parameter declaration at line `<num>` (File:`<file-name>`) in module '`<du-name>`'

Severity

Recommended3

STARC05-1.1.4.3v

Do not use constant declarations with same name in different entities/architectures. (VHDL)

Description

The STARC05-1.1.4.3v Tag flags same constant declared in more than one entity/architecture.

Using constants with same name in different entities should be avoided as it would be confusing when packaging constant declarations later.

Message Details

The following message appears at the location where a constant *<name>* is declared in an entity or an architecture *<name>* when a constant with the same name was already defined in design unit *<du-name>* in file *<file-name>* at line *<num>*:

Same constant '*<name>*' name is already used in constant declaration at line *<num>* (file:*<file-name>*) in *<type>* '*<name>*'

Where *<type>* can be entity or architecture.

Severity

Recommended3

STARC05-1.1.4.5

Parameters used only in a single layer must start with the layer identification character. (Verilog)

Description

The STARC05-1.1.4.5 Tag flags local parameter names or local text macro names that do not start with the first character of the parent module's name.

The layer identification character for a parameter or a text macro is the first character of the name of the module where it is declared.

The STARC05-1.1.4.5 Tag assumes that `'include` directives are used to include global parameters inside a module and thus all parameters directly specified within a module are local parameters.

Adding layer identification characters to parameter names or text macro names, which are used in a single layer would distinguish them from other parameters.

Message Details

The following message appears at the location where a Tag-violating local parameter or local text macro `<name>` is declared in parent module `<module-name>`:

Parameter name '`<name>`' does not start with layer identification character (first character of parent module name '`<module-name>`')

Severity

Reference

STARC05-1.1.4.7

Vector port bit-widths should be specified using parameters. (Verilog)
Port bit-width should be specified using generics. (VHDL)

Description

Verilog

The STARC05-1.1.4.7 Tag flags port vectors with directly specified bit-widths.

It is recommended to use parameters to specify vector port bit-widths.

Specifying bit-width for vector objects using parameters makes the design reusable and easily maintainable.

To avoid excessive messages, the STARC05-1.1.4.7 Tag reports only first five messages for a module. Set the value of the [strict](#) Tag parameter to `yes` or `STARC05-1.1.4.7` to view all messages.

NOTE: *The STARC05-1.1.4.7 Tag also flags port vectors where the bit-width is specified using text macros (``define` directive).*

VHDL

The STARC05-1.1.4.7 Tag flags port vectors with directly specified bit-widths.

The STARC05-1.1.4.7 Tag checks port declarations in entity declarations and blocks.

To avoid excessive messages, the STARC05-1.1.4.7 Tag reports only the first five offending port declarations. Set the [strict](#) Tag parameter to `yes` to report all messages.

NOTE: *The STARC05-1.1.4.7 Tag does not check top-level design units as the [STARC05-1.1.4.9](#) Tag requires that generics should not be used at the top-level.*

Message Details

Verilog

The following message appears at the location where a vector port `<port-name>` is declared without using parameters to specify the bit-width:

Bit-width specification of port '`<port-name>`' is not

parameterized

VHDL

The following message appears at the location where a Tag-violating port `<port-name>` is declared:

Bit-width specification of port '`<port-name>`' does not use generic

Severity

Recommended

Verilog Examples

Consider the following example where the declaration of vector port `in1` does not use parameters and the declarations of vector ports `in2` and `out2` uses parameters:

```
module test(in1, in2, out1, out2);
    parameter WIDTH=8;

    input [7:0] in1;
    input [(WIDTH-1):0] in2;
    output out1;
    output [WIDTH:0] out2;

    wire [7:0] in1;
    wire [2:0] xyz;

    ...
endmodule
```

For this example, SpyGlass generates the following message:

Bit-width specification of port '`in1`' is not parameterized

STARC05-1.1.4.8

Parameters with numeric value more than nine must have base type specification. (Verilog)

Description

The STARC05-1.1.4.8 Tag flags those parameter declarations with numeric value more than nine, which do not have base type specification.

Parameters of numeric value of 10 or more are assumed decimal values if the base type is not specified. This may not be the correct design intention.

NOTE: *The STARC05-1.1.4.8 Tag ignores all unused parameters inside the design and reports only the used parameters.*

You can set the value of the [ignore_parameter_in_port](#) parameter to `yes` to ignore the parameters that are used only in signal/variable range either at the time of declaration or used as bit-select of part-select. By default, the value of this parameter is set to `no`.

Message Details

The following message appears at the location where a Tag-violating parameter `<param-name>` is declared:

Specify base explicitly for parameter '`<param-name>`'

Severity

Recommended1

STARC05-1.1.4.8v

Initial value in vector constant declaration must have base specifier. (VHDL)

Description

The STARC05-1.1.4.8v Tag flags vector constant declarations, which do not use base specifier.

It is recommended that vector constant declarations must use one of the base specifiers — B"<bit_value>", O"<bit_value>", or X"<bit_value>".

Following examples violate the STARC05-1.1.4.8v Tag:

```
constant const1 : std_logic_vector(7 downto 0)
    := "10001011";
type address is array ( 0 to 7) of bit;
constant const2 : address := "11110000";
```

The above vector constants must be declared as follows:

```
constant const1 :std_logic_vector(7 downto 0)
    := B"10001011";
type address is array ( 0 to 7) of bit;
constant const2 : address := B"11110000";
```

Using base specifier to describe constant constructs makes debugging efficient and makes it possible to use the underscore delimiter.

NOTE: *It is not possible to use base specifiers for VHDL '87 std_logic_vector types. Thus, it is recommended to use VHDL '93 specification.*

Message Details

The following message appears at the location where a Tag-violating vector constant <name> is declared:

Base specifier not used for constant '<name>'

Severity

Recommended3

STARC05-1.1.4.9v

Generic must not be used at top-level module. (VHDL)

Description

The STARC05-1.1.4.9v Tag flags generics used at top-level.

Using generics at top-level may modify circuit structure by external parameters and therefore, using generic is not appropriate to create ASIC that is fixed object.

NOTE: *The STARC05-1.1.4.9v Tag does not report the following:*

- Generics that are declared at top-level but not used
- Generics that are used at top-level as delay parameters

Message Details

The following message appears at the location where a Tag-violating generic `<gen-name>` is declared:

Generic '`<gen-name>`' used at top-level

Severity

Recommended1

Examples

In the following example, the *STARC05-1.1.4.9v* Tag reports violation because generic is used at the top level:

```
entity top is
  generic ( status_in : integer := 2);
end top;
architecture arc2 of top is
  signal s1:integer;
begin
  s1<=status_in;
end arc2;
```

STARC05-1.1.6.1

Architecture name must contain "RTL". (VHDL)

Description

The STARC05-1.1.6.1 Tag flags architecture names that do not contain RTL.

Architectures must be named by its description type for easier understanding.

However, the STARC05-1.1.6.1 Tag reports violation if the name of an architecture does not contain RTL, irrespective of the type of description.

NOTE: *The STARC05-1.1.6.1 Tag is case-insensitive.*

Message Details

The following message appears at the first line of a Tag-violating architecture description `<arch-name>`:

Architecture name '`<arch-name>`' does not contain
`$starc2005_arc_name`

Severity

Recommended1

STARC05-1.1.6.4

An entity and its architectures must be described in same file (VHDL)

Description

The STARC05-1.1.6.4 Tag flags architecture descriptions if they are not in the same file as their corresponding entity descriptions.

If an entity and its corresponding architecture(s) are described in separate files, it is imperative to specify the files in the correct order so that the entity is compiled before its architectures. To avoid possible confusion, it is recommended that descriptions of an entity and its architecture(s) must be in the same file.

Message Details

The following message appears at the first line of an architecture description `<arch-name>` (described in file `<file1-name>`) if its corresponding entity description `<entity-name>` is in a different file `<file2-name>`:

Architecture '`<arch-name>`' (File: '`<file1-name>`') and its corresponding entity '`<entity-name>`' (File: '`<file2-name>`') are not in the same file

Severity

Mandatory

STARC05-1.3.1.2

Do not use synchronous reset in the design

Description

The STARC05-1.3.1.2 Tag flags synchronous resets used in the design.

The STARC05-1.3.1.2 Tag flags all synchronous resets used in the design but reports only the first use of each synchronous reset.

Synthesis tools may optimize the synchronous reset logic such that the corresponding sequential element may have ambiguous initial states.

NOTE: *The STARC05-1.3.1.2 Tag works with simple synchronous reset constructs only.*

Message Details

The following message appears at the location where a synchronous reset signal `<rst-name>` is first used to reset a flip-flop:

synchronous reset '`<rst-name>`' used

Severity

Mandatory

Verilog Examples

Consider the following example that uses the signal RST to synchronously reset the flip-flop OUT:

```
module test (OUT_CLK, CLK, RST, IN);
  input CLK, RST, IN;
  output OUT_CLK;

  reg OUT_CLK;

  always @ (posedge CLK)
  begin
    if (RST)
      OUT_CLK = 1'b0;
    else
      OUT_CLK = IN;
  end
endmodule
```



```

    end
endmodule

```

For this example, SpyGlass generates the following message:

Synchronous reset 'RST' used

To fix this message, modify the example to use the signal RST as asynchronous reset as follows:

```

module test (OUT_CLK, CLK, RST, IN);
    input CLK, RST, IN;
    output OUT_CLK;

    reg OUT_CLK;

    always @ (posedge CLK and negedge RST)
    begin
        if (!RST)
            OUT_CLK = 1'b0;
        else
            OUT_CLK = IN;
        end
    endmodule

```

VHDL Examples

Consider the following example where flip-flop output1 has a synchronous reset signal reset:

```

library ieee;
use ieee.std_logic_1164.all

architecture struct of test is
begin
    process(ck, data1)
    begin
        if rising_edge(ck) then
            if(reset = '0') then
                output1 <= '0';
            else

```

```
        output1<=data1;  
    end if;  
end if;  
end process;  
end struct;
```

For this example, SpyGlass generates the following message:

Synchronous reset 'reset' used

STARCO5-1.3.1.5a

Do not use Design Compiler synchronous set and reset directives.

Description

The STARCO5-1.3.1.5a Tag flags those tool-specific synthesis directives that control the synchronous sets/resets in the design.

Currently, SpyGlass checks for and flags the following synthesis directives:

Synthesis directives inserted as comments in a Verilog design

```
// synopsys sync_set_reset ...  
// synopsys sync_set_reset_local ...  
// synopsys sync_set_reset_local_all ...
```

Synthesis directives inserted as attributes in a VHDL design

```
attribute sync_set_reset of <signal-name>...  
attribute sync_set_reset_local of <signal-name>...  
attribute sync_set_reset_all of <signal-name>...
```

Designing using tool-specific synthesis directives has the following issues:

- Such designs cannot be used with other synthesis tools.
- Such directives are inserted as comments in Verilog designs. A mis-typed directive can cause unpredictable behavior as comments are not checked for correct syntax.

Message Details

The following message appears at the location where a tool-specific synthesis directive *<directive-name>* is encountered:

Design Compiler directive '*<directive-name>*' used

Severity

Recommended3

STARC05-1.3.1.5b

Do not use Design Compiler asynchronous set and reset directives.

Description

The STARC05-1.3.1.5b Tag flags those tool-specific synthesis directives that control the asynchronous sets and resets in the design.

Currently, SpyGlass checks for and flags the following synthesis directives:

Synthesis directives inserted as comments in a Verilog design

```
// synopsys async_set_reset ...  
// synopsys async_set_reset_local ...  
// synopsys async_set_reset_local_all ...
```

Synthesis directives inserted as attributes in a VHDL design

```
attribute async_set_reset of <signal-name>...  
attribute async_set_reset_local of <signal-name>...  
attribute async_set_reset_local_all of <signal-name>...
```

Designing using tool-specific synthesis directives has the following issues:

- Such designs cannot be used with other synthesis tools.
- Such directives are inserted as comments in Verilog designs. A mis-typed directive can cause unpredictable behavior as comments are not checked for correct syntax.

Message Details

The following message appears at the location where a tool-specific synthesis directive *<directive-name>* is encountered:

Design Compiler directive '*<directive-name>*' used

Severity

Recommended3

STARC05-1.4.3.6

Do not use Flip-Flops with inverted edges

Description

The STARC05-1.4.3.6 Tag flags flip-flops triggered at the negative edge of their clocks.

Depending on the ASIC library used, there may be two types of flip-flops — those that work on a positive clock edge and those that work on an inverted clock edge. When the two types of flip-flops are mixed in a circuit, there can be problems. It is best not to use flip-flops that work on an inverted clock.

Message Details

The following message appears at the location where the output of a flip-flop of clock *<clk-name>*, is first assigned when the flip-flop is triggered at the negative edge of the clock:

Inverted edge of clock '*<clk-name>*' used

Severity

Recommended

STARC05-1.6.1.4

Clock generation, reset generation, RAM, Setup/Hold ensure buffers, and I/O cells must be a module at top-level

Description

The STARC05-1.6.1.4 Tag flags those clock generation descriptions, reset generation descriptions, memories, setup/hold ensure buffers, and I/O pad cells that are not in the top-level layer of the design.

Inferred logic should not be placed in the top-level layer of the design. However, the clock and reset generation modules, memories, setup/hold ensure buffers and I/O pad cells can be placed at the top-level layer.

The STARC05-1.6.1.4 Tag accepts a user-defined list of module names and flags any of the specified modules' instantiations anywhere in the design except in the top-level layer.

Parameters

- [*top_gate_list*](#): By default, the STARC05-1.6.1.4 Tag flags the modules named `clockgenmodule` and `resetgenmodule` if these modules are not in the top-level layer of the design. You can specify to check for other module names using the [*top_gate_list*](#) Tag parameter.
- [*only_top_gate_list*](#): By default, the value of the parameter is no. Set the value of the parameter to yes to specify whether you can instantiate any other module, except the special modules, at the top level.

Message Details

The following message appears at the location where an instantiation of the user-specified module `<module-name>` is found in a layer other than the top-level layer of the design:

Instantiate module '`<module-name>`' only at the top-level

Severity

Mandatory

STARC05-2.1.1.2v

At least one of the packages IEEE.std_logic_arith and IEEE.std_logic_unsigned must be included in each entity. (VHDL)

Description

The STARC05-2.1.1.2v Tag flags those entities that include neither IEEE package std_logic_arith nor IEEE package std_logic_unsigned.

Unsigned arithmetic, conversion, and comparison functions for data types that are required for RTL design, are defined in IEEE package std_logic_arith and std_logic_unsigned. Thus, it is recommended to include at least one of these packages in every entity.

Message Details

The following message appears at the first line of an entity declaration `<entity-name>` that includes neither IEEE package std_logic_arith nor IEEE package std_logic_unsigned:

```
Neither package IEEE.std_logic_arith nor package  
IEEE.std_logic_unsigned included in entity '<entity-name>'
```

Severity

Recommended1

STARC05-2.1.1.3

When using IEEE.numeric_std, do not mix with std_logic_unsigned or std_logic_arith

Description

The STARC05-2.1.1.3 Tag flags cases where the IEEE package `numeric_std` is used with the `std_logic_unsigned` package or the `std_logic_arith` package.

The `numeric_std` package is a package similar to the `std_logic_arith` package and the IEEE standard. However, it is currently not commonly used. Although this package may become a main stream in the future, it is not recommended under these circumstances.

The `numeric_std` and `std_logic_arith` packages are common in the data type of unsigned and signed but differ in the name of type conversion function.

Message Details

The following message appears at the location of an entity `<entity-name>` that has objects of both the IEEE package `numeric_std` and of the `std_logic_unsigned` package or the `std_logic_arith` package

Package IEEE.numeric_std mixed with package IEEE.std_logic_arith or package IEEE.std_logic_unsigned in entity '<entity-name>'

Severity

Recommended

STARC05-2.1.10.1

Do not use block statement in the design (VHDL)

Description

The STARC05-2.1.10.1 Tag flags block constructs.

Guard expressions and block headers in block constructs are not supported by some logic synthesis tools.

Message Details

The following message appears at the first line of a block construct *<blk-name>*:

Block '<blk-name>' used in the design

Severity

Recommended1

STARC05-2.1.10.10

Alias declarations should not be used in the design. (VHDL)

Description

The STARC05-2.1.10.10 Tag flags alias declarations.

Some synthesis tools do not support alias declarations. Hence, such declarations should be avoided in the design.

Message Details

The following message appears at the location of an alias declaration
`<alias>`:

Alias '`<alias>`' used in the design

Severity

Recommended

STARC05-2.1.10.11

Bus and register signals should not be used in the design. (VHDL)

Description

The STARC05-2.1.10.11 Tag flags bus and register signals.

Some synthesis tools do not support signal declarations of bus and register types. Hence, such types should be avoided in the design.

Message Details

The following message appears at the location where a bus or register signal *<sig-name>* is used in the design:

Bus/Register signal '<sig-name>' used in the design

Severity

Recommended

STARC05-2.1.10.12

Disconnect specification must not be used in the design. (VHDL)

Description

The STARC05-2.1.10.12 Tag flags disconnect specifications.

Some synthesis tools do not support disconnect specifications. Hence, such specifications must not be used in the design.

Message Details

The following message appears at the location of a disconnect specification:

```
Disconnect specification used in the design
```

Severity

Mandatory

STARC05-2.1.10.13

Waveforms should not be used in the design. (VHDL)

Description

The STARC05-2.1.10.13 Tag flags multiple waveform elements in signal assignments.

In signal assignments and conditional signal assignments, multiple waveform elements should not be present in RHS. For example,

```
sig1 <= d1 after 1ns, d2 after 2ns;  
sig2 <= d1 after 1ns;
```

Some synthesis tools may not properly synthesize designs containing multiple element waveforms.

Message Details

The following message appears at the location of an assignment statement that has multiple waveform elements:

```
Multiple element waveform may not be synthesizable
```

Severity

Mandatory

STARC05-2.1.10.2

Do not use object of type record (VHDL)

Description

The STARC05-2.1.10.2 Tag flags data objects of type record.

The elements of record data types are divided into each element when generating logic.

Message Details

The following message appears at the location where a data object *<name>* of type record is declared:

Record object '*<name>*' declared

Severity

Recommended1

STARC05-2.1.10.3

Do not use shared variable in the design (VHDL)

Description

The STARC05-2.1.10.3 Tag flags shared variables.

Shared variable can pass signals between process statements without going through signal declarations. As signals have delta delay while variables do not, use of shared variables may result in unexpected behavior or circuit may be generated during RTL simulation or netlist generation after logic synthesis.

Message Details

The following message appears at the location where a shared variable `<var-name>` is declared:

shared variable '`<var-name>`' declared

Severity

Recommended1

STARC05-2.1.10.4

Do not use while-loop statement in the design (VHDL)

Description

The STARC05-2.1.10.4 Tag flags while-loop constructs.

In sequential circuits, while-loop constructs without wait statement, may not be supported by some synthesis tools.

Message Details

The following message appears at the location where a while-loop construct is encountered:

```
while loop statement used in the design
```

Severity

Recommended1

STARC05-2.1.10.5

Do not use procedure in the design (VHDL)

Description

The STARC05-2.1.10.5 Tag flags procedure constructs in the design.

Procedure constructs can describe multiple outputs and assign to signals immediately, which may result in unpredictable circuit behavior of synthesized design.

Message Details

The following message appears at the location where a procedure *<name>* is called:

Procedure '*<name>*' used in the design

Severity

Recommended1

STARC05-2.1.10.6

Do not use selected signal assignment in the design (VHDL)

Description

The STARC05-2.1.10.6 Tag flags selected signal assignments.

Selected signal assignment whose waveforms and target contain only static expressions is equivalent to a process statement whose final wait statement has no explicit sensitivity clause. Such statements execute once at the beginning of simulation and then are suspended permanently.

Message Details

The following message appears at the location where a selected signal assignment is encountered:

```
selected signal assignment used in the design
```

Severity

Recommended1

STARC05-2.1.10.8

Do not use attributes, which are defined in Synopsys library (VHDL)

Description

The STARC05-2.1.10.8 Tag flags Synopsys library attributes.

A synthesis tool-specific attribute has no effect when the design is used with any other synthesis tool.

Message Details

The following message appears at the location where a Synopsys library attribute `<attr-name>` is encountered:

Synopsys Attribute '`<attr-name>`' used in the design

Severity

Recommended2

STARC05-2.1.10.9

Access type should not be used in the design. (VHDL)

Description

The STARC05-2.1.10.9 Tag flags objects of access data type.

Access types must not be used in the design as these types may not be synthesized by some synthesis tools.

Message Details

The following message appears at the location where an object of access data type is specified:

Access type used in the design

Severity

Mandatory

STARC05-2.1.2.3v

Use range specification for integer types. (VHDL)

Description

The STARC05-2.1.2.3v Tag flags integer type object declarations without range specification.

The STARC05-2.1.2.3v Tag does not flag a message when natural subtype is used without range specification as natural subtype is pre-ranged from 0 to high in standard package.

Some synthesis tools may generate 32-bits wide comparator for the part of the design where integer data is compared as integers are considered to be 32-bits wide by default. Thus, it is recommended to explicitly specify the range of integer type objects to avoid generation of large circuits.

The STARC05-2.1.2.3v Tag does not report violation for the following:

- Integer type constants
- Integer type variables (whose class is not constant) when used as any of the following:
 - Vector index
 - Left or right part of range

Consider the following example:

```
variable INDEX_1 : INTEGER :=1;
    --range not specified
```

```
variable INDEX_2 : INTEGER :=2;
    --range not specified
```

```
variable NOT_INDEX : INTEGER :=3;
    --range not specified
```

```
A(INDEX_1) = '0'; -- INDEX_1 used as vector index
signal sig : std_logic_vector(INDEX_2 downto 0);
-- INDEX_2 used as left part of range

for I in 10 downto INDEX_2 loop
-- INDEX_2 used as right part of range

-- loop statements
```

```
end loop;
```

Here, SpyGlass does not report any violation for `I` in the `for` loop, as it is a `CONSTANT` that has an *implicit* declaration of type, `INTEGER`. This is also applicable to `index` used in the `for...generate` schemes.

Message Details

The following message appears at the location where an object `<obj-name>` is described without range specification:

Object '`<obj-name>`' has integer type without range specification

Severity

Mandatory

STARC05-2.1.2.6

Specify Range for `std_logic_vector` (VHDL)

Description

The STARC05-2.1.2.6 Tag flags objects of `std_logic_vector` data type that are specified without range specification.

Range for objects of `std_logic_vector` data type should be specified in the design except for function specification. If it is not specified, a range error may occur during simulation. Also, there is a possibility that logic synthesis tool generates circuit based on the bit-width interpretation, which is different from simulator's interpretation that is very hazardous.

Message Details

The following message appears at the location where an object `<obj-name>` of data type `std_logic_vector` is described without range specification:

Range is not specified for '`std_logic_vector`'

Severity

Mandatory

STARC05-2.1.3.1v

Linkage or buffer ports must not be described. (VHDL)

Description

The STARC05-2.1.3.1v Tag flags ports of type buffer or linkage.

Some synthesis tools may not support buffer and linkage ports.

By default, the STARC05-2.1.3.1v Tag flags buffer or linkage port declarations in entities only. Use the [strict](#) Tag parameter to flag buffer or linkage port declarations in block constructs also.

Message Details

The following message appears at the location where a Tag-violating port declaration `<port-name>` of type `<port-type>` is encountered:

```
<port-type> port '<port-name>' may not be synthesizable by some tools
```

Where `<port-type>` can be buffer or linkage.

Severity

Mandatory

STARC05-2.1.3.2

**Bit-width of function return value must match the bit-width of assignment destination signal. (Verilog)
Do not use buffer and linkage (VHDL)**

Description

Verilog

The STARC05-2.1.3.2 Tag flags the bit-width mismatch between the function return value and the signal being assigned.

If the function return value is wider than the destination signal, the function return value is left-truncated and the signal is assigned the truncated value.

If the function return value is narrower than the destination signal, the function return value is left-padded with zeros and the signal is assigned the padded value.

VHDL

The requirement of the STARC05-2.1.3.2 Tag is covered by the [STARC05-2.1.3.1](#) Tag.

Message Details

Verilog

The following message appears at the location where there is a bit-width mismatch between the return value of function `<func-name>` of bit-width `<bit-width1>` and the signal `<sig-name>` of bit-width `<bit-width2>` being assigned:

```
Bit-width mismatch between return value of function '<func-name>' ('<bit-width1>' bits) and assignment destination signal '<sig-name>' ('<bit-width2>' bits). [Hierarchy: '<hier-path>']
```

Where, `<hier-path>` is the hierarchical name of the instance (in `<module-name>:<inst-name>@<master-module-name>` format) where the Tag-violating construct is encountered.

Severity

Mandatory

Verilog Examples

Consider the following example where the signal `out_lower` (bit-width 1) is being assigned the return value of function `func` (bit-width 2) thus resulting in a bit-width mismatch:

```
module test (out, in1, in2);
    output out;
    input in1, in2;

    lower lower (out, in1, in2);
endmodule

module lower (out_lower, in1_lower, in2_lower);
    output out_lower;
    input in1_lower, in2_lower;

    assign out_lower = func (in1_lower, in2_lower);

    function [1:0] func;
        input in1_func, in2_func;
        begin
            func = {in1_func, in2_func};
        end
    endfunction
endmodule
```

For this example, SpyGlass generates the following message at the location of the signal assignment:

Bit-width mismatch between return value of function 'func'('2' bits) and assignment destination signal 'out_lower'('1' bits).
[Hierarchy: 'test:lower@lower']

STARC05-2.1.3.3

Port mode must be explicitly specified (VHDL)

Description

The STARC05-2.1.3.3 Tag flags ports in entities that have been described without explicit port mode.

If the port mode is not specified for a port, the port is assumed to be an input (in) port. This may not be design intent.

By default, the STARC05-2.1.3.3 Tag checks the port declarations in entity declarations only. Set the value of the [strict](#) Tag parameter to `yes` or `STARC05-2.1.3.3` to check port declarations of block constructs also.

Message Details

The following message appears at the location where a port declaration `<port-name>` without explicit port mode is encountered:

No explicit mode specified for port '`<port-name>`'

Severity

Mandatory

STARC05-2.1.3.5

**Global signals must not be assigned in function description.
(Verilog)**

Description

The STARC05-2.1.3.5 Tag flags assignments to global signals inside a function description.

As the function should return only one value, assigning both the function return value and a global signal can be confusing and generates redundant logic during synthesis.

Message Details

The following message appears at the location where a global signal `<var-name>` is assigned in a function description:

Global variable '`<var-name>`' is assigned in a function description

Severity

Mandatory

STARC05-2.1.4.2

Bit-wise operators should be used instead of logic operators in single-bit operations. (Verilog)

Description

The STARC05-2.1.4.2 Tag flags logical operations on scalar operands.

While the results of both bit-wise operations and the logical operations on scalar operands is same, it is recommended to use bit-wise operators for consistency with vector operands (see the [STARC05-2.1.4.5](#) Tag).

No violation is reported for logical negation on integer variables.

Message Details

The following message appears at the start of a statement in which a logical operator `<op-name>` is used for a single-bit operand:

Use bit-wise operator instead of logical operator '<op-name>'

Severity

Reference

Verilog Examples

Consider the following example where logical operation using the not (!) operator is being performed in scalar operand (scalar result of [in1 & in2 | in3]):

```
module test (out, in1, in2, in3);
    output out;
    input in1, in2, in3;

    assign out = !(in1 & in2 | in3);
endmodule
```

For this example, SpyGlass generates the following message at the start of the statement in which the logical operator (!) is being used:

Use bit-wise operator instead of logical operator '!'

STARC05-2.1.4.2v

Shift operator should not be used in the design. (VHDL)

Description

The STARC05-2.1.4.2v Tag flags expressions using the following shift operators:

sll	srl	sla	sra	rol	ror

Expressions using the shift operators may not be supported by some logic synthesis tools. Thus, it is recommended to use the concatenation operator for shift operations.

Message Details

VHDL

The following message appears at the location where an expression `<expr>` using a shift operator is encountered:

shift operator used in expression '`<expr>`'

Severity

Reference

STARC05-2.1.4.6a

Reduction of a single-bit expression should not be performed (Verilog)

Description

The STARC05-2.1.4.6a Tag flags expressions using reduction operation on single-bit operands.

Reduction operators (&, |, and ^) reduce a vector operand to a single-bit value. Hence, reduction operation on single-bit operands is redundant.

Message Details

The following message appears at the location where a reduction operation using reduction operator *<operator>* is performed on a single-bit operand *<operand>*:

Use of unary reduction operator "<operator>" on a 1-bit operand "<operand>" is redundant

Severity

Recommended3

STARC05-2.1.4.6b

Reduction of Large expression should not be performed (Verilog)

Description

The STARC05-2.1.4.6b flags expressions where reduction operator is performed on large operands.

By default, the STARC05-2.1.4.6b Tag flags reduction operations on operands of bit-width more than 32 bits. Use the [bit_width_reduction](#) Tag parameter to specify a different number.

Reduction operators reduce a vector operand to a single-bit value. When used with large operands, the reduction function is performed at multiple levels.

Message Details

The following message appears at the location where a where a reduction operation using reduction operator *<operator>* is performed in an operand *<operand>* of bit-width *<num>* where *<num>* is more than the 32 or the number specified by the `bit_width_reduction` Tag parameter:

Unary reduction operator "*<operator>*" used on large *<num>*-bit operand "*<operand>*"

Severity

Caution

STARC05-2.1.5.1v

When-else nesting in conditional signal assignment statement must not exceed 5 levels. (VHDL)

Description

The STARC05-2.1.5.1v Tag flags conditional signal assignment constructs with deep when-else nesting.

Deep when-else nesting makes it difficult to understand the code and reduces the readability.

By default, the STARC05-2.1.5.1v Tag flags conditional signal assignment constructs with more than 5 levels of when-else levels. Use the [whenelse_cnt](#) Tag parameter to specify a different number.

Message Details

The following message appears at the location of a conditional signal assignment constructs with more than the specified number `<num>` of when-else nesting:

```
More than <num> levels in when-else nesting in conditional  
signal assignment statement
```

Severity

Recommended1

STARCO5-2.1.6.1

**Array specification should follow recommended convention.
(Verilog)**

Description

The STARCO5-2.1.6.1 Tag flags array specifications that are not in [MSB:LSB] format.

Arithmetic operations on arrays work on the assumption that the arrays are defined in the [MSB:LSB] format where LSB is less than MSB. Such operations on reverse arrays require complicated conversions.

By default, the STARCO5-2.1.6.1 Tag ignores multi-dimensional arrays and memory. Set the value of the *strict* Tag parameter to *yes* or STARCO5-2.1.6.1 to check multi dimensional arrays and memory also.

Message Details

The following message appears at the location where an array *<array-name>* is not specified in [MSB:LSB] format:

Declare array '*<array-name>*' as [MSB:LSB].[Hierarchy: '*<hier-path>*']

Severity

Recommended2

STARC05-2.1.6.3

Expressions should not be used as array indexes. (Verilog)

Expression should not be used as array indices. (VHDL)

Description

Verilog

The STARC05-2.1.6.3 Tag flags expressions used as array indexes.

Array indexes should be simple signal names. Using expressions as array indexes can make the design complicated and may generate redundant logic during synthesis.

VHDL

The STARC05-2.1.6.3 Tag flags expressions used as array indices.

Array indices must be simple signal names or static values. Designs where expressions are used as array indices, may be too complicated. Also, some synthesis tools may generate redundant logic for such designs.

Message Details

Verilog

The following message appears at the location where an array index *<index-name>* is an expression:

Array index '*<index-name>*' is an expression

VHDL

The following message appears at the location where an expression *<expr>* is used as an array index:

Expression '*<expr>*' used as array index

Severity

Recommended

STARCO5-2.1.6.4

Index variable width should not be short (Verilog)

Description

The STARCO5-2.1.6.4 Tag flags variables used as index of a bit-select or an array where the width of the variable is less than the width of the bus or size of the array.

When a variable used to index a bit-select or array has a width less than the width of the bus or the size of the array, some elements of the bus or array will not be accessible.

Message Details

The following message appears at the location where an index variable `<index-name>` is used to index the variable or signal `<name>` and the width of the index variable is less than the width of the variable or signal: Variable/Signal '`<name>`' is indexed by '`<index-name>`' which is smaller than full range of this vector

Severity

Caution

Verilog Examples

Consider the following example where the width of index variable `i` is less than then the width of signals `out` and `in` that it is indexing:

```
module test(out, in, clk);
    input [3:0] in;
    input clk;
    output [3:0] out;

    reg [3:0] out;
    wire i;

    always @ (posedge clk)
        out[i] = in[i];
endmodule
```

For this example, SpyGlass generates the following messages:

variable/Signal 'in' is indexed by 'i' which is smaller than full range of this vector

variable/Signal 'out' is indexed by 'i' which is smaller than full range of this vector

STARC05-2.1.7.1

Direct assignment must be used for aggregates (VHDL)

Description

The STARC05-2.1.7.1 Tag flags aggregate specifications that do not use direct assignments.

Not assigning vector aggregates directly to the target may cause error by some synthesis and simulator tools.

Example,

```
SIG <=(IN1,others =>IN2);{Good}
```

```
SIG <=IN1 &(1 =>IN2, 2 => IN3);{Bad}
```

The STARC05-2.1.7.1 Tag ignores record aggregates.

Message Details

The following message appears at the location where direct assignment is not used for an aggregate specification:

Direct assignment not used for vector aggregates

Severity

Mandatory

STARC05-2.1.7.3

Bit position based and others based assignment should not be used together for aggregate specification (VHDL)

Description

The STARC05-2.1.7.3 Tag flags aggregate specifications that use both bit-position based assignment and others clause based assignment.

Mixing explicit bit position and others based assignment for aggregates may be interpreted as syntax error by some simulators. Instead, concatenation should be used.

For example:

```
library ieee;  
use ieee.std_logic_1164.all  
  
SIG <= (2 => IN1, others => '0');{Bad}  
SIG <= "00" & IN1;{Good}
```

Message Details

The following message appears at the location of a Tag-violating aggregate specification:

Bit position based and others based assignment mixed in
aggregate specification

Severity

Recommended

STARC05-2.1.8.1

Constrained arrays should not be used as sub-program arguments (VHDL)

Description

The STARC05-2.1.8.1 Tag flags sub-program arguments of constrained data type.

Using unconstrained arrays as sub-program argument, the array range of the signal, which calls function, is automatically applied.

Message Details

The following message appears at the location where range is specified for an argument *<argu-name>* of a sub-program *<subp-name>*:

Range specified for argument '*<argu-name>*' in *<subp-type>* '*<subp-name>*'

Where *<subp-type>* can be function or procedure.

Severity

Recommended

STARC05-2.1.8.10

No other statement should follow the return statement in function description (VHDL)

Description

The STARC05-2.1.8.10 Tag flags functions that have statements after the return statement in any of the branches.

As the statements that follow the last return value assignment statement are ignored, such statements can be confusing.

Message Details

The following message appears at the location where a statement is encountered after the return statement in a branch of function description *<func-name>*:

Statement exists after the return statement in function *<func-name>*

Severity

Mandatory

STARC05-2.1.8.2

Specify range for return values in function description when return type is array (VHDL)

Description

The STARC05-2.1.8.2 Tag flags return values of type array that do not have range specification in function description.

Returning unconstrained arrays may generate syntax errors by some simulators.

Message Details

The following message appears at the location where the return value *<name>* of a function *<func-name>* is of unconstrained array type:

Range not specified in return value '*<name>*' in '*<func-name>*'

Severity

Mandatory

STARC05-2.1.8.4

Only OTHERS clause must be used when specifying aggregate in sub-program description (VHDL)

Description

The STARC05-2.1.8.4 Tag flags sub-program aggregates specified without using the `others` clause.

Using other than `others` clause as aggregate in sub-program description may be flagged as a syntax error by some simulators and logic synthesis tools.

Message Details

The following message appears at the location where an aggregate is described in sub-program `<subp-name>` without the `others` clause:

other than OTHERS clause used as aggregate in <subp-type>
'<subp-name>'

Where `<subp-type>` can be function or procedure.

Severity

Mandatory

STARC05-2.1.8.5a

Do not use nested sub-program description in the design (VHDL)

Description

The STARC05-2.1.8.5a Tag flags nested sub-program descriptions.

Having sub-program description inside another sub-program increases the complexity of the design and makes it difficult to understand. Therefore, nesting of functions or procedures should be avoided in the design.

Message Details

The following message appears at the location where a sub-program *<subp2-name>* is described inside another sub-program *<subp1-name>*:

Nested sub-program '*<subp2-name>*' described in *<subp-type>* *<subp1-name>*

Where *<subp-type>* can be function or procedure.

Severity

Mandatory

STARC05-2.1.8.5b

Do not call a sub-program within a subprogram (VHDL)

Description

The STARC05-2.1.8.5b Tag flags sub-program calls in sub-program descriptions.

Calling a sub-program within a sub-program increases the complexity and makes it difficult to understand the design. Therefore, nesting of functions or procedure should be avoided in the design.

Message Details

The following message appears at the location where a sub-program `<subp2-name>` is called inside another sub-program `<subp1-name>`:

Nested sub-program call '`<subp2-name>`' in `<subp-type>` `<subp1-name>`

Where `<subp-type>` can be function or procedure.

Severity

Mandatory

STARC05-2.1.8.6

Global signals must not be read in a subprogram description (VHDL)

Description

The STARC05-2.1.8.6 Tag flags global signals read directly inside a subprogram description.

A variable that is outside the scope of a subprogram must not be read inside the subprogram construct as changes in the variable outside the subprogram may result in unknown function behavior.

NOTE: *The STARC05-2.1.8.6 Tag supports generate-for block.*

NOTE: *Global signal/variable is not allowed inside a pure function. SpyGlass produces a syntax error STX_472 in such a case.*

Message Details

The following message appears at the location where a global signal or variable `<name>` is read directly inside a subprogram `<subp-name>`:

Global <type> '`<name>`' read in <subp-type> <subp-name>

Where `<subp-type>` can be function or procedure.

Severity

Mandatory

STARC05-2.1.8.9

A function must have the return statement and return a valid value in all possible states of the function. (VHDL)

Description

The STARC05-2.1.8.9 Tag flags function descriptions where return statement and a valid return value are not described in all possible states.

Some simulation tools assume the function return value to be 0 for the undefined state whereas the synthesis tools assume the function return value to be x for the undefined state. Thus, there is a possibility of pre- and post-synthesis simulation mismatch of the design.

Therefore, a function must have a return statement for all possible states, and should return a valid value in all possible states.

Message Details

The following message appears at the location of a function `<func-name>` where the return statement and a valid return value are not described in all possible states:

Return value not described for all possible states in function `<func-name>`

Severity

Recommended1

STARC05-2.1.9.4

Do not use built-in attribute except range, length, left, right, high, low, reverse_range, event (VHDL)

Description

The STARC05-2.1.9.4 Tag flags unsupported built-in attributes.

While not all logic synthesis tools support complete set of built-in attributes, most synthesis tools support the following built-in attributes:

range	length	left	right	high
low	reverse_range	event		

Using the widely supported attributes ensures portability of design over different synthesis tools.

Message Details

The following message appears at the location where an unsupported built-in attribute *<name>* is used:

Built-in attribute '<name>' used

Severity

Mandatory

STARCO5-2.1.9.5

Do not use user-defined attributes in the design (VHDL)

Description

The STARCO5-2.1.9.5 Tag flags user-defined attributes.

Use of attributes other than the most generic ones may cause unexpected behavior since the level of support during synthesis and simulation is not assured.

Declaration of user-defined attributes is not a violating condition. Violation is reported on the usage or specification of the attributes.

NOTE: *If an attribute is used, the violation is reported at the place of its use and not where it is specified. However, for the unused attributes, the violation is reported at the place of specification.*

Message Details

The following message appears at the location where a user-defined attribute `<name>` is used:

User-defined attribute '`<name>`' used

Examples

Consider the following example:

```
entity ent is
  port(x,y:in bit;
        z:out bit
  );
end ent;
```

```
architecture arc of ent is
```

```
  signal s:bit;
  attribute Project_Name1: string;
  attribute Project_Name1 of s : signal is "STARCO-VHDL";

  attribute Project_Name2: string;
  attribute Project_Name2 of s : signal is "STARCO-VER";
```

```
--violation

begin
  process(x,y,s)
  begin
    if ( s'Project_Name1 = "STARCO-VHDL" ) then
      --violation
      z <= x;
    end if;

  end process;
end arc;
```

In the above example, the violations are reported as explained below.

- For Project_Name1, the following violation is reported at the place of use:

User-defined attribute 'Project_Name1' used

- For Project_Name2, the following violation is reported at the place of specification because it is unused:

User-defined attribute 'Project_Name2' used

Severity

Mandatory

STARC05-2.10.1.2

Parentheses should be used for logical operations to improve readability. (Verilog)
Parenthesis should be used in expressions with multiple operators (VHDL)

Description

Verilog

The requirement of this Tag is covered by the [STARC05-2.1.4.1](#) Tag.

VHDL

The STARC05-2.10.1.2 Tag flags complex expressions where different operators at same precedence level are used without parenthesis.

When operators of different precedence levels appear in same expression, then parenthesis are still required as the order of execution is may be ambiguous. Thus, the following example violates the STARC05-2.10.1.2 Tag:

```
return in1 + in2 * in3;
```

However, the expressions where operators of same precedence level are used, you should provide parenthesis to explicitly specify the order of valuation. Thus, the following example violates the STARC05-2.10.1.2 Tag:

```
return in1 + in2 - in3;
```

In complex expressions using multiple operators, it is recommended to use parenthesis to force the required order of execution. Use of parenthesis also increases the readability of expressions and reduces the possibility of errors.

Message Details

VHDL

The following message appears at the location where a complex expression using operators of same precedence level has no parenthesis:

Use parenthesis in complex expressions <opr>

Where <opr> is the first offending operator found.

Severity

Recommended (VHDL)

STARC05-2.10.1.7

Do not use RAM output signals for a conditional expression of if statements. (Verilog)

Description

The STARC05-2.10.1.7 Tag flags if constructs where the condition expression uses memory outputs.

Avoid using RAM output signals for a conditional expression of if constructs.

Message Details

The following message appears at the location of the condition expression of an if construct that uses a memory output signal:

Do not use RAM output signal for conditional expression of if statements

Severity

Recommended1

STARC05-2.10.1.8

Do not use RAM output signals for the selection expression of case statements that do assign 'x' in the default clause. (Verilog)

Description

The STARC05-2.10.1.8 Tag flags case constructs where the select expression uses memory outputs.

Avoid using RAM output signals for the select expression of case constructs.

Message Details

The following message appears at the location of the select expression of a case construct that uses a memory output signal:

Do not use RAM output signal for selection expression of case statements

Severity

Recommended1

STARC05-2.10.3.1

Bit-widths of operands of a relational operator should match. (Verilog)

Description

The *STARC05-2.10.3.1* Tag flags bit-width mismatches between LHS and RHS expressions of relational operations.

Bit-width mismatch between LHS and RHS of a relational operation may result in incorrect or confusing result.

NOTE: *If the count of operators is more than 500 in an expression, then the expression is ignored for Tag checking.*

By default, integer constants are assumed to have natural width ($\log_2(N)+1$). To consider LRM width (32 bits) for integer constants, set the [use_lrm_width](#) Tag parameter to *yes*.

By default, the *STARC05-2.10.3.1* Tag does not report violation when the right or left expression is a constant, including parameter, sized or unsized based number, and unsized integer. Setting the [check_static_value](#) parameter to *yes* changes this behavior, and setting the [strict](#) parameter in addition further alters the behavior.

The following table summarizes these variations in behavior:

Type of left or right expression	check_static_value set to no	check_staic_value set to yes	
		strict set to no	strict set to yes
Parameter	Does not report	Reports if the width of constant is larger	Reports any width mismatch
Sized based number (8'h15)	Does not report	Reports if the width of constant is larger	Reports any width mismatch
Unsized based number ('h15)	Does not report	Reports if the width of constant is larger	Reports any width mismatch
Unsized integer (18)	Does not report	Reports if the width of constant is larger	Reports any width mismatch

See the following example for details.

Also, when set to yes, the `check_static_value` parameter checks for width mismatch involving static expressions and non-static expressions that contain a static part. Refer to the [check_static_value](#) section for more details.

Consider the following example:

```
module top(input clk);
  reg [3:0] a, b, data;
  integer i1;
  int i2;
  parameter P1 = 119;
  parameter P2 = 2;
  always @(posedge clk)
  begin
    if (data == P1)           //violation with check_static_value
      a <= 17;
    else if(data == P2) //violation with check_static_value
    and strict
      b = 8'hbb;
    if (data == 10'd119) //violation with check_static_value
      a <= 17;
    else if(data == 2'b10) //violation with check_static_value
    and strict
      b = 8'hbb;
    if (data == 'd119)       //violation with
    check_static_value
      a <= 17;
    else if(data == 'b10) //violation with check_static_value
    and strict
      b = 8'hbb;
    if (data == 119) //violation with check_static_value
      a <= 17;
    else if(data == 2) //violation with check_static_value and
    strict
      b = 8'hbb;
  end
endmodule
```


In the above example, violations are reported for different cases (mentioned above in red) when the [check_static_value](#) and/or [strict](#) parameters are set.

By default, the STARC05-2.10.3.1 Tag reports violation for disabled code in loops and conditional (if condition, ternary operator) statements. Set the value of the [disable_rtl_deadcode](#) parameter to yes to disable violations for disabled code in loops and conditional (if condition, ternary operator) statements.

By default, the 2.10.3.1 Tag does not check the bit-width as per LRM. Set the [nocheckoverflow](#) Tag parameter to yes to check the bit-width as per LRM. However, for constants, natural width is considered.

The following examples show the bit-width calculation for different values of the [nocheckoverflow](#) Tag parameter:

■ Example1:

```
input [3:0] a;
input [4:0] b;
if (a == a*b)
```

Here, if you set the value of the [nocheckoverflow](#) Tag parameter to no, LHS is assigned 4 bits and RHS is assigned 9 bits.

If you set the value of the [nocheckoverflow](#) Tag parameter to yes, LHS is assigned 4 bits and RHS is assigned 5 bits.

■ Example 2:

```
input [3:0] a;
input [4:0] b;
if (a == a+b)
```

Here, if you set the value of the [nocheckoverflow](#) Tag parameter to no, LHS is assigned 4 bits and RHS is assigned 6 bits.

If you set the value of the [nocheckoverflow](#) Tag parameter to yes, LHS is assigned 4 bits and RHS is assigned 5 bits.

Message Details

The following message appears at the location where the bit-width *<op1-width>* of the first operand expression *<op1-exp>* is different from the bit-width of the second operand expression *<op2-exp>* for the relational

```
operator <op-name>:  
operand bit-width mismatch for operator '<op-name>':  
'<op1-exp>'('<op1-width>' bits) and '<op2-exp>'('<op2-width>' bits) [Hierarchy: '<hier-path>']
```

Severity

Recommended2

STARC05-2.10.3.2a

Bit-width of operands of a logical operator must match. (Verilog)

Description

The STARC05-2.10.3.2a Tag flags bit-width mismatches between operand expressions of logical operations.

Bit-width mismatch between operands of a logical operation may lead to incorrect results.

The STARC05-2.10.3.2a Tag does not flag a relational operation if an integer constant is used as one of the operands and the width of that constant is less than or equal to the width of the other operand.

NOTE: *If the count of operators is more than 500 in an expression, then the expression is ignored for Tag checking.*

By default, integer constants are assumed to have natural width ($\log_2(N)+1$). To consider LRM width (32 bits) for integer constants, set the [use_lrm_width](#) Tag parameter to `yes`.

The STARC05-2.10.3.2a Tag does not report violation for expressions where the width of the constant expression, constant integer, or base number is less than the width of the other operand. For example, consider the following expressions:

```
assign out = base [4:0] && (1023>>6);
assign out1 = base1 [4:0] || 10;
assign out2 = base2 [4:0] || 1023;
```

You can set the value of the [nocheckoverflow](#) parameter to `yes` to calculate the width according to the LRM, as shown in the following table:

Operator	LRM width	Normal width
<code>+, -</code>	Max (LHS width, RHS width)	Max (LHS width, RHS width) + 1
<code>*</code>	Max (LHS width, RHS width)	LHS width + RHS width
<code>/</code>	Max (LHS width, RHS width)	LHS width
<code>%</code>	Max (LHS width, RHS width)	RHS width

Also, consider the following example:

```
(in1[3:2] + in1[1]) && in2[3:2]
```

In the above expression, the STARCO5-2.10.3.2a Tag flags a violation, if the value of the [nocheckoverflow](#) parameter is set to `no`. The Tag does not flag a violation, if the value of the `nocheckoverflow` parameter is set to `yes`.

By default, the STARCO5-2.10.3.2b Tag does not report violation for the static expressions and non-static expressions containing static expressions. Set the value of the [check_static_value](#) parameter to `yes` or `<Tag_list>` to report violation for all cases with width mismatch, involving static expressions and non-static expressions having a static part. Other possible values are `only_const` and `only_expr`.

The default value of the [enable_rtl_deadcode](#) parameter is `no`. Set this parameter to `yes` to report different violations for dead codes in the RTL resulting from `if/for/while` and ternary operators that are ignored when the [disable_rtl_deadcode](#) parameter is enabled.

Message Details

The following message appears at the location where a bit-width mismatch between operands of a logical operation is encountered:

operand bit-width mismatch for operator '<op-name>':
'<op1-exp>'('<op1-width>' bits) and '<op2-exp>'('<op2-width>' bits) [Hierarchy: '<hier-path>']

Severity

Mandatory

STARCO5-2.10.3.2c

Unequal length operands in bit/arithmetic operator (Verilog)

Description

The STARCO5-2.10.3.2c Tag flags bit-width mismatches between values being combined in a bit or arithmetic operation.

Values of different width are being combined in a bit or arithmetic operation. This may be intended behavior, but since it is also an easy opportunity to make mistakes, all such cases should be examined closely. For example, addition of two words of unequal width may signal that you forgot to update the width of one of the buses.

NOTE: *If the count of operators is more than 500 in an expression, then the expression is ignored for Tag checking.*

The STARCO5-2.10.3.2c Tag ignores arithmetic operations where:

- Both operands are constant integer or integer variable expressions.
- One of the operands is an integer constant and the width of that constant is less than or equal to the width of the other operand.

By default, integer constants are assumed to have natural width ($\log_2(N)+1$). To consider LRM width (32 bits) for integer constants, set the [use_lrm_width](#) Tag parameter to `yes`.

By default, the STARCO5-2.10.3.2c Tag reports violation for disabled code in loops and conditional (if condition, ternary operator) statements. Set the value of the [disable_rtl_deadcode](#) parameter to `yes` to disable violations for disabled code in loops and conditional (if condition, ternary operator) statements.

The STARCO5-2.10.3.2c Tag does not report violation for expressions where the width of the constant expression, constant integer, or base number is less than the width of the other operand. For example, consider the following expressions:

```
assign out = base [4:0] + (1023>>6);
assign out1 = base1 [4:0] + 10;
assign out2 = base2 [4:0] + 1023;
```

You can set the value of the [nocheckoverflow](#) parameter to `yes` to calculate the width according to the LRM, as shown in the following table:

Operator	LRM width	Normal width
+, -	Max (LHS width, RHS width)	Max (LHS width, RHS width) + 1
*	Max (LHS width, RHS width)	LHS width + RHS width
/	Max (LHS width, RHS width)	LHS width
%	Max (LHS width, RHS width)	RHS width

Also, consider the following example:

```
(in1[3:2] + in1[1]) + in2[3:2]
```

In the above expression, the STARC05-2.10.3.2c Tag flags a violation, if the value of the *nocheckoverflow* parameter is set to no. The Tag does not flag a violation, if the value of the *nocheckoverflow* parameter is set to yes.

Message Details

The following message appears at the location where a bit-width mismatch between LHS *<exprl>* (*<LHS-bit-width>* bits) and RHS *<exprr>* (*<RHS-bit-width>* bits) of an operation of operator *<operator>* is encountered:

For operator (*<operator>*), left expression: "*<exprl>*" width *<LHS-bit-width>* should match right expression: "*<exprr>*" width *<RHS-bit-width>* [Hierarchy: '*<hier-path>*']

Severity

Mandatory

STARC05-2.10.3.5

Base should be specified for constant value used. (Verilog)

Description

The *STARC05-2.10.3.5* Tag reports constants declared without the base type specification.

Constants declared without base type specifications can be confusing to other users as such constants are assumed to be decimal. However, the designer may have intended to specify an octal or a hexadecimal number. Hence, it is mandatory to explicitly specify the base type ('b', 'o', 'd', or 'h').

The Tag does not report violations for the following:

- RHS operand of shift operators, such as <<, >>, <<<, >>>
- Expressions where one operand is of integer type and other operand is not of register or net type
- Constants with value 0 or 1
- Constants used with integer variables

By default, the Tag does not report violations for constants used with integer variables. Set the *strict* parameter to *yes* to report violations in such cases.

Message Details

The following message appears at the location where a constant value *<const-value>* is declared without the base type specification:

Missing base type specification for constant value(s) '[array separated constant list]' in expression '[expression]

Severity

Recommended2

Verilog Example

The following example illustrates the behavior of the *STARC05-2.10.3.5* Tag:

```

module test;
  function f;
  integer i, j, tmp;
  reg w1, w2, w3;
  begin
    tmp = 59;           //violation by default
    tmp = 1;           //no violation
    tmp = 42*i;         //violation when strict is set
    tmp = i+55;         //violation when strict is set
    tmp = 2*i + 1;      //violation when strict is set
    tmp = 32'd2*i + 1;  //no violation
    tmp = 2*i + 45 * (3 + i*2) + 86 + 32*j;
                        //violation when strict is set
    w2 = w1*65;         //violation by default
    w2 = w1*65 + i*23;  //violation by default
    w2 = i*w1 + 44;     //violation by default
    w2 = w1 + 44;       //violation by default
    w3 = (w2>>20) + w3*65; //violation by default
    w3 = (w2*32 + i*42 + w3*65)<<20;
                        //violation by default
    w3 = (j*32 + i*42 + 65)<<20;
                        //violation when strict is set
    w3 = (j*32 + i*42>>30 + 65) + i*20;
                        //violation when strict is set
    w3 = (j*32 + (i*42<<30) + w1) + i*20;
                        //violation when strict is set
    w3 = (j*32 + (i*42<<30) + w1) + 20;
                        //violation by default
    w3 = (w2*32 + (i*42>>30) + j) + 20;
                        //violation by default
    w3 = 45*w2 << 20;  //violation by default
    w3 = 0*w2 << 20;   //no violation
  end
endfunction
endmodule

```


STARC05-2.10.3.6

Bit-width should be specified for based constants of width more than 4 bits. (Verilog)

Description

The STARC05-2.10.3.6 Tag flags constant declarations of based number constants of bit-width more than the specified number that do not have bit-width specification.

The STARC05-2.10.3.6 Tag flags the following example as highlighted:

```
...
wire [5:0] w1;
assign w1 = 'b101010;
assign w1= 6'b101010;
...
```

By default, the STARC05-2.10.3.6 Tag flags constant declarations of based number constants of bit-width more than 4 bits. Use the [bitwidth_basedconst](#) Tag parameter to specify a different number.

Bit-width of any based number constant that has a bit-width greater than 4 must be specified explicitly to make it more readable and avoid errors.

By default, the [ignore_vector_bit_selection](#) parameter is set to no and the Tag checks bit selects, part selects, array bit selects and array part selects. Set the value of this parameter to yes to ignore for such cases.

Set the [strict](#) parameter to yes to ignore violations for the based constants inside concatenations. By default, the strict parameter is set to no and the Tag reports violations for such cases. For example:

```
wire [31:0] infra = {{{'d32 - MEM_PDN_BIT0}{1'b0}}},
infra_mem} ;    // No violation for multiplier

wire [31:0] infra1 = {{{6'd32 - MEM_PDN_BIT0}{'b0}}},
infra_mem} ;    // No violation for concatenation
```

Message Details

The following message appears at the location where a based number constant `<const-name>` is declared without the bit-width specification:

Missing base width specification in '<const-name>'

Severity

Mandatory

STARC05-2.10.4.3

`Std_logic_vector(conv_unsigned(&, &))` should be used to convert integer to `std_logic_vector` (VHDL)

Description

The STARC05-2.10.4.3 Tag flags integer to `std_logic_vector` conversions that do not use the `conv_unsigned` method of `std_logic_vector` package.

For conversion of integers to `std_logic_vector`, most simulators and synthesis tools treat the MSB as the signed bit until it is once converted to unsigned. Therefore, while assigning integer value to objects of type `std_logic_vector`, it is recommended to use the conversion function `std_logic_vector(conv_unsigned(&, &))`.

Message Details

The following message appears at the location where an integer is assigned the value of an object of type `std_logic_vector` without using the `conv_unsigned` method:

```
std_logic_vector(conv_unsigned(&,&)) not used to convert  
integer to std_logic_vector
```

Severity

Recommended1

STARC05-2.10.4.7

Qualified expressions should not be used in the design. (VHDL)

Description

The STARC05-2.10.4.7 Tag flags qualified expressions used in the design. In VHDL, data type is judged by the context of description. Consider the following example:

```
signal A : std_logic_vector(7 downto 0);  
A <= "101111001";  
...
```

In the current context, "10111001" is not a string nor is a bit_vector data type object; it is an std_logic_vector data object.

Normally, the above interpretation is possible. However, in some cases, such judgement is not possible. Consider the following example:

```
case (A & B & C) is  
  when "000" => Y <= "1011";  
  ...
```

In the above case, the A&B&C data type is not settled. Hence, an error occurs. In such cases, data type can be settled by using the qualified expressions as in the following example:

```
subtype std3bit is std_logic_vector(2 downto 0);  
case std3bit'(A & B & C) is  
  ...
```

Data type is settled by data type '().

However, the qualified expressions are verbose. Therefore, descriptions that do not require qualified expressions are preferable.

Message Details

The following message appears at the location of a qualified expression: qualified expression used in the design

STARCO5 Tags

Severity

Recommended

STARCO5-2.10.4.8

Function To_stdlogicvector should not be used in the design. (VHDL)

Description

The STARCO5-2.10.4.8 Tag flags To_stdlogicvector function calls.

In VHDL' 87, when assigning hexadecimal values to objects of std_logic_vector data type, the following conversion method was taken because notation system X "" was supported only by the bit_vector data type:

```
signal A : std_logic_vector(7 downto 0);  
A <= To_StdLogicVector(X"3F");
```

With VHDL' 93, notation system X"" is supported by the std_logic_vector data type. also. Therefore, the above description example generates syntax error because it cannot be judged whether the object X"3F" is of the std_logic_vector data type or of the bit_vector data type; it can be std_logic_vector (X"3F") but can also be as simple as A<=X"3F".

Message Details

The following message appears at the location of a To_stdlogicvector function call:

Function 'To_stdlogicvector' used in the design

Severity

Recommended

STARC05-2.10.5.1

Arithmetic/relational operation with operand bit-width more than 8 in multiple conditional expressions is same. (Verilog)

Description

The STARC05-2.10.5.1 Tag flags two or more condition expressions in the same module when:

- More than two same arithmetic or relational operations exist in conditional expressions in the same module
- Bit-width of any of the operands used with the arithmetic or relational operator is more than the specified number.

By default, the STARC05-2.10.5.1 Tag flags operations where the operand bit-width is more than 8 bits. Use the [arithrel_op_maxsize](#) Tag parameter to specify a different number.

The STARC05-2.10.5.1 Tag flags checks for operations involving the following operators:

Arithmetic operators	+	-	*					
Relational operators	<	>	<=	>=	==	!=	===	!==

Conditional expressions with common logic generate duplicate circuitry that can be minimized by resource sharing of these operators.

Using arithmetic or relational operators with large bit-width operands generate large and complex circuits.

Message Details

The following message appears at the location where an arithmetic or relational sub-expression `<expr>` is found in a conditional expression when the same sub-expression was earlier found in a condition expression at line `<num>` in the same module `<module-name>` and the bit-width of one or more operands in more than number specified by the `arithrel_op_maxsize` Tag parameter:

Logic '`<expr>`' with operand bit-width greater than `$arithrel_op_maxsize` is common with conditional expression (line: `<num>`). [Hierarchy: '`<module-name>`']

Severity

Reference

Verilog Examples

Consider the following example where logically-equivalent sub-expressions (IN1 < IN2 and IN1 > IN2) exist in two conditional expressions in the same module:

```
module test(IN1, IN2, OUT1, OUT2);
    input  [16:0] IN1;
    input  [16:0] IN2;
    output [16:0] OUT1;
    output [16:0] OUT2;

    reg    [16:0] OUT1;
    reg    [16:0] OUT2;

    always @(IN1 or IN2)
        if(IN1 < IN2)
            OUT1 = IN1 * 2;
        else
            OUT1 = IN1 / 2;

    always @(IN1 or IN2)
        if(IN2 > IN1)
            OUT2 = IN2 * 4;
        else
            OUT2 = IN2 / 4;
endmodule
```

If the above example is analyzed with the default setting (8) of the `arithrel_op_maxsize` Tag parameter, SpyGlass generates the following message:

Logic '(IN2>IN1)' with operand bit-width greater than 8 is common with conditional expression (line: 10) [Hierarchy: ':test']

STARC05-2.10.5.2

Do not use any arithmetic operator twice in the same "if" statement.

Description

The STARC05-2.10.5.2 Tag flags multiple use of the same operator in an if statement.

Multiple use of the same operator is a candidate for resource sharing.

The STARC05-2.10.5.2 Tag reports violation when the assignment statements in all different branches of if-else construct have common LHS and the RHS' share an arithmetic operator (+, -, *, and /) with one common operand for that operator.

One violation is reported for every such pair of those common operator and operand; only condition being that they are used exactly twice.

Message Details

The following message appears when an operator `<opr>` at line `<num>` is used twice in the same if statement:

Two '`<opr>`' (line: `<num>`) is a candidate for resource sharing

Severity

Reference

STARC05-2.10.5.3

Do not use any arithmetic operator thrice or more in the same "if" statement.

Description

The STARC05-2.10.5.3 Tag flags arithmetic operators used thrice or more in the same if statement.

Multiple use of the same operator is a candidate for resource sharing.

The STARC05-2.10.5.3 Tag reports violation when assignment statements in all different branches of if-else construct have common LHS and the RHS' share an arithmetic operator (+, -, *, and /) with one common operand for that operator.

One violation is reported for every such pair of those common operator and operand; only condition being that they are used thrice or more.

Message Details

The following message appears when an arithmetic operator *<opr>* at line *<num>* is used thrice or more in an if statement:

Three or more '*<opr>*' (line: *<num>*) is a candidate for resource sharing

Severity

Recommended1

STARC05-2.10.5.3a

Reports violation when there are two arithmetic operations in the Always block

When to Use

Use this Tag to detect Always block that contain two arithmetic operations

Description

The STARC05-2.10.5.3a Tag reports violation for always block that contain two arithmetic operations.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Arithmetic operations (actual count <count>) is equal to 2 in module <module-name> [Hierarchy:<hierpath>]

Potential Issues

The violation message is reported when the Tag detects two arithmetic operations in the Always block.

Consequences of Not Fixing

Not fixing the violation may result in timing issues in the design.

How to Debug and Fix

Click the violation message to highlight the violating Always block in the HDL viewer.

To fix the violation, ensure that the Always block does not contain two arithmetic operations.

Example Code and/or Schematic

Example 1

```
module test();  
reg SIG1, SIG2, SIG3, SIG4;  
reg OUT1;  
always @(SIG1 or SIG2 or SIG3 or SIG4) // violation,  
because "+" operators are 2.  
begin  
OUT1 = SIG1 + SIG2 + SIG3 ;  
end  
endmodule
```

Example 2

```
module test();  
reg SIG1, SIG2, SIG3, SIG4;  
reg OUT1;  
always @(SIG1 or SIG2 or SIG3 or SIG4) // violation,  
because "+" and "-" operators are 2.  
begin  
OUT1 = SIG1 + SIG2 - SIG3 ;  
end  
endmodule
```

Example 3

```
module test();  
reg SIG1, SIG2, SIG3, SIG4;  
reg OUT1;  
always @(SIG1 or SIG2 or SIG3 or SIG4) // violation,  
because "+" operators are 2.  
begin  
if(SIG1)  
OUT1 = SIG1 + SIG2 ;  
else  
OUT1 = SIG1 - SIG3 ;  
end
```

```
endmodule
```

Example 4

```
module test();
reg SIG1, SIG2, SIG3, SIG4;
reg OUT1;
always @(SIG1 or SIG2 or SIG3 or SIG4) // no violation, because "+,*,/"
operators are 3, not 2.
begin
OUT1 = SIG1 + SIG2 * SIG3 / SIG4 ;end
endmodule
```

Example 5

```
module test();
reg SIG1, SIG2, SIG3, SIG4;
reg OUT1;
always @(SIG1 or SIG2 or SIG3 or SIG4) //no violation, because "+"
operators are 1, which is less than 2.
begin
OUT1 = SIG1 + SIG2;end
endmodule
```

Example 6

```
module test();
reg SIG1, SIG2, SIG3, SIG4;
reg OUT1, OUT2;
always @(SIG1 or SIG2 or SIG3 or SIG4) // violation,
because "+" operators are 3, which is more than 2.
always @(SIG1 or SIG2 or SIG3 or SIG4) // violation,
because "+" operators are 2.
begin
OUT1 = SIG1 + SIG2;
OUT2 = SIG3 + SIG4 ;
end
```

endmodule

Default Severity Label

Info

Tag Group

Creating Functional Libraries

Reports and Related Files

No related report or files

STARC05-2.10.5.3b

Reports violation when there are three or more arithmetic operations in Always block

When to Use

Use this Tag to check if the Always block contains three or more arithmetic operations.

Description

The STARC05-2.10.5.3b Tag reports violation for always block that contains three or more arithmetic operations.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Arithmetic operations (actual count <count>) is more than or equal to 3 in module <module-name>
[Hierarchy:<hier_path>]

Potential Issues

The violation message is reported when the Tag detects three or more arithmetic operations in the Always block.

Consequences of Not Fixing

Not fixing the violation may result in timing issues in the design.

How to Debug and Fix

Click the violation message to highlight the violating Always block in the HDL viewer.

To fix the violation, ensure that the Always block does not contain three or more arithmetic operations.

Example Code and/or Schematic

Example 1

```
module test();
reg SIG1, SIG2, SIG3, SIG4;
reg OUT1;
always @(SIG1 or SIG2 or SIG3 or SIG4) // violation,
because "+" operators are 3.
begin
OUT1 = SIG1 + SIG2 + SIG3 + SIG4 ;
end
endmodule
```

Example 2

```
module test();
reg SIG1, SIG2, SIG3, SIG4;
reg OUT1;
always @(SIG1 or SIG2 or SIG3 or SIG4) // violation,
because "+", "-", and "*" operators are 3.
begin
OUT1 = SIG1 + SIG2 - SIG3 * SIG4;
end
endmodule
```

Example 3

```
module test();
reg SIG1, SIG2, SIG3, SIG4;
reg OUT1;
always @(SIG1 or SIG2 or SIG3 or SIG4) // violation,
because "+", "-", and "*" operators are 4.
begin
if(SIG1)
OUT1 = SIG1 + SIG2 * SIG4 ;
else
OUT1 = SIG1 - SIG3 * SIG4;
end
```


STARCO5 Tags

```
endmodule
```

Example 4

```
module test();  
reg SIG1, SIG2, SIG3, SIG4;  
reg OUT1;  
always @(SIG1 or SIG2 or SIG3 or SIG4) // no violation, because "+"  
operators are 2, which is less than 3.  
begin  
OUT1 = SIG1 + SIG2 * SIG3 ;  
end  
endmodule
```

Example 5

```
module test();  
reg SIG1, SIG2, SIG3, SIG4;  
reg OUT1, OUT2, OUT3;  
always @(SIG1 or SIG2 or SIG3 or SIG4) // violation,  
because "+" operators are 3.  
begin  
OUT1 = SIG1 + SIG2;  
OUT2 = SIG3 + SIG4 ;  
OUT3 = SIG2 + SIG4;  
end  
endmodule
```

Default Severity Label

Info

Tag Group

Creating Functional Libraries

Reports and Related Files

No related report or files

STARC05-2.10.5.5

Do not describe arithmetic operations with conditional operators (?) in assign statement. (Verilog)

Description

The STARC05-2.10.5.5 Tag flags conditional assignment statements where arithmetic operations are described.

No resource sharing of arithmetic operation would be performed in such cases. Instead, you should use if-else description, if possible.

In the following example, addition operations are used in the conditional assignment statement:

```
assign B = (A==1'b1) ? C + D: C + E;
```

The above construct can be remodeled as follows:

```
if (A==1'b1)
    B = C + D;
else
    B = C + E;
```

Message Details

The following message appears at the location where an arithmetic expression is found in a conditional assignment statement:

Arithmetic operation described with ternary operator

Severity

Recommended

STARC05-2.10.6.1

Possible loss of carry or borrow in addition or subtraction (Verilog)

Description

The *STARC05-2.10.6.1* Tag reports assignment statements with an addition or subtraction operation where the LHS bit-width is same as the RHS bit-width.

By default, the Tag does not report the following or similar statements:

Statement	Condition
<code>a = a + 1;</code>	Same variable used in both LHS and RHS
<code>a = b + 1;</code>	Bit-width of variable <code>b</code> is same or is less than the bit-width of variable <code>a</code>

To report such cases also, set the [strict](#) parameter to `yes` or `STARC05-2.10.6.1`.

By default, the Tag reports a violation for disabled code in loops and conditional (if condition, ternary operator) statements. Set the [disable_rtl_deadcode](#) parameter to `yes` to disable violations for disabled code in loops and conditional statements.

By default, the Tag does not report a violation for the following cases:

- Where same variable is assigned to itself:

```
reg [2:0] a;
a = a + 1;
```

- Where bit-width of RHS is less than or equal to the width of the LHS:

```
reg [2:0] a;
reg [2:0] b;
reg [1:0] c;
a = b + 1;
a = c + 1;
```

Set the [strict](#) parameter to report a violation for such cases.

By default, Tag checking is disabled for RHS expressions that are static or for RHS expressions that are non-static but contain a static part.

Consider the following example:

```
wire [2:0] a, b, c;
assign a = 3'b101 + 3'b110; //RHS is static
assign a = b + c + 1; //RHS is non-static with static part
assign a = b + 5; //RHS is non-static with static part
```

To report violations in the above example, use the [check_static_value](#) parameter as follows:

- Set this parameter to `only_const` to enable Tag checking for RHS expression, which is static.
- Set this parameter to `only_expr` to enable Tag checking for RHS expression, which is a non-static expression that contains a static part.
- Set this parameter to `yes` to enable Tag checking for static expression and non-static expression that contains a static part.

NOTE: Cases of the type, $a = b + c + 1$, are considered as complex counters. Set either the [strict](#) or [check_static_value](#) parameter to report violations for such cases.

By default, violations are not reported for the counter type of cases. Set the [strict](#) parameter to report violations for such cases. Consider the following example:

```
wire [2:0] a, b, c;
assign b = c + 1; //Will flag with strict set to yes
assign a = b + c + 5; //Not a counter, will not be flagged
//with strict set to yes
```

For static expressions, the Tag reports a violation if carry or borrow bit is lost. For example:

```
wire [1:0] out;
assign out = 2'b10 + 1'b1;
assign out = 2'b10 + 2'b11;
```

The same logic is also applicable for static expressions inside concatenation. For example:

```
wire [2:0] out1;
wire c;
assign out1 = {2'b10+2'b01,c};
assign out1 = {2'b10+2'b10,c};
```

NOTE: The STARCO5-2.10.6.1 Tag reports assignment statements where the bit-width of

RHS performing addition or subtraction is same as the bit-width of LHS. If there is a mismatch in the size of assignment statements in RHS and LHS, the [STARCO5-2.10.3.2b](#) Tag reports a violation.

When the result of an addition or subtraction operation is assigned to a bus of the same or lesser bit-width, the carry or borrow bit may be lost.

When a static value is added or subtracted from a non-static value, SpyGlass adds the maximum non-static value to the static value, and based on this value, SpyGlass calculates the width of an expression.

For example, consider the following code snippet:

```
wire [11:0] b;
wire [3:0] c;
  assign b = 12'hFF0 + c;
  assign b = 12'hFF1 + c;
```

In the above example, the maximum value of the first expression is 4095 and there is no overflow. Therefore, the Tag does not report a violation for the first expression. In addition, the maximum value of the second expression is 4096 and there is an overflow. Therefore, the Tag reports a violation for the second expression.

By default, the Tag does not report any violation inside concat operator if MSB is added to carry bit. For example, no violation is reported in the following case:

```
wire [2:0] b,c;
wire [3:0] result;
wire [5:0] out

  assign result = {b[0],b} + {c[0],c};
  // LSB added to carry
  assign result = {b[2],b} + {c[2],c};
  // MSB added to carry
  assign out = {{2{b[2]}},b[2],b} + {{2{c[2]}},c[2],c};
  // MSB added to carry
  assign out = {{3{b[2]}},b[2],b[2:1]} + {{3{c[2]}},c[2],c[2:1]};
  // MSB added to carry
```

Set the [strict](#) parameter to report such cases.

Consider a case in which RHS contains an addition operator with at least

one concatenation expression. The Tag does not report a violation if the number of continued zeros at the end of concatenation expression is greater than or equal to the width of other operand. For example, no violation is reported in the following case:

```
wire [6:0] a;
wire [3:0] b;
wire [2:0] c;
assign a = {b, 3'b0} + c;
assign a = {3'b111} + {b, 3'b0};
```

By default, if the RHS expression is a binary expression and each term is a concatenation expression padded with zero bits, the Tag checks for overflow while considering zero padded bits. That is, violation is not reported if the actual width of the expression can be accommodated in zero padded bits.

When the [handle_large_expr](#) parameter is set to *yes*, the Tag limits this checking for expressions containing up to 25 terms and for larger expressions, the LRM width is considered. For example:

```
module top();
wire [5:0] al;
reg a,b,c,d,e,f,g,h,i,j;
assign al[4:0] = {4'b0,a} + ({4'b0, b} + ({4'b0, c} +
({4'b0, d} + ({4'b0, e} + ({4'b0, f} + ({4'b0, g} + {4'b0,
h})))))) ; //8 terms, no violation
assign al[4:0] = {4'b0,a} + ({4'b0, b} + ({4'b0, c} +
({4'b0, d} + ({4'b0, e} + ({4'b0, f} + ({4'b0, g} + {4'b0,
h})))))) + {4'b0,a} + ({4'b0, b} + ({4'b0, c} + ({4'b0, d} +
({4'b0, e} + ({4'b0, f} + ({4'b0, g} + {4'b0, h})))))) +
{4'b0,a} + ({4'b0, b} + ({4'b0, c} + ({4'b0, d} + ({4'b0, e}
+ ({4'b0, f} + ({4'b0, g} + {4'b0, h})))))) + {4'b0, b};
// 25 terms, no violation
assign al[4:0] = {4'b0,a} + ({4'b0, b} + ({4'b0, c} +
({4'b0, d} + ({4'b0, e} + ({4'b0, f} + ({4'b0, g} + {4'b0,
h})))))) + {4'b0,a} + ({4'b0, b} + ({4'b0, c} + ({4'b0, d}
+ ({4'b0, e} + ({4'b0, f} + ({4'b0, g} + {4'b0, h})))))) +
{4'b0,a} + ({4'b0, b} + ({4'b0, c} + ({4'b0, d} + ({4'b0,
e} + ({4'b0, f} + ({4'b0, g} + {4'b0, h})))))) + {4'b0, b}
```

```
+ {4'b0, b};          //26 terms, no violation by default,
violation is reported when parameter handle_large_expr is
set

endmodule
```

In the above example, by default the Tag will not report violation for any of the assignment.

When the [handle_large_expr](#) parameter is set then the Tag will report violation for the last statement as LRM width is considered for this because it has more than 25 terms.

By default, the [control_sig_detection_nf](#) parameter is set to no. If you set this parameter to yes, the Tag treats a signal to be a control signal only if it is used in a conditional operator or control binary operator within the scope of currently processing statement. The Tag does not consider the signal to be a control signal even if it is used as a control signal in other places of the module (that is, the statements other than the current statement).

Message Details

The following message appears at the location of an assignment statement with an addition or subtraction operation in the design unit

<module-name> where the LHS bit-width *<widthl>* is same as the RHS bit-width *<widthr>*:

Possible assignment overflow: lhs width *<widthl>* (Expr: '*<lexpr>*') should be greater rhs width *<widthr>* (Expr: '*<rexpr>*') to accommodate carry/borrow bit [Hierarchy: '*<hier-path>*']

Where, *<hier-path>* is the complete hierarchical path.

How to Debug

Double-click the violation message. The HDL Viewer window highlights the line where addition or subtraction can cause overflow.

Severity

Caution

Examples

Consider the following example:

```
module test5(set, reset,q);
input set, reset;
output q;

reg q;
reg [7:0]a;

always @(set or reset)
begin
    if (set == 1'b1 )
    begin
        a <= a + q;

    end
    else
    begin
        a <= a - q;

    end
end

endmodule
```

In the above example, the *STARCO5-2.10.6.1* Tag reports the following violation messages:

Possible assignment overflow: lhs width 8 (Expr: 'a') should be greater than rhs width 8 (Expr:'(a + q)') to accommodate carry/borrow bit, [Hierarchy: ':test5']

Possible assignment overflow: lhs width 8 (Expr: 'a') should be greater than rhs width 8 (Expr:'(a - q)') to accommodate carry/borrow bit, [Hierarchy: ':test5']

STARC05-2.10.6.5

Large multipliers must not be described using the multiplication operator.

Description

The *STARC05-2.10.6.5* Tag reports large multipliers that are described by using the multiplication operator.

It is recommended to describe large multipliers (with results of 16 bits or wider) by using logical operations or gate-level circuits obtained from standard libraries.

By default, the Tag reports multipliers with an operand of bit-width more than 16 bits. Use the [starc2005_size_of_mult_max](#) parameter to specify a different bit-width.

By default, the Tag considers natural width of integer constants and unsized based numbers, which is $\log_2(N) + 1$. Set the [use_lrm_width](#) parameter to `yes` to consider the LRM width, which is 32 bits.

Synthesis tools may not be able to generate efficient circuits of large multipliers that are described by using the multiplication operator.

NOTE: *The Tag supports the generate-for block.*

The Tag ignores the following cases in Verilog and VHDL, where:

Verilog

- One of the operands is a loop variable.
- One of the operands is a constant power of 2.
- Both the operands are constants.
- One of the operands is 0 or 1.

VHDL

- Both operands of multiplication operator are constants.
- Multipliers in function and procedure body as size of the arguments may depend on the actual values passed to the function or procedure call.
- One of the operands is a constant power of 2.
- One of the operands is 0 or 1.

In Verilog and VHDL, by default, the Tag does not report violations for large multipliers used as array indexes. Set the *strict* parameter to *yes* to report violations for such cases.

Message Details

Verilog

The following message appears at the location of the expression *<expr>* that uses the multiplication operator and one operand of the multiplier has bit-width more than the bit-width limit (*<num>* bits) specified by the *starc2005_size_of_mult_max* parameter:

Large multiple (wider than *<num>* bits) described using the multiplication operator (*<expr>*) [Hierarchy: '*<hier-path>*']

In the violation message, *<hier-path>* is the hierarchical name of the instance (in *<module-name>: <inst-name>@<master-module-name>* format) where the Tag-violating construct is encountered.

VHDL

The following message appears at the location where a large multiplier is described in expression *<expr>*:

Large multiplier (wider than \$starc2005_size_of_mult_mux) described using the multiplication operator in expression '*<expr>*' [Hierarchy: ':<entity-name>(<arch-name>):']

Severity

Recommended1

Examples

Example 1

Consider the following example:

```
wire a = b[3:0] * 5;
wire a = b[3:0] * 'b101;
wire a = b[3:0] * (2 + 3);
```

In the above example, the *STARC05-2.10.6.5* Tag, by default, does not report any violation and considers the natural width of the constants for width calculation.

If you set the *use_lrm_width* parameter to yes, the Tag reports violations for first two cases. However, the Tag does not report violations for the third case because it consists of an expression and only natural width is considered for width calculation.

Example 2

The following example illustrates the behavior of the *STARCO5-2.10.6.5* Tag (VHDL):

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity sample2 is
  port(
    sel,i_data  : in  std_logic_vector(15  downto 0);
    o_data      : out std_logic_vector(7  downto 0);
    a,b         : in  integer ;
    c           : out integer
  );
end sample2;

architecture rtl of sample2 is
  component LOW is
    generic ( N : natural:=7 );
    port ( AA : in std_logic_vector( N downto 0));
  end component;
begin
  c <= b * a;    //violation by default
  inst : LOW port map ( AA => sel(a*b downto 8));
                //violation when strict set
  o_data(6) <= i_data(a * 7);
                //violation when strict set
  o_data(2) <= i_data(conv_integer(sel) * 3);
                //violation when strict set
end rtl;
```

STARC05-2.10.6.6

Do not describe more than one arithmetic operation in one line

Description

The STARC05-2.10.6.6 Tag flags assignment expressions having more than one arithmetic operation.

Using more than one operator in the one line may cause 2 bits to carry around the end. Bit-width on two sides of an assignment will be different and it cannot be recommended.

Also, when more than one operation is performed in one line, arithmetic operators will be linearly created. To clarify the order of operations, this kind of operation should be described as one operation per line.

By default, the STARC05-2.10.6.6 Tag does not consider arithmetic operators with static operand(s). Set the *strict* parameter to 1 to consider such operators.

NOTE: *The VHDL version of this Tag corresponds to the item 7 of Chapter 2, Section 10, Subsection 6 of the HDL Designs for IP Reuse guidelines.*

Message Details

Verilog

The following message appears at the location of an expression `<expr>` with more than one arithmetic operation:

The expression '`<expr>`' has more than one arithmetic operators

VHDL

The following message appears at the location of an assignment expression `<expr>` with more than one arithmetic operation:

The assignment expression '`<expr>`' has more than one arithmetic operators

Severity

Recommended

STARC05-2.10.7.1

Do not use arithmetic operation in the conditional expression of if statements.

Description

The STARC05-2.10.7.1 Tag flags conditional expressions of if constructs where arithmetic operations are described.

Arithmetic operation statements in the conditional expressions of if statement are not subject to resource sharing. Therefore, arithmetic operation should be done in assign statement using intermediate variables and those variables should be used in if-conditions. Such modeling helps in resource sharing.

The STARC05-2.10.7.1 Tag ignores static conditional expression.

Message Details

The following message appears at the location where an arithmetic expression is found in the conditional expression of an if construct:

Arithmetic operation performed in if-condition

Example

Verilog

Consider the following example:

```
module Top(in1, in2, clk, out1);
    input in1, in2;
    input clk;
    output reg out1;
    wire w1,w2,w3,w4;

    assign w1 = 1'b0;
    assign w2 = 1'b1;
    assign w3 = 1'b0;

    always @(posedge clk)
    begin
        if((w1>w2+w3) > 1'b1) // Violation
```

STARC05 Tags

```

        out1 <= in1;
    else if(w1+w2 < 1'b1) // Violation
        out1 <= in2;
    else if(w1+w2 > 1'b0) // Violation
        out1 <= in1 | in2;
end

```

```
endmodule
```

In the above example, the STARC05-2.10.7.1 Tag reports violations because of arithmetic operations in the conditional expressions of `if` statements.

VHDL

Consider the following example:

```

library ieee;
use ieee.std_logic_1164.all;

entity ent1 is
    port(ain1,ain2:in BIT;Q,QBAR:out BIT;aio:inout BIT);
end entity;

architecture arch1 of ent1 is

    begin
        aio <= ain1;
        Q <= aio;
        P1:process (ain1,ain2)
            variable NUM : integer := 13;
            variable NUM1 : integer := 6;
            variable NUM2 : integer := 7;
            constant zero :integer := 0;

            begin
                if(NUM1 + NUM2 > NUM) then -- Violation
                    if(NUM1 - NUM2 < NUM) then -- Violation
                        Q <= '1';
                    elsif(NUM1 - NUM2 < NUM) then -- Violation

```

```
        Q <= '0';
        elsif(NUM1 rem NUM2 = zero) then -- Violation

        Q <= ain1;
        end if;
        elsif(NUM1 - NUM2 < NUM) then -- Violation
            Q <= '0';
        end if;
    end process P1;

end arch1;
```

In the above example, the STARCO5-2.10.7.1 Tag report violations because of arithmetic operations in the conditional expressions of `if` statements.

Severity

Recommended

STARC05-2.10.7.2

**Common operational expressions should be described separately.
(Verilog)
Separate the common operational expressions in if-elsif branch
conditions (VHDL)**

Description

Verilog

The requirement of this Tag is covered in the [STARC05-2.7.2.1](#) Tag.

VHDL

The STARC05-2.10.7.2 Tag flags if-elsif-else construct branches that have common arithmetic operational expression or sub-expression as a branch found earlier.

Some synthesis tools do not detect resource sharing across operational expressions in conditional branch conditions. It is recommended to separate out a sub-expression that is common to a number of branch conditions and evaluate it separately.

Message Details

VHDL

The following message appears at the location where an if-elsif-else construct branch has common operational expression or sub-expression *<expr>* as a branch found earlier at line *<num>*:

Common logic '*<expr>*' in multiple conditional expressions
(First found at: *<num>*)

Severity

Recommended

VHDL Examples

Consider the following example where the expression $A+B$ is common in first and second branches:

```
library ieee;
use ieee.std_logic_1164.all
```

```
...
33:process(A, B, C)
34:  begin
35:    if(A+B-C >= "0011") then
36:      Y <= "11";
37:    elsif(A+B+C <= "1000") then
38:      Y <= "00";
39:    else
40:      Y <= "10";
41:    end if;
42:end process;
```

For this example, SpyGlass generates the following message:

Common logic '(a + b)' in multiple conditional expressions
(First found at: 35)

STARCO5-2.10.8.2

Keep the left side of the division or modulus operator within 12 bits.

Description

The STARCO5-2.10.8.2 Tag flags when the bit-width of the LHS expression of the division or modulus operator is not within 12 bits.

Even if the left side of the division operator is a constant, a divider is inferred and hence care should be taken about the bit-width.

In case of nested use of division/modulus operator, the STARCO5-2.10.8.2 Tag reports violation only for the outermost operator.

Consider the following examples when the STARCO5-2.10.8.2 Tag reports violation:

```
Parameter ONE = 16'h0001;
```

```
Parameter TWO = 32'h00000001;
```

- **ONE / a;** // violation
- (ONE % a + TWO) / b; only one violation for / operator
- (TWO % ONE) / a; // only one violation for / operator

Message Details

The following message appears when the width *<width>* of left-side expression *<expr>* of division/modulus operator exceeds 12 bits:

width *<width>* of left side expression *<expr>* of *<division/modulus>* operator exceeds 12 bits[Hierarchy: '*<hier-path>*']

Severity

Mandatory

STARC05-2.10.8.3

If right side of the division or modulus operator is not a power of two, keep it within 8 bits.

Description

The STARC05-2.10.8.3 Tag flags when the bit-width of the RHS expression of the division or modulus operator is not within 12 bits.

If right hand side expression is a power of two, this Tag will not report the violation (0 and 1 are not considered as power of two).

In case of nested use of division/modulus operator, the STARC05-2.10.8.3 Tag reports violation only for the outermost operator.

The area and delay of a divider increase more rapidly with an increase in the bit-width of the right side of an operator than that of the left side. When the right side of the division or modulus operator is a power of two, logic synthesis tools infer a shifter. In this case, there is no limit to the bit-width.

Consider the following examples:

```
Parameter ONE = 16'h0001;
```

```
Parameter TWO = 32'h00000001;
```

- `a / TWO;` // violation
- `a % (ONE + b / TWO);` // only one violation for %
- `a / (TWO % ONE);` // only one violation for / operator
- `a / 10'b0000000000` // violation
- `a / 10'b0000000001` // violation
- `a / 10'b0010000000` // no violation; power of two

Message Details

The following message appears when the width *<width>* of the right-side expression *<expr>* of division/modulus operator exceeds 8 bits:

```
width <width> of right side expression <expr> of <division/
modulus> operator exceeds 8 bits[Hierarchy: '<hier-path>']
```

STARCO5 Tags

Severity

Mandatory

STARC05-2.11.1.2

Bit change in state transition should be minimal (use Gray code). (Verilog)
Bit change in FSM state transitions should be minimized (use Gray code). (VHDL)

Description

Verilog

The STARC05-2.11.1.2 Tag flags case constructs modeling FSMs where the number of bits changed between states is more than the specified number.

By default, STARC05-2.11.1.2 Tag flags case constructs where the number of bits changed between states is more than one. Use the [num_max_bit_changes](#) Tag parameter to change the maximum allowed number of bit changes during state transitions.

If the Finite-state machine is not too large, it is recommended to minimize changes in bits between state transitions by using Gray code. Finite-state machine output is normally used in the signal data path and hence following this guideline would improve speed of the circuit.

VHDL

The STARC05-2.11.1.2 Tag flags case constructs in FSM descriptions where the number of bits changed between states is more than one.

The STARC05-2.11.1.2 Tag also generates statistical data regarding the fraction of total state transitions that do not follow Gray encoding. By default, in addition to the explicitly coded transitions, the STARC05-2.11.1.2 Tag also considers transitions where the source state happens to be handled by the others clause of the case construct.

By default, the STARC05-2.11.1.2 Tag flags case constructs in FSM descriptions where the number of bits changed between states is more than one. Use the [num_max_bit_changes](#) Tag parameter to specify a different number.

For lower logic area, it is recommended to minimize changes in bits of FSM State Variable during state transitions by using Gray code. It is, however, not feasible to have a fully Gray-coded machine, but transitions following Gray code should be ideally maximized.

Message Details

Verilog

The following message appears at the first line of a case construct that has `<num1>` clauses out of a total of `<num2>` (effective `percentAge` `<num3>%`) have more than one bit changing in the state variable:

'`<num1>`' out of '`<num2>`' ('`<num3>%`') state transitions have more than 1 bit changing in state variable

VHDL

The following message appears at the first line of a case construct modeling an FSM that has `<num1>` clauses out of a total of `<num2>` have more than `<$num_max_bits_change>` bits changing in the state variable:

'`<num1>`' out of '`<num2>`' state transitions have more than `<$num_max_bits_changes>` bit(s) changing in FSM state variable

Severity

Recommended3

STARC05-2.11.1.4

Number of states in a FSM should not exceed specified limit

Description

The STARC05-2.11.1.4 Tag flags FSM descriptions with more than 40 states.

For Verilog designs, SpyGlass calculates the number of FSM states to be the number of case construct clauses excluding the default clause or the number of if-else-if construct branches excluding the else branch.

As the number of states in an FSM increases, the number of registers required also increases, thus resulting in larger circuits.

For VHDL designs, SpyGlass calculates the number of FSM states to be the number of states handled by the case construct clauses plus the number of states handled by the others clause.

By default, the STARC05-2.11.1.4 Tag flags FSMs that have more than 60 states. Use the [starc2005_num_max_fsm_states](#) Tag parameter to specify a different number.

FSMs with a large number of states have a large circuit area requirement and a complex state variable assignment problem.

NOTE: *The STARC05-2.11.1.4 Tag supports generate-for block.*

Message Details

Verilog

The following message appears at the location where an FSM description with `<num>` states is encountered and `<num>` is more than the number specified by the `starc2005_num_max_fsm_states` Tag parameter:

```
Number of FSM states (<num>) is more than  
$starc2005_num_max_fsm_states
```

VHDL

The following message appears at the location where an FSM description with `<num>` states is encountered and `<num>` is more than the number specified by the `starc2005_num_max_fsm_states` Tag parameter:

```
Number of FSM states (<num>) is more than maximum allowed limit  
($starc2005_num_max_fsm_states states)
```


STARCO5 Tags

Severity

Recommended2

STARC05-2.11.2.1

FSM description should not be mixed with other descriptions in the same module. (Verilog)
FSM description should not be mixed with non-FSM descriptions in the same module (VHDL)

Description

Verilog

The STARC05-2.11.2.1 Tag flags non-FSM descriptions that are in the same module as FSM descriptions.

The STARC05-2.11.2.1 Tag requires that FSM and non-FSM logic should be separated into different modules.

Mixing different types of logic in one module makes it difficult to create a single effective synthesis strategy for that module.

VHDL

The STARC05-2.11.2.1 Tag flags non-FSM descriptions that are in the same hierarchical layer as FSM descriptions.

The STARC05-2.11.2.1 Tag requires that FSM and non-FSM logic should be separated into different hierarchical layers.

Mixing different types of logic in one hierarchical layer makes it difficult to create a single effective synthesis strategy for that hierarchical layer.

Message Details

Verilog

The following message appears at the first line of a non-FSM description if the module also has an FSM description (starting line *<num>* and filename *<filename>*):

Non-FSM description mixed with FSM description (line '*<num>*' filename '*<filename>*') in same module

VHDL

The following message appears at the first line of a non-FSM description if the module also has an FSM description (starting line *<num>*):

Non-FSM Logic Element(s) mixed with FSM logic exist in module '*<entity-name>*(*<arch-name>*)'

STARCO5 Tags

Severity

Recommended

STARC05-2.11.4.1

One-hot encoding should be used for allocating states

Description

Verilog

The STARC05-2.11.4.1 Tag flags Finite-state machine descriptions with less than 15 states that do not use one-hot encoding for allocating states.

The one-hot encoding method of state allocation is advantageous because it does not need any decoding circuitry, and hence its speed is much faster and the circuit size is smaller. However, this style needs more registers and hence the area increases. If there are less than 15 states in the Finite-state machine, the increase in area is not much (20-30%).

VHDL

The STARC05-2.11.4.1 Tag flags Finite-state machine descriptions that do not use one-hot encoding for allocating states.

The one-hot encoding method of state allocation is advantageous because it does not need any decoding circuitry, and hence its speed is faster and the circuit size is smaller. However, this style needs more registers and hence the area increases.

Message Details

Verilog

The following message appears at the location of a state label *<name>* that does not use the one-hot encoding style:

```
State label '<name>' does not follow one-hot encoding style
```

VHDL

The following message appears at the location of a state label *<name>* that does not use the *<num>* encoding style:

```
State label '<name>' does not follow '<num>' encoding style
```

Severity

Recommended

STARC05-2.11.4.2

**State values should be defined using parameters or macros.
(Verilog)
Use constant to facilitate allocation changes (VHDL)**

Description

Verilog

The STARC05-2.11.4.2 Tag flags combinational part of an FSM description where the state values are explicit constant values.

It is recommended to describe state values using parameter encoding or macro encoding. Thus it is easier to modify the state allocations according to circuit requirements or encoding patterns as all states can be defined together in one place.

By default, this Tag checks the one-hot labels for violations. Set the value of the [ignore_one_hot_labels](#) parameter to `yes` if you do not want to process one-hot labels for violations.

VHDL

The requirement of this Tag is covered by the [STARC05-2.11.1.2](#) Tag.

Message Details

Verilog

The following message appears at the location of the combinational part of an FSM where the state values have been described using explicit constant values:

Use only parameter encoding or macro encoding to define FSM states

Severity

Recommended

STARC05-2.11.5.2

Do not assign state encoding by attaching attributes to the state variable which is declared as a type (VHDL)

Description

The STARC05-2.11.5.2 Tag flags state variables that have been declared as a user-defined type and have used in FSM state encoding by attaching attributes.

The STARC05-2.11.5.2 Tag flags state variables as in the following example:

```
type FSM_STATE is (ST_INIT, ST_ADD, ST_SUB, ST_MUL);  
attribute ENCODING is STRING;  
attribute ENCODING of FSM_STATE:type is "00 01 11 10";
```

With some synthesis tools, it is possible to allocate logic values for the state variable (declared to be of a specific type) using attributes. This practice, however, is not universally supported and decreases portability to other logic synthesis tools.

Message Details

The following message appears at the location where a state variable *<var-name>* is declared as a user-defined type and an attribute *<attr-name>* is attached to the state variable before using it in FSM description:

Attribute '*<attr-name>*' found on enumerated type '*<var-name>*' used for encoding FSM states

Severity

Recommended2

STARC05-2.2.2.1

Signals read inside a combinational always block must be present in the sensitivity list (Verilog)
Signals read inside a combinational process block must be present in the sensitivity list (VHDL)

Description

The *STARC05-2.2.2.1* Tag reports signals or variables that are read inside a combinational always/process construct but are missing from the construct's sensitivity list.

Signals missing from the sensitivity list can lead to a mismatch between pre- and post-synthesis simulation. Before synthesis, the simulation only evaluates changes in the combinational logic when the signals in the sensitivity list change. However, the synthesis process generate logic that reads all required values, whether or not they are in the sensitivity list. In effect, the missing signals are added to the sensitivity list. Thus, the post-synthesis simulation results are different from those of the pre-synthesis simulation.

NOTE: *The Tag supports the multi-line functionality.*

To ensure that the pre-synthesis simulation matches the post-synthesis simulation results, you must add the missing signals to the sensitivity list of the containing always/process construct.

NOTE: *The Tag ignores record type constructs in VHDL designs.*

The Tag does not report delay variables that are read in the always construct but are not present in the sensitivity list. Set the [strict](#) parameter to `yes` to report such variables.

Message Details

Verilog

The following message appears at the sensitivity list of a combinational always construct where signal `<sig-name>` is read in the construct but is not present in the sensitivity list of the construct:

```
Signal '<sig-name>' missing from sensitivity list [Hierarchy:  
'<module-hier-name>']
```

VHDL

The following message appears at the first line of a process construct where a signal `<sig-name>` is read inside the process construct but is not specified in the process' sensitivity list:

The `<type> '<sig-name>'` (or some of its bits) read in the process is missing from the process sensitivity list.
[Hierarchy: '`<hier-path>`']

Severity

Mandatory

STARC05-2.2.2.2a

A variable specified in the sensitivity list must be read in the contained block. (Verilog)

A signal should not be included in sensitivity list of a process if it is not read in that process. (VHDL)

Description

Verilog

The STARC05-2.2.2.2a Tag flags signals that are in the sensitivity list of a combinational always construct but are not completely read in the construct.

By default, The STARC05-2.2.2.2a Tag flags a separate message for each Tag-violating signal in each instance of a design unit. Set the [ignoreMultipleInstances](#) Tag parameter to flag only one message for a Tag-violating signal of a design unit instantiated multiple times in the design.

While such signals do not affect the functionality, they slow down the design simulation. Also, the logic related to a signal in the sensitivity list may have been accidentally removed or not added, which results in wrong functionality.

Many simulators re-evaluate the always construct on each change in a redundant signal, even though the evaluation itself is redundant.

VHDL

The STARC05-2.2.2.2a Tag flags signals that are not required in the sensitivity list of a process construct.

In process constructs modeling combinational logic, the STARC05-2.2.2.2a Tag flags those signals that are present in the sensitivity list but are not completely read in the process construct.

In process constructs modeling sequential logic, the STARC05-2.2.2.2a Tag flags those signals present in the sensitivity list that are not being used as clock or reset signals. Consider the following example:

```
SEQUENTIAL_PROC: process(clk, reset, in1, in2)
begin
    ...
end process;
```

The STARC05-2.2.2.2a Tag flags signals `in1` and `in2` even if they are being read in the process construct as they are not being used as clock or

reset.

By default, The STARCO5-2.2.2.2a Tag flags a separate message for each Tag-violating signal in each instance of a design unit. Set the [ignoreMultipleInstances](#) Tag parameter to flag only one message for a Tag-violating signal of a design unit instantiated multiple times in the design.

Detecting unwanted signals in the sensitivity list is potentially important in two ways:

- If the signal is really not required in the process, including it the sensitivity list unnecessarily slows simulation; however it will not lead to functional errors.
- Perhaps this signal should be read in the process and was accidentally removed; this may signal a functional error.

By default, the STARCO5-2.2.2.2a Tag reports a violation only when none of the bits of signal are read in a combinational always block. If you set the value of the [strict](#) parameter to `yes`, the STARCO5-2.2.2.2a reports a violation for signals that are partly read in the block.

Message Details

Verilog

When the value of the [strict](#) parameter is not set, the following message appears at the location of the sensitivity list of an always construct where signal/variable `<name>` is present in the sensitivity list of the always construct but is not required:

```
The signal/variable '<name>' is not required in sensitivity
list as none of its bits is read in always block [Hierarchy:
'<hier-path>']
```

When the value of the [strict](#) parameter is set to `yes`, the following message appears at the location of the sensitivity list of an always construct where signal/variable `<name>` is present in the sensitivity list of the always construct but is not required:

```
The signal/variable '<name>' (or some of its bits) which is not read in
always block is not required in the sensitivity list [Hierarchy: '<hier-path>']
```

VHDL

When the value of the [strict](#) parameter is set to `no`, the following message appears at the location of the sensitivity list of a process construct where

signal/variable *<name>* is present in the sensitivity list of the always construct but is not required:

The *<type>* '*<name>*' is not required in sensitivity list as none of its bits is read in process [Hierarchy: '*<hier-path>*']

Where, *<type>* can be in port or signal.

When the value of the *strict* parameter is set to yes, the following message appears at the location of the sensitivity list of a process construct where signal/variable *<name>* is present in the sensitivity list of the always construct but is not required:

The *<type>* '*<name>*' is not required in sensitivity list as only some or none of its bits are read in process [Hierarchy: '*<hier-path>*']

Where, *<type>* can be in port or signal.

Severity

Recommended

STARC05-2.2.2.2b

Do not define constants in the sensitivity list

Description

The STARC05-2.2.2.2b Tag flags constants (locally static and globally static) included in sensitivity lists.

Message Details

The following message appears at the location of the sensitivity list of an always construct where a constant `<const>` is present:

Constant '`<const>`' used in sensitivity list[Hierarchy: '`<hier-path>`']

Severity

Recommended

STARC05-2.2.2.3v

Do not use wait statement in process statements (Sensitivity list should be used). (VHDL)

Description

The STARC05-2.2.2.3v Tag flags process constructs with wait statements.

In VHDL designs, wait statements should not be used in process constructs. For example,

```
library ieee;  
use ieee.std_logic_1164.all  
  
process  
begin  
    YOUT <= A and B;  
    wait on A,B;  
end process;
```

Logic synthesis tools do not support such descriptions. The wait until construct can be used in sensitivity lists when inferring flip-flops. However, it should not be described since description style may become confusing.

Message Details

The following message appears at the location of a wait statement used in a process construct:

wait statement used in process

Severity

Mandatory

STARC05-2.2.2.3b

At least one event expression must be described for an always construct. (Verilog)

Description

The STARC05-2.2.2.3b Tag flags always constructs without any event expressions.

Logic synthesis with this kind of description might be possible but not in many cases. Also, description without any event expression "@" has risk of unlimited loop at simulator that is hazardous.

Message Details

The following message appears at the location of an always construct that has no event constructs:

At least one event expression should be described for an always construct

Severity

Mandatory

STARC05-2.2.3.2

Do not use nonblocking assignments for assigning initial values in always blocks. (Verilog)

Description

The STARC05-2.2.3.2 Tag flags nonblocking assignment statements used to assign initial values in always constructs.

Assigning into the signal is postponed in the case of nonblocking assignment. However, designs using nonblocking assignments generally execute the assignment statement in order, and this can cause simulators zero-time glitches.

The STARC05-2.2.3.2 Tag considers that a flip-flop output has been assigned an initial value in an always construct if there exists a direct assignment to the flip-flop output followed by a conditional assignment to the same signal in the same always construct.

NOTE: *The STARC05-2.2.3.2 Tag supports generate-for block.*

Message Details

The following message appears at the location where a signal *<sig-name>* is assigned an initial value using a nonblocking assignment statement:

Signal '*<sig-name>*' is assigned initial value using nonblocking assignment in always block

Severity

Mandatory

STARC05-2.3.1.3

Flip-flop data paths should have delays

Description

The STARC05-2.3.1.3 Tag flags those flip-flop descriptions that do not have delays in their data paths.

To avoid excessive messages, the STARC05-2.3.1.3 Tag flags only 10 messages per module and 500 messages per design. Set the [allviol](#) Tag parameter to *yes* or *<Tag-name>* to report all violation messages.

NOTE: *The STARC05-2.3.1.3 does not flag initialization (sets or resets) without delay. Consider the following Verilog example:*

```
always @(posedge CLK or negedge RST)
  if (!RST)
    Q <= 1'b0;
  else
    Q <= D;
```

NOTE: *The STARC05-2.3.1.3 does not flag output assignment in reset description.*

In multi-clock designs, delay values should be specified at the D input assignments in flip-flop descriptions to minimize potential race issue during simulation.

Message Details

The following message appears at the location where a flip-flop data path without a delay is encountered:

Insert delay in flip-flop data path

Severity

Recommended3

Verilog Examples

Consider the following flip-flop description example:

```
always @(posedge CLK)
begin
  Q <= D;
```



```
end
```

For this example, SpyGlass generates the following message:

Insert delay in flip-flop data path

To fix this message, insert delay in the data path as follows:

```
always @(posedge CLK)
begin
    Q <= #5 D;
end
```

STARC05-2.3.1.4

Do not use delay values other than in always constructs that infer FFs. (Verilog)
Paths other than flip-flop data paths must not have delays. (VHDL)

Description

Verilog

The STARC05-2.3.1.4 Tag flags delay values used in combinational constructs.

Delay values are specified in flip-flop assignment expressions. However, such values should not be used elsewhere such as combinational circuits. If you specify delay values in a combinational circuit, simulation might become dependent on those delay values.

Instead of specifying delay values directly, you can parameterize those delay values (`#DELAY`). This enables you to change the delay values based on different technologies. This also facilitates simulations in RTL that take delays into consideration.

VHDL

The STARC05-2.3.1.4 Tag flags delays for paths specified other than flip-flop data paths. Delay values are ignored during synthesis. Simulation of designs with delay values used in combinational circuits is delay value dependent. However, you may use a delay value at D input assignment to avoid racing problems during RTL simulation.

Message Details

Verilog

The following message appears at the location where delay assignment is encountered in a combinational block:

Delay value used in combinational block

VHDL

The following message appears at the location where delay assignment is encountered in a path other than flip-flop data path:

Delay used in path other than flip-flop data path

STARCO5 Tags

Severity

Recommended1

STARC05-2.3.1.5a

Delay values must be integral

Description

Verilog

The STARC05-2.3.1.5a Tag flags fractional delay values in the design.

In Verilog designs, delay precision should be controlled using the ``timescale` directive. A fractional delay value is normally truncated to the current timescale.

Using fractional delay values may cause efficiency problems in some simulators. Therefore, only integers should be used for delay values.

NOTE: *The STARC05-2.3.1.5a Tag does not flag negative integer delay values. See the [STARC05-2.3.1.5b](#) Tag to catch negative delay values.*

VHDL

The STARC05-2.3.1.5a Tag flags delays with non-integral values.

In VHDL designs, using decimal point in delay value may cause efficiency problems in some simulators. Therefore, integers should be used for delay value.

Message Details

Verilog

The following message appears at the location where a fractional delay value is encountered:

Fractional delay value `<value>` used

where, `<value>` is a delay expression.

VHDL

The following message appears at the location where a fractional delay value is encountered:

Non-integral delay value `<value>` used

where, `<value>` is a delay expression.

STARCO5 Tags

Severity

Mandatory

STARC05-2.3.1.7b

Do not use function in asynchronous reset description.

Description

Verilog

The functionality of STARC05-2.3.1.7b (Verilog) Tag is already covered by the [STARC05-2.1.2.1](#) Tag

VHDL

The STARC05-2.3.1.7b Tag flags flip-flops that use functions in their asynchronous reset description.

Using a function in the condition expression for an asynchronous reset might cause an error since it does not match any condition of logic synthesis tools for generating flip-flops.

When function is used for conditional expression to infer reset in if statement, it will not be synthesized correctly. Function statement should not be used for asynchronous reset line. It is safer to consider function statement only for combinational circuits, and to use a call from continuous assignment.

Message Details

VHDL

The following message appears where a function <func-name> is used for asynchronous reset <reset-name> in a if statement:

Function '<function-name>' used for asynchronous reset '<reset-name>'

Severity

Mandatory

STARC05-2.3.1.8

Do not use wait construct to describe clock style (VHDL)

Description

The STARC05-2.3.1.8 Tag flags wait statements used in clocking description. Asynchronous reset condition cannot be put in flip-flop description if wait statement is used to describe the clock event. Therefore, clock event must be described using if statement only.

The STARC05-2.3.1.8 Tag flags the following clocking styles, which use wait statements:

```
wait until clk'event and clk = 1;  
wait until not clk'stable and clk = 1;  
wait until rising_edge(clk);  
wait until falling_edge(clk);
```

The STARC05-2.3.1.8 Tag does not flag a message if the clocking style that is using wait statement, is not synthesizable.

Message Details

The following message appears at the location where a clocking description using wait statement is encountered:

wait construct used for clock description

Severity

Mandatory

STARC05-2.3.1.9

Functions 'rising_edge/falling_edge' should not be used in the design. (VHDL)

Description

The STARC05-2.3.1.9 Tag flags calls of rising_edge/falling_edge functions.

Clock description should always use event or stable attributes as they are defined in standard packages whereas functions rising_edge and falling_edge are defined in std_logic_1164 package. Therefore, using these functions should be avoided.

Message Details

The following message appears at the location of a rising_edge/falling_edge function call:

Function '<func-name>' used in the design

Where <func-name> is rising_edge or falling_edge.

Severity

Reference

STARC05-2.3.2.2

Do not use blocking and nonblocking assignments together in same always block. (Verilog)

Description

The STARC05-2.3.2.2 Tag flags the always constructs that have both blocking and nonblocking assignment statements.

In general, blocking assignments should be used in combinational blocks and nonblocking assignments in sequential blocks. Mixing the two assignment types in the same always construct can lead to mismatches between pre- and post-synthesis simulation. The task calls are considered as blocking assignments, so the violations are reported accordingly.

Message Details

The following message appears at the location where a different assignment type is encountered:

Both non-blocking '<non-blocking-stmt>' and blocking assignments '<blocking-stmt>' used in same always block

For a non-blocking or blocking assignment, the previous respective blocking or non-blocking assignment is also highlighted.

Severity

Mandatory

Examples

Consider the following example:

```
module top(clk,in1,in2,out1,out2);  
    input in1, in2,clk;  
    output out1,out2;  
  
    reg out1,out2;  
  
    always @(posedge clk)  
        begin
```

```
        out1 <= in2;  
        out2 = ~in1;  
    end
```

```
endmodule
```

In the above example, the STARCO5-2.3.2.2 Tag reports a violation because the blocking and nonblocking assignments are present together in the same `always` block. The following message is reported by this example:

Both non-blocking 'out1 <= in2;' and blocking assignments 'out2 = ~in1;' used in same always block

STARC05-2.3.2.4

Variables, if used, must be assigned to a signal before the end of the process (VHDL)

Description

The *STARC05-2.3.2.4* Tag reports variables that are used but are then not assigned to any signal before end of the process.

The Tag reports violations whenever a variable is written in the process but no signal assignment to this variable value exists in a subsequent statement of the process body.

Variables being used inside sequential processes may end up being inferred as flip-flops. This situation might lead to problematic behavior and RTL might not be acceptable for some synthesis tools. To avoid this inference, you must assign the values of these variables to signals before end of the process.

Violation for combinational processes are not reported.

In addition, the Tag does not report violations for variables that are assigned to signals through intermediate variables.

By default, the [check_var_in_condition](#) parameter is set to `yes`. This indicates the Tag does not report violations for the following cases:

- Variables used inside `if-else`, `case`, or `while` loop conditions
- Variables used as a loop index of the `for` loop
- Variables used as an array index of another variable

Set the [check_var_in_condition](#) parameter to `no` to report violations for such cases.

Message Details

The following message appears at the location where the variable `<var-name>`, which is not assigned to a signal before end of the process, is written:

variable '`<var-name>`' not assigned to any signal before the end of the process

Severity

Mandatory

VHDL Examples

Example 1

The following example demonstrates the behavior of the *STARC05-2.3.2.4* Tag:

```
library ieee;
use ieee.std_logic_1164.all

library ieee;
use ieee.std_logic_1164.all;

entity STARC_2324 is
end STARC_2324;

architecture RTL of STARC_2324 is
signal CLK : std_logic;
begin
process (CLK)
variable A,B,C,D,E,F : std_logic;
begin
    if (CLK'event and CLK = '1') then
        A := '1';           //no violation for A here
                           //as it is assigned to
                           //signal CLK via variables B, C, and E
        B := '1';           //no violation for B
                           //as it is assigned to
                           //signal CLK via variables C and E
        C := '1';           //no violation for C
                           //as it is assigned to
                           //signal CLK via variable E
        D := '1';           //no violation for D
                           //as it is assigned to
                           //signal CLK via variable E
        E := '1';           //no violation for E
```

STARCO5 Tags

```

                                //as it is assigned to signal CLK
F := '1';                       //violation for F
                                //as it is not assigned to any signal

B := A;
C := B;
E := C XOR D;
CLK <= E;
A := B;                         //violation for A here
                                //as it is set here but not assigned to
                                //any signal before the end of the process
    end if;
end process;
end RTL;

```

Example 2

In the following example, the *STARCO5-2.3.2.4* Tag, by default, does not report any violation for variables A, B, D, E, F, G, P, Q, R, and S. The Tag reports violations for these variables when the [check_var_in_condition](#) parameter is set to no.

```

library ieee;
use ieee.std_logic_1164.all;

entity test is
port
(
    clk : in std_logic;
    RST : in std_logic;
    in1 : in std_logic_vector (3 downto 0)
);

end entity;

architecture rtl of test is
    signal C : natural;
    begin
        process
            type arr is array(0 to 3) of std_logic_vector (4 downto
0);

```

```
variable P,Q,R,S : arr;
variable A,B,D,E,F,G : integer;
variable Z : natural;
begin
    A:=1;B:=1;D:=1;
//violation for A, B, and D when check_var_in_condition=no
    E:=1;F:=1;G:=1;
//violation for E, F, and G when check_var_in_condition=no
    if (clk'event and clk = '1') then
        if(F = 1) then
            C<=Z;
        end if;
        P := Q;
//violation for P when check_var_in_condition=no
        Q := R;
//violation for Q when check_var_in_condition=no
        R := P;
//violation for R when check_var_in_condition=no
        S := P;
//violation for S when check_var_in_condition=no
        for y in Q(0)(1) to Q(D)(A) loop
            C <= Z;
        end loop;
        while(R(G)(B) = S(0)(E)) loop
            C <= Z;
        end loop;
    end if;
end process;
end architecture;
```

STARC05-2.3.3.2a

Do not use both edges of the same clock in an always block.

(Verilog)

Do not use both edges of the same clock in a single process block

(VHDL)

Description

Verilog

The *STARC05-2.3.3.2a* Tag flags always constructs that have statements using both edges of the clock signals.

always constructs with both edges of the clock signals cannot be synthesized. Such constructs do not exist independently and are difficult to implement in hardware.

VHDL

The *STARC05-2.3.3.2a* Tag flags process constructs that have statements using both edges of the clock signals.

process constructs with both edges of the clock signals cannot be synthesized. Such constructs do not exist independently and are difficult to implement in hardware.

Message Details

Verilog

The following message appears at the first line of the always construct in hierarchy *<hier-path>* that has statements using both edges of the clock signal *<clk-name>*:

Both edges of the same clock '*<clk-name>*' used in same always block. [Hierarchy: '*<hier-path>*']

VHDL

The following message appears at the first line of the process construct that has statements using both edges of the clock signal *<clk-name>*:

Both edges of the same clock '*<clk-name>*' used in same process block

Severity

Mandatory

Verilog Examples

Consider the following example where both edges of the clock signal `clk` are used:

```
module test (in, clk, out1, out2);  
    input in, clk;  
    output out1, out2;  
  
    reg out1, out2;  
  
    always  
    begin  
        @ (posedge clk)  
            out1 <= in;  
        @ (negedge clk)  
            out2 <= ~in;  
    end  
endmodule
```

For this example, SpyGlass generates the following message at the first line of the `always` construct:

Both edges of the same clock 'clk' used in same always block.
[Hierarchy: 'test']

STARC05-2.3.4.1

Do not specify flip-flop initial value using initial construct. (Verilog)

Description

The STARC05-2.3.4.1 Tag flags flip-flop initial values specified in the initial constructs.

While initial blocks are allowed, they are not synthesizable. Thus, the statement resetting a flip-flop through the initial construct is ignored.

NOTE: The STARC05-2.3.4.1 Tag supports generate-for block.

Message Details

The following message appears at the location where the flip-flop *<flop-name>* is reset in an initial construct in hierarchy *<hier-path>*:

Flip-flop '*<flop-name>*' is initialized in initial block
[Hierarchy: '*<hier-path>*']

Severity

Mandatory

Verilog Examples

Consider the following example where the flip-flop output out is initialized in an initial block:

```
module test (in, out, clk);
    input in1, clk;
    output out;

    reg out;
    initial
        begin
            out = 1'b0;
        end
    always @(posedge clk)
        out <= in;
endmodule
```

For this example, SpyGlass generates the following message at the location where the flip-flop output `out` is being assigned a value in the initial block:

Flip-flop 'out' is initialized in initial block

STARC05-2.5.1.5b

A signal without a resolution function must not have multiple drivers. (VHDL)

Description

The STARC05-2.5.1.5b Tag flags nets that do not have resolution function but have multiple drivers.

It is an elaboration error if a non-resolved signal is driven by multiple sources.

NOTE: *Currently, multi-dimensional arrays are not being handled. Also, the records are considered as SCALAR type.*

Message Details

The following message appears when signal name *<sig-name>* does not have a resolution function but has multiple drivers:

Non-resolved signal '*<sig-name>*' has multiple drivers
[Hierarchy: '*<hier-path>*']

Severity

Mandatory

STARC05-2.6.1.2

Use intermediate variables when same logic is used in more than one place.

Description

The STARC05-2.6.1.2 Tag flags use of duplicated logic without any intermediate variables.

The STARC05-2.6.1.2 Tag flags violation when no intermediate variable is used in case of duplicated logics. Duplicated logic means that the same arithmetic operator (plus, minus, multiplier, division, and percent operators) with multiple operands, which are identical, has been used in multiple statements over the scope of the module.

By default, the STARC05-2.6.1.2 tag reports violation for expressions that contain for-loop index along with the constants. Set the value of the `ignore_for_index` parameter to yes to not report violations for expressions that contain for-loop index along with constants.

Message Details

The following message appears when intermediate variables are not used for expression `<expr>` in case of duplicated logic at (comma-separated) line(s) `<num(s)>`:

Intermediate variables should be used for '`<expr>`' in case of duplicated logic [Line numbers: `<num(s)>`]

Severity

Recommended2

STARC05-2.6.1.3

always construct has more than five output signals. (Verilog)
A process block should not have more than five outputs (VHDL)

Description

Verilog

The STARC05-2.6.1.3 Tag flags always constructs that have more than five signals being assigned.

By default, the STARC05-2.6.1.3 Tag flags always constructs that have more than 5 outputs. Use the [alwaysblk_optcount](#) Tag parameter to specify a different number.

NOTE: *The STARC05-2.6.1.3 Tag does not count assignments to different bit-slices of a vector net as different assignments for the purpose of Tag-checking. All such assignments for one vector net are counted as one.*

VHDL

The STARC05-2.6.1.3 Tag flags process blocks that have more than five output signals, ports, or variables being assigned.

By default, the STARC05-2.6.1.3 Tag flags process blocks where more than 5 output signals, ports, or variables are being assigned. Use the [processblk_optcount](#) Tag parameter to specify a different number.

NOTE: *The STARC05-2.6.1.3 Tag does not process bit-slices of the same vector net as multiple output signals being driven.*

Consider the following example where different bit-slices of signal `data1` are assigned in different assignment statements:

```
process (clk1)
begin
    data1[(0) <= in1;
    data1( 3 downto 0) <= in2;
    data1(7 downto 4) <= in3;
    data2 <= in4;
    data3 <= in5;
    data4 <= in6;
    data5 <= in7;
end process;
```

For this example, SpyGlass counts the number of signal outputs to be

five (data1 to data5) and thus, will not generate a message.

Message Details

Verilog

The following message appears at the first line of an always construct that has *<num>* of signals being assigned and *<num1>* is more than the limit set by the `alwaysblk_optcount` Tag parameter:

```
Number of signals assigned in always block ('<num>') is more
than $alwaysblk_optcount
```

VHDL

The following message appears at the first line of a process block where the total number of output signals, ports, or variables being assigned (*<num>*) exceeds the maximum limit set by the `processblk_opcount` Tag parameter:

```
Number of outputs assigned in process block (Count: <num>) is
more than $processblk_opcount
```

Severity

Recommended

STARC05-2.6.1.4a

Always/Process construct should not have more than 300 lines

Description

Verilog

The STARC05-2.6.1.4a Tag flags always constructs that are longer than 300 lines.

By default, the STARC05-2.6.1.4a Tag flags always constructs that have more than 300 HDL, comment, and blank lines.

By default, the STARC05-2.6.1.4a Tag flags always constructs that are longer than the value specified by the parameter [starc2005_always_line1](#) and shorter than or equal to the value specified by the parameter [starc2005_always_line2](#) Tag parameter.

Dividing the logic description in to smaller blocks makes it easier to comprehend the nature and association of input and output signals. While the size of logic generated is not related to the number of lines, each single execution unit (like always constructs) must not contain too many lines.

VHDL

The STARC05-2.6.1.4a Tag flags process constructs that are longer than 300 lines.

By default, the STARC05-2.6.1.4a Tag flags process blocks with more than 300 HDL, comment, and blank lines.

By default, the STARC05-2.6.1.4a Tag flags always constructs that are longer than the value specified by the [starc2005_process_line1](#) Tag parameter and shorter than or equal to the value specified by the [starc2005_process_line2](#) Tag parameter.

NOTE: *In a process block, the line number is counted from the process line till the end process [label] line as shown in the following example:*

```
PROC_1:
    process -- start line
        begin
            ...
        end process PROC_1; -- end line
```

Dividing the logic description in to smaller blocks makes it easier to

comprehend the nature and association of input and output signals. While the size of logic generated is not related to the number of lines, each single execution unit (like process block) should not contain too many lines.

Message Details

Verilog

The following message appears at the first line of an always construct where the total number of lines (*<num>*) exceeds the limit set by the `starc2005_always_line1` Tag parameter:

```
Always block contains '<num>' lines (exceeds
$starc2005_always_line1)
```

VHDL

The following message appears at the first line of a process block where the total number of lines (*<num>*) exceeds the limit set by the `starc2005_process_line1` Tag parameter:

```
Number of Lines in process block (Count: <num>) is more than
$starc2005_process_line1
```

Severity

Recommended3

STARC05-2.6.1.4b

Always/Process construct should not have more than 2000 lines.

Description

Verilog

The STARC05-2.6.1.4b Tag flags always constructs that are longer than the value specified by the [starc2005_always_line2](#) Tag parameter.

Dividing the logic description in to smaller blocks makes it easier to comprehend the nature and association of input and output signals. While the size of logic generated is not related to the number of lines, each single execution unit (like always constructs) must not contain too many lines.

NOTE: *Comment lines and blank lines present inside an always block are also included in the line count of the always block.*

VHDL

The STARC05-2.6.1.4b Tag flags process constructs that are longer than the value specified by the parameter [starc2005_process_line2](#) Tag parameter.

Dividing the logic description in to smaller blocks makes it easier to comprehend the nature and association of input and output signals. While the size of logic generated is not related to the number of lines, each single execution unit (like process constructs) must not contain too many lines.

NOTE: *Comment lines and blank lines present inside a process block are also included in the line count of the process block.*

Message Details

Verilog

The following message appears when the number of lines *<num>* in the always block exceeds the number specified by the [starc2005_always_line2](#) Tag parameter:

Always block contains '*<num>*' lines (exceeds \$starc2005_always_line2)

VHDL

The following message appears when the number of lines *<num>* in the process block exceeds the number specified by the [starc2005_process_line2](#) Tag parameter:

Process block contains '<num>' lines (exceeds
\$starc2005_process_line2)

Severity

Mandatory

STARC05-2.6.2.1

Do not describe more than one statements (if/case/while/for/forever/repeat) separately within a single always block (Verilog)
Do not describe more than one statements (if/case/while/for/loop) separately within a single process block (VHDL)

Description

Verilog

The STARC05-2.6.2.1 Tag flags always constructs that have more than one if, case, for, while, foreach, or repeat constructs.

Thus, the STARC05-2.6.2.1 Tag flags an always construct that has two if constructs, or one if construct and one case construct, and so on.

The STARC05-2.6.2.1 Tag does not flag the always constructs that have nested constructs of one type. Thus, the following model is allowed:

```
if (...)
    if (...)
        if (...) // no violation
    ...
...
```

The STARC05-2.6.2.1 Tag flags always constructs that have more than one construct of the specified type with another construct of the specified type. Thus, the following model is not allowed:

```
if (...)
    begin
        if () // one branch
            begin
                ...
            end

        case () // second branch
            ...
        endcase
    end
```

VHDL

The STARC05-2.6.2.1 Tag flags process constructs that have more than one if, case, while, for, or loop statements.

Thus, the STARC05-2.6.2.1 Tag flag process constructs that have two if statements, or one if statement and one case construct, and so on.

The STARC05-2.6.2.1 Tag ignores nested constructs of same construct type. Thus, the following model is allowed:

```
library ieee;
use ieee.std_logic_1164.all

if ... then
  if ... then
    if ... then
      ...
    end if;
  end if;
end if;
```

However, the STARC05-2.6.2.1 Tag flags process constructs that have construct of one type nested in another construct of different type. Thus, the following model is not allowed as a case construct is nested inside an if statement:

```
if ... then
  if ... then
    ...
  end if;

  case .... is
    ...
  end case;
end if;
```

Message Details

Verilog

The following message appears at the first line of an always construct that has more than one construct of specified types:

Multiple statements (if/case/while/for/forever/repeat) in a single always block

STARCO5 Tags

VHDL

The following message appears at the first line of a process block that has more than one if, case, while, for, or loop statements:

Multiple statements (if/case/while/for/loop) in a single process block

Severity

Mandatory

STARC05-2.7.1.3a

An "if" clause must have a corresponding "else" clause in describing combinational circuits. (Verilog)
An IF clause must have a corresponding ELSE clause in combinational circuits (VHDL)

Description

The STARC05-2.7.1.3a Tag flags if statements in combinational circuits that do not have corresponding else statements.

The Tag also reports violation when an `else` statement corresponding to an `if` statement is empty.

The else statement covers conditions that are not met by any of the preceding if or else if statements. A missing else statement may lead to an inferred latch. Thus, to ensure that the resulting circuit is always combinational, the if statements must have the corresponding else statements or all the target signals must have a default initialization through an assignment statement.

By default, the *STARC05-2.7.1.3a* Tag does not report violation for the signals that are assigned inside both case block and if condition. In order to consider a signal as assigned inside case block, signal should be assigned a value in each defined branch. Set the value of the [check_conditional_assignment](#) parameter to yes to report violation for such signals.

NOTE: *The STARC05-2.7.1.3a Tag supports generate-for block.*

Message Details

Verilog

The following message appears at the location of an if statement that does not have a corresponding else statement in a combinational block:

"if" statement does not have a corresponding "else" statement in combinational block

VHDL

The following message appears at the location of an if statement that does not have a corresponding else statement in a combinational block:

IF statement does not have a corresponding ELSE statement in

combinational block

Severity

Mandatory

Example

Consider the following example:

```
module test (ena, enb, tempA, tempB);  
input ena, enb;  
output tempA, tempB;  
  
reg tempA, tempB;  
  
always @(*)  
begin  
    case(ena)  
        00 : tempA = 0 ;  
        01 : tempA = 1 ;  
    endcase  
  
    if (enb)begin  
        tempA = tempA + 6 ;  
    end  
end  
endmodule
```

In the above example, by default, the *STARCO5-2.7.1.3a* Tag does not report violation for the signal, `tempA`, that is assigned inside both `case` block and `if` condition. Set the value of the [check_conditional_assignment](#) parameter to `yes` to report violation for `tempA`.

STARC05-2.7.2.1

**Common part of conditional expressions should be prioritized.
(Verilog)
Common part of conditional expressions must be prioritized
(VHDL)**

Description

Verilog

The STARC05-2.7.2.1 Tag flags behavioral constructs where a conditional expression has common logic with other conditional expressions.

The STARC05-2.7.2.1 Tag flags the following cases:

- A nested if statement where one if condition expression has common logic with the condition expression in one or more subsequent if conditions.
- An if-else-if construct where the condition expression in each of the conditional branches has the same common sub-expression.

In both cases, it is recommended to separate the common part in the expressions and include them as another higher-priority if statement.

To conclude equality of two sub-expressions, the operators must be exactly the same and the operands must be specified in the same manner. For example, if a signal `x` is aliased to another signal `y` using a continuous assignment, the usage of signal `x` in one control expression is not considered same as usage of signal `y` in another control expression.

Conditional expressions with common logic generate duplicate circuitry that can be minimized by prioritizing the common part.

VHDL

The STARC05-2.7.2.1 Tag flags if-elsif-else construct branches that have common conditional logic expression or sub-expression as a following branch.

The STARC05-2.7.2.1 Tag flags a set of consecutive if-elsif-else construct branches if the same expression or sub-expression is used in all branch conditional expressions and only one of AND operator or NOR operator is used in all branch conditional expressions. The common expression or sub-expression can be at any level in the branch conditional expression.

Consider the following example where `expr1` is used in first two branch

conditions and both branches use the AND operator:

```
IF (expr1 AND expr2)
...
ELSIF (expr3 AND expr1)
...
ELSE (expr4 AND expr5)
...
```

The above example can be simplified as follows:

```
IF (expr1)
  IF (expr2)
    ...
  ELSIF (expr3)
    ...
  ELSE (expr4 AND expr5)
    ...
```

The STARCO5-2.7.2.1 Tag does not flag the following examples:

```
IF (expr1 AND expr2)
...
ELSIF (expr4 AND expr3)
...
ELSE (expr1 AND expr5) -- not consecutive
...
IF (expr1 AND expr2)
...
ELSIF (expr1 NOR expr3) -- different operator
...
ELSE (expr4 AND expr5)
...
```

Conditional expressions with common logic generate duplicate circuitry that can be minimized by prioritizing the common part.

Message Details

Verilog

The following message appears at the first line of a behavioral construct

where a conditional expression has common logic *<expr>* with other conditional expressions:

Common logic '*<expr>*' in multiple conditional expressions of '*<construct-type>*' construct

Where *<construct-type>* can be *nested-if* or *if-else-if*.

VHDL

The following message appears at the location of an *if-elsif-else* construct branch that has a common condition expression or sub-expression *<expr>* as one of the following branches:

Common sub-expression '*<expr>*' in multiple branches of an *if-elsif-else* construct

Severity

Recommended

STARC05-2.7.3.1a

Nesting in if-else constructs should not be deeper than the number of levels as specified by parameter ifelse_nestlevel_reco (default 7).

Description

The STARC05-2.7.3.1a Tag flags very deep nested if-else constructs.

By default, the STARC05-2.7.3.1a Tag flags those nested if-else constructs that are more than seven levels deep. Use the [starc2005_ifelse_nestlevel_reco](#) Tag parameter to specify a different maximum allowed depth.

Association of an if statement with its corresponding else statement becomes difficult to understand for very deep if-else constructs.

Message Details

The following message appears at the location of an if-else construct that has more levels than the value of the `ifelse_nestlevel_reco` Tag parameter:

```
More than $starc2005_ifelse_nestlevel_reco nesting levels in  
the if-else construct
```

Severity

Recommended3

Examples

Consider the following example where the level of if-else construct is one:

```
always @(in1)
  if(in1==4'd0)
    out1=8'd0;
  else if(in1==4'd1)
    out1=8'd1;
  else out1=8'd8;
```

Consider another example where the level is two:

```
always @(in1)
```

```
begin
  if (RST)
    begin
      if (IP1)
        begin
          out1=8'd0;
        end
      end
    end
  end
end
```

Association of an if statement with its corresponding else statement becomes difficult to understand for very deep if-else constructs.

STARC05-2.7.3.1b

**Where feasible case statements should be used rather than 'if-else' statements, if performance is important. (Verilog)
Use "case" statements instead of if/else statements, where feasible, if performance is important. (VHDL)**

Description

Verilog

The STARC05-2.7.3.1b Tag flags if..else constructs with large number of branches that test the same signal in all branches.

By default, the STARC05-2.7.3.1b Tag flags those if..else constructs with more than five branches that test the same signal in all branches. Use the [starc2005_ifelse_branchlevel_reco](#) Tag parameter to specify a different limit on the number of branches.

If the condition expression of if..else constructs test the same signal more than n times, the nesting is assumed to be done n number of times. A signal is assumed tested if in a comparison operation ($=$, $<=$, $>=$, $!=$, and so on), the signal and only the signal appears on the LHS of the operands.

A chain of if..else statements on the same test signal will infer a priority encoder. The same statement re-written as a case statement will infer a mux, which is likely to be faster.

For example, consider:

```
if      ( x == 2'b00 ) y = a;
else if ( x == 2'b01 ) y = b;
else if ( x == 2'b11 ) y = c;
else           y = 0;
```

The above example rewritten as follows will implement faster logic:

```
case (x)
  2'b00: y = a;
  2'b01: y = b;
  2'b11: y = c;
  default: y = 0;
endcase
```

VHDL

The STARCO5-2.7.3.1b Tag flags `if..elsif` constructs with large number of branches that test the same signal in all branches.

By default, the STARCO5-2.7.3.1b Tag flags those `if..elsif` constructs with more than five branches that test the same signal in all branches. Use the [starc2005_ifelse_branchlevel_reco](#) Tag parameter to specify a different limit on the number of branches.

If the condition expression of `if..elsif` constructs test the same signal more than n times, the nesting is assumed to be done n number of times. A signal is assumed tested if in a comparison operation (`==`, `<=`, `>=`, `!=`, and so on), the signal and only the signal appears on the LHS of the operands.

A chain of `if..elsif` statements on the same test signal will infer a priority encoder. The same statement re-written as a case statement will infer a mux, which is likely to be faster.

For example, consider:

```
library ieee;
use ieee.std_logic_1164.all

if ( x = "00" ) then y <= a;
  elsif ( x = "01" ) then y <= b;
  elsif ( x = "11" ) then y <= c;
  else y <= '0';
end if;
```

The above example rewritten as follows will implement faster logic:

```
case x is
  when "00" => y <= a;
  when "01" => y <= b;
  when "11" => y <= c;
  when others => y <= 0;
end case;
```

Message Details

Verilog

The following message appears at the location of an `if-else` construct

STARCO5 Tags

that is more than five levels deep and tests the same signal in all branches:
Case statement is recommended over deeply nested 'if' statement

VHDL

The following message appears at the location of an `if-elsif` construct that is more than five levels deep and tests the same signal `<sig-name>` in all `<num>` branches:

Case Statement preferred instead of if-then-else. Condition Clause Variable '`<sig-name>`' is being used `<num>` times

Severity

Recommended3

STARC05-2.7.3.1c

Chain of if...else if constructs must not be longer than number of levels as specified by parameter `starc2005_ifelse_chainlevel_reco` (default 7). (Verilog)
The chain of 'if-elsif' construct must not be longer than number of levels specified by parameter `starc2005_ifelse_chainlevel_reco` (default 7). (VHDL)

Description

Verilog

The STARC05-2.7.3.1c Tag flags very long if..else..if constructs.

By default, the STARC05-2.7.3.1c Tag flags those if..else..if constructs that are longer than five levels. Use the [starc2005_ifelse_chainlevel_reco](#) Tag parameter to specify a different maximum length.

The length of a if..else..if construct is assumed to be the number of if statements. For example, the following if..else..if construct is three levels long:

```
if (...)
else if (...)
else if (...)
else ...
```

VHDL

The STARC05-2.7.3.1c Tag flags very long if..else..if constructs.

By default, the STARC05-2.7.3.1c Tag flags those if..else..if constructs that are longer than five levels. Use the [starc2005_ifelse_chainlevel_reco](#) Tag parameter to specify a different maximum length.

The length of a if..elsif construct is assumed to be the number of if or elsif statements in the construct. For example, the following if..elsif construct is three levels long:

```
if (...) then ...;
elsif (...) then ...;
elsif (...) then ...;
else ..;
```


Message Details

Verilog

The following message appears at the location of an `if..else..if` construct that has more than `$starc2005_ifelse_chainlevel_reco` if statements:

Chain of 'if...else if' construct must not be more than
`$starc2005_ifelse_chainlevel_reco` levels long

VHDL

The following message appears at the location of an `if.elsif` construct that has more than `$starc2005_ifelse_chainlevel_reco` if statements:

The chain of 'if-elsif' construct must not be more than
`$starc2005_ifelse_chainlevel_reco` levels long

Severity

Recommended3

STARC05-2.7.3.4

Nested if-statements should be merged

Description

Verilog

The STARC05-2.7.3.4 Tag flags deeply nested if constructs.

The STARC05-2.7.3.4 Tag flags nested if constructs where a child if statement can be merged with its parent if or else if statement (to generate a combined if statement).

Deeply nested if statements can decrease readability of the RTL because associating an else statement to the corresponding if statement can be quite un-intuitive. Reducing nesting depth by merging if statements wherever possible can be an effective method of decreasing nesting depth and increasing comprehensibility.

VHDL

The STARC05-2.7.3.4 Tag flags deeply nested if-[elsif]-[else] constructs.

The STARC05-2.7.3.4 Tag flags if-[elsif]-[else] constructs where an if statement can be merged with its parent if statement (to generate a combined if statement) or its parent else statement (to generate a combined elsif statement).

The STARC05-2.7.3.4 Tag does not flag if-[elsif]-[else] constructs where merging produces complex branch condition. For example, an if statement of an if-else construct that is embedded inside another if statement is not flagged.

Deeply nested if-[elsif]-[else] constructs can decrease readability of the RTL as it becomes progressively difficult to associate else statement with its corresponding if statement. Merging branched or deeply nested if-[elsif]-[else] constructs is an effective method of decreasing nesting depth and increasing comprehensibility.

Message Details

Verilog

The following message appears at the location of an if statement that can be merged with its parent if or else if statement to reduce the nesting level:

Merge "if" condition into the parent "if/else if" branch to

reduce nesting depth

VHDL

The following message appears at the location of an if statement that can be merged with its parent if, else, or elsif statement to reduce the nesting level:

Merge "if" condition into the parent "if" branch to reduce nesting depth

Merge "if" condition into the parent "else" branch to reduce nesting depth

Merge "if" condition into the parent "elsif" branch to reduce nesting depth

Severity

Recommended

Verilog Examples

In the following example of the nested if construct, the second if statement can be merged with the parent if statement:

```
...
if (a = '1') then
    if (b = '1') then
        ...
    end if;
end if;
...
```

VHDL Examples

Consider the following example of a deeply nested if-else construct:

```
library ieee;
use ieee.std_logic_1164.all

if(RST = '1') then
    if(CLK'event) then
        if(CLK = '0') then
            Q <= D;
```

```
        end if;
        if(CLK = '1') then
            Q <= D;
        else
            Q <= '1';
        end if;
    end if;
    if(D = '1') then
        Q <= '0';
    end if;
end if;
```

The second and the last if statements can be merged with the parent if statement to reduce the nesting levels as follows:

```
if(RST = '1' and CLK'event ) then
    if( CLK = '0' ) then
        Q <= D;
    end if;
    if( CLK = '1' ) then
        Q <= D;
    else
        Q <= '1';
    end if;
end if;

if( RST = '1' and D = '1') then
    Q <= '0';
end if;
```

STARC05-2.7.4.2

Attach begin-end to "if" statements.

Description

The STARC05-2.7.4.2 Tag flags if-if statements without use of else and begin-end statements.

The STARC05-2.7.4.2 Tag reports violation only for sequential FF description and will not violate for combination description. Also, this Tag will not flag when begin-end block is specified for if/else statement and when there is no ambiguity in the else statement specification corresponding to each if statement.

Message Details

The following message appears when begin-end statements are not specified for if statements in the module `<module-name>`:

```
Attach begin-end to if statements in module <module-name>
```

Severity

Reference

STARC05-2.7.4.3

Do not use fork-join constructs. (Verilog)

Description

The STARC05-2.7.4.3 Tag flags fork-join constructs in the design.
fork-join constructs are not synthesizable as they have no physical significance.

Message Details

The following message appears at the location where a fork-join construct is encountered:

fork-join construct used

Severity

Mandatory

STARC05-2.8.1.3

**case constructs should not have overlapping clause conditions.
(Verilog)**

Description

The STARC05-2.8.1.3 Tag flags overlapping clauses in case constructs.

While overlapping clauses are allowed as only the first found clause meeting the condition value is executed, synthesis tools create a complicated netlist with similar priority values as the if-else constructs.

The STARC05-2.8.1.3 Tag does not report a violation for the following:

- x and z in case
- x in casez
- z in casex

NOTE: *The Tag supports the multi-line functionality.*

By default, the Tag reports violations for the overlapping case constructs. Set the value of the [ignore_same_case_item](#) parameter to yes to ignore violations for the same case items.

Message Details

The following message appears at the location of a case clause condition that overlaps previously found case clause condition for condition value `<cond-value>`:

overlapping case clause condition (<cond-value>) used

Severity

Recommended1

Example

Consider the following example:

```
module test(in1,sel,out1,out2);
  input [3:0]in1, sel;
  output out1,out2;
  reg out1,out2;
```

```
always @(sel)
  case(sel)
    5'bx0000 : out1 = in1[0];
    5'bx0000 : out1 = in1[1];
    5'b01000 : out1 = in1[2];
    5'b01000 : out1 = in1[3];
    default  : out1 = 1'bx;
  endcase

always @(sel)
  casex(sel)
    5'bx0000 : out1 = in1[0];
    5'bx0000 : out1 = in1[1];
    5'b01000 : out1 = in1[2];
    5'b01000 : out1 = in1[3];
    default  : out1 = 1'bx;
  endcase
```

In the above example, the Tag does not flag a violation for 5'bx0000 because it is used in a case statement.

The Tag reports violation for the following:

- 5'b01000 because it is used in a case statement
- 5'bx0000 because it is used in a casex statement
- 5'b01000 because it is used in a casex statement

STARC05-2.8.1.4

**case constructs must have the default clause. (Verilog)
Case statements and selected signal assignment statements must
always have OTHERS clause (VHDL)**

Description

Verilog

The STARC05-2.8.1.4 Tag flags case constructs that do not have the default clause.

By default, the STARC05-2.8.1.4 Tag does not flag case constructs without the default clause when all conditions are specified. Set the value of the [strict](#) Tag parameter to yes or STARC05-2.8.1.4 to report such case constructs.

The STARC05-2.8.1.4 Tag also flags a violation if all signals are not assigned in all branches of the case construct. By default, the STARC05-2.8.1.4 Tag does not report a violation if all cases are covered with all the signals assigned in each case statement but if all signals are not assigned in the default clause (or the default clause is missing). Set the [strict](#) Tag parameter to report such cases.

For a case construct without the default clause, the synthesis tool assumes that the previous output value is to be retained when a condition value other than those specified in the construct is received. This assumption forces the tool to infer a latch in the netlist.

Even when all possible states of the case condition signal are described, it is recommended to describe the default clause to handle the situations when the case condition signal is unknown or high impedance.

By default, the STARC05-2.8.1.4 Tag checks for only combinational always block. It does not check for sequential circuits because no latches are inferred. Instead, enabled flip-flops are formed. Set the [starc_check_sequential_block](#) Tag parameter to check sequential always block.

By default, the [report_onehot_case](#) parameter is set to yes and the STARC05-2.8.1.4 tag reports case constructs if all clauses of conditions are not described in the case statement without the default clause. Set the value of the [report_onehot_case](#) parameter to no to not report a violation for one-hot case blocks without default clause.

By default the *STARCO5-2.8.1.4* tag reports violations for the unique and priority cases. Set the value of the *ignore_unique_and_priority* parameter to yes to ignore violations for the unique, unique0, and priority cases.

Use the *handle_case_select* parameter to specify the maximum width of case selector that the supported Tag should process.

NOTE: *The STARCO5-2.8.1.4 Tag ignores case constructs that have full case pragma specified.*

VHDL

The STARCO5-2.8.1.4 Tag flags case constructs and select signal assignment statements without an others clause.

The others clause must always be specified in a case construct or select signal assignment statement to handle unexpected situations even if the signal set in case constructs or select signal assignment statement covers all potential alternatives. An others clause catches subtle errors as design evolves or case selector contains unconstrained signals.

By default, the STARCO5-2.8.1.4 Tag does not flag case constructs without the default clause when all conditions are specified. Set the value of the *strict* Tag parameter to yes or STARCO5-2.8.1.4 to report such case constructs.

Message Details

Verilog

The following message appears at the starting line of a case construct that does not have the default clause:

Missing default clause in case construct, or all signals are not assigned in all branches

VHDL

The following message appears at the first line of a case construct that does not have an others clause:

others clause not found in case statement

The following message appears at the location of a select signal assignment statement that does not have an others clause:

others clause not found in selected signal assignment statement

Severity

Mandatory

Verilog Examples

Consider the following example where no assignment to out2 is done:

```
module casetest2(in1,in2,en,out1,out2);
    input [1:0]in1,in2,en;
    output[1:0]out1,out2;
    reg [1:0]out1,out2;
    always @(in1 or in2 or en)
    case(en)
        2'b00: begin
            out1 = in1;
        end
        2'b01: begin
            out1 = ~in1;
            out2 = ~in2;
        end
        2'b10: begin
            out1 = in2;
            out2 = in1;
        end
        2'b11: begin
            out1 = ~in2;
            out2 = ~in1;
        end
        default: begin
            out1 = ~in2;
            out2 = ~in1;
        end
    endcase
endmodule
```

STARC05-2.8.1.6

The case expression width should match case select expression width (Verilog)

Description

The STARC05-2.8.1.6 Tag flags case expression whose width is smaller than the case selector width as it might lead to unwanted target in case expressions.

Modify the case expressions to explicitly match the case selector in all bits. This makes behavior easier to understand and does not rely on possibly undefined behavior in handling unmatched bits.

Width of case expression will be calculated according to following methods:

- If the [nocheckoverflow](#) Tag parameter is set to `yes`, width is calculated as per LRM. However, for constants, natural width is considered.
- If the [nocheckoverflow](#) Tag parameter is set to `no`, width is calculated according to the following methods:

- ☐ For plus/minus operator, value based width is considered. For example:

```
case (sel)
(a[2:0]+b[2:0]): out=1'b1;
// Width of a[2:0]+b[2:0] will be 4 (7+7=14, 4 bits
// wide)
```

- ☐ For multiplication operator, width will be sum of operand widths. For example:

```
case (sel)
(a[2:0]*b[2:0]): out=1'b1;
// Width of a[2:0]*b[2:0] will be 6

(a1[2:0]*b1[2]): out=1'b0;
// Width of a1[2:0]*b1[2] will be 3
```

- ☐ For division operator, LHS width is considered. For example:

```
case (sel)
(a[2:0]/b[2]): out=1'b0;
// Width of a[2:0]/b[2] will be 3
```

STARC05 Tags

- ❑ For concat operator, width is calculated as a sum of all the operands. For example:

```
case(sel)
{1'b0,a[2:0]}: out=1'b1;
// Width=3 bits (leading 0 ignored)

{1'b1,a[2:0]}: out=1'b0;
// Width=4 bits (1+3 bits)
```

Also, set the value of the [use_lrm_width](#) parameter to `yes` to consider LRM width, which is 32 bits.

The STARC05-2.8.1.6 Tag does not flag in the following scenarios:

- When case select is a static expression
- When case expression is a constant (integer/macro), and its width is smaller than case select width

Message Details

The following message appears at the location of a case clause label whose width (`<widthc>`) is not same as the case construct selector width (`<widths>`):

Case label width (`<widthc>`) does not match selector width (`<widths>`)

Severity

Caution

Verilog Examples

Consider the following example where the clause `sel` of the case construct is of width 4 bits and the case clause labels have same or different bit-widths:

```
module test(in, out, sel, data);
  input [3:0] in, data, sel;
  output [3:0]out;
  reg [3:0]out;

  parameter [3:0]S = 23;
```

```
parameter [2:0]P = 10;
parameter [15:0]Q = 16'h1101;
parameter R = 2;

always @(in or sel)
begin
  case (sel)
    S : out = 4'b0000;
    P : out = 4'b0000;
    Q : out = 4'b0001;
    R : out = 4'b0011;
    in: out = 4'b0011;
    4'b1010 : out = 4'b0011;
    3'b110 : out = 4'b0011;
    default out = 0;
  endcase
end
endmodule
```

The case clause labels (S, in, and 4'b1010) have the same width (4 bits) as the case construct selector sel.

The case clause labels P (3 bits), Q (16 bits), R (32 bits being an integer), and 3'b110 (3 bits) do not have the same bit-width as the case construct selector sel. Hence, SpyGlass generates the following messages respectively:

Case label width (3) does not match selector width (4)

Case label width (16) does not match selector width (4)

Case label width (32) does not match selector width (4)

Case label width (3) does not match selector width (4)

STARC05-2.8.2.1

if constructs should be used instead of large case constructs.

Description

The STARC05-2.8.2.1 Tag flags large case constructs.

For Verilog designs, the STARC05-2.8.2.1 Tag flags the following types of case constructs:

- case constructs where case construct condition bit-width exceeds the value of the [starc2005_maxInputSize](#) Tag parameter (default value 20).
- case constructs that do not violate the above condition but have (((case construct condition bit-width > value of the [starc2005_maxCondBitWidth](#) Tag parameter [default value 4]) OR (number of case clauses > value of the [starc2005_maxCaseClause](#) Tag parameter [default value 5])) AND (case construct output bit-width > value of the [starc2005_maxOutBitWidth](#) Tag parameter [default value 12])).

Nested case constructs are checked on a construct-by-construct basis. Thus, a design containing a case construct with 10-bit condition nested inside another case construct with 10-bit condition does not violate the STARC05-2.8.2.1 Tag as each individual case construct does not individually violate the STARC05-2.8.2.1 Tag.

The size of case constructs depends on the size of case condition, size of case output, and number of case clauses. Synthesis of large case constructs is time-consuming and the generated circuit is not efficient as the synthesized area and synthesis time increase at exponential rates after the size of case construct exceeds beyond a limit.

It is recommended to use if constructs instead of large case constructs as the synthesis of equivalent if constructs is more efficient.

By default, the STARC05-2.8.2.1 tag does not report violation for casez constructs. Set the value of the [starc_report_casez](#) parameter to report violation for casez constructs as well.

NOTE: *The STARC05-2.8.2.1 Tag supports generate-for block.*

NOTE: *Records and multi-dimensional arrays are not handled by the STARC05-2.8.2.1 Tag.*

By default, the value of the [starc_use_natural_width](#) parameter is set to no. Set the value of the parameter to yes to enable the STARC05-2.8.2.1 tag to consider static width of constants and parameters.

By default, the STARC05-2.8.2.1 tag reports different violations for multiple hierarchies. Set the value of the [report_one_hier](#) parameter to report one violation for multiple hierarchies.

Message Details

The following message appears at the first line of a case construct that exceeds the recommended size:

```
case construct with condition bit-width <num1> bits, case  
output bit-width <num2> bits and <num3> clauses exceeds  
recommended size [Hierarchy: '<hier-path>']
```

Where, *<num1>* bits is the bit-width of the case construct condition, *<num2>* bits is the bit-width of the case construct output, *<num3>* is the number of case clauses, and *<hier-path>* is the hierarchical name of the instance (in *<du-name>:<inst-name>@<master-du-name>* format) where the Tag-violating construct is encountered.

Severity

Recommended3

STARC05-2.8.2.2

Item count should be within the value specified by parameter 'starc2005_maxTableSize'.

Description

The STARC05-2.8.2.2 Tag flags case constructs with very large item count.

By default, the STARC05-2.8.2.2 Tag flags case constructs where the total number of case clauses exceeds 200. Use the [starc2005_maxTableSize](#) Tag parameter to specify a different limit.

Message Details

Verilog

The following message appears at the first line of a case construct that exceeds the recommended size:

```
item count should be within $starc2005_maxTableSize
```

VHDL

The following message appears at the first line of a case construct that exceeds the recommended size:

```
Number of case clauses should be less then  
$starc2005_maxTableSize
```

Severity

Recommended3

STARC05-2.8.3.1

default clause output in case constructs should not be a constant value. (Verilog)
Constant values should not be assigned in OTHERS clause of case constructs (VHDL)

Description

Verilog

The STARC05-2.8.3.1 Tag flags default clauses of case constructs that have fixed values.

It is recommended to assign X as the default value in a case construct if you do not care about the default clause. The logic created for such case constructs is smaller and better optimized than the case constructs with fixed value in the default clause.

VHDL

The STARC05-2.8.3.1 Tag flags constant outputs generated by others clauses of case constructs.

others clause of a case construct is executed if none of the case clauses match the case select expression. It is recommended to assign X to the output of the others clause in a case construct if you do not care about the value assigned in the others clause. The logic created for such case constructs is smaller and better optimized than the case constructs with fixed value in the others clause.

NOTE: *The STARC05-2.8.3.1 Tag does not flag a message if the others clause is assigning a constant value to a bit-select or a part-select.*

Message Details

Verilog

The following message appears at the location of the default clause of a case construct of output `<output-name>` in hierarchy `<hier-path>` when the output is assigned a constant value:

Constant value assigned to '`<output-name>`' in default clause of case construct. [Hierarchy: '`<hier-path>`']

VHDL

The following message appears at the location of an others clause if the

STARCO5 Tags

clause assigns a constant *<value>* to the signal *<sig-name>*:
Constant value '*<value>*' assigned to '*<sig-name>*' in OTHERS
clause of case construct

Severity

Recommended

STARC05-2.8.3.2

Either default clause does not exist or 'X' is not assigned in the default clause. (Verilog)
Either others clause does not exist or 'X' is not assigned in the others clause. (VHDL)

Description

Verilog

The STARC05-2.8.3.2 Tag flags when all the conditions of a case statement exist without a default clause or when X (unknown) is not assigned in the default clause.

The STARC05-2.8.3.2 Tag checks case clause only when all the conditions (that is, full case) are specified.

This Tag flags only when a signal is assigned some value in all the case clauses, and either default clause does not exist or the above signal is not assigned X in the default clause.

By default, the Tag reports violations for the overlapping case constructs. Set the value of the [ignore_same_case_item](#) parameter to yes to ignore violations for the same case items.

VHDL

The STARC05-2.8.3.2 Tag flags when all the conditions of a case statement exist without the others clause or when X (unknown) is not assigned in the others clause.

The STARC05-2.8.3.2 Tag checks case clause only when all the conditions (that is, full case) are specified.

This Tag flags only when a signal is assigned some value in all the case clauses, and either others clause does not exist or the above signal is not assigned X in the others clause.

Message Details

Verilog

The following message appears when all the conditions *<full-case>* of a case statement *<case-sel-expr>* does not either X assignment to or default clause for the violating signal *<sig-name>*:

`<full-case> '<case-sel-expr>' does not have <X assign| default clause> signal '<sig-name>'`

VHDL

The following message appears when all the conditions of a case statement `<case-sel-expr>` does not either X assignment to or others clause for the violating signal `<sig-name>`:

Full Case '`<case-sel-expr>`' does not have `<X assign | others clause>` signal '`<sig-name>`'

Severity

Reference

STARC05-2.8.3.6

Do not use a signal to which don't care is assigned for selection expression of case statements that do not assign 'X' in default clause. (Verilog)
Do not use a signal to which don't care is assigned for selection expression of case statements that do not assign 'X' in 'others' clause. (VHDL)

Description

The STARC05-2.8.3.6 Tag flags case constructs where the select expression uses a signal that is assigned don't care and the 'others' clause does not assign don't care to a signal.

Do not use signals to which don't care conditions are assigned, for case construct selection expressions if the default clause does not assign don't care to signals. Because the don't care condition does not propagate without assigned in the 'others' clause. Thus, the pre- and post-synthesis simulation results may differ.

NOTE: *The STARC05-2.8.3.6 Tag supports generate-for block.*

Message Details

The following message appears at the location of a case construct clause that uses signal *<sig-name>* that is assigned don't care, in the select expression but the 'others' clause does not assign don't care to signals:

Don't care signal '*<sig-name>*' used in the selection expression of a case statement

Severity

Recommended1

STARC05-2.8.3.7

Do not use a signal to which don't care is assigned for selection expression of casex statements. (Verilog)

Description

The STARC05-2.8.3.7 Tag flags casex constructs where the select expressions uses a signal to which don't care value is assigned.

Do not use signals, to which don't care conditions are assigned for selection expressions. Because, 'x' may not be able to propagate and simulation results may differ between RTL and gate level.

Message Details

The following message appears at the location of a casex construct clause that uses signal *<sig-name>* that is assigned don't care value in the select expression:

Don't care signal '*<sig-name>*' used in the selection expression of a casex statements

Severity

Recommended1

STARC05-2.8.4.1a

casex or casez constructs must contain a condition with don't care (Verilog)

Description

The STARC05-2.8.4.1a Tag flags casex and casez constructs that have no condition having don't care.

Without don't care, the use of casex or casez constructs is redundant and simple case construct should be used in its place.

Message Details

The following message appears at the location of a casex or a casez construct that does not any condition with don't care:

None of the condition of '<type>' statement contains a don't care

Where *<type>* can be casex or casez.

Severity

Caution

STARC05-2.8.4.1b

Coding style of don't-care condition in casex or casez branches is not proper (Verilog)

Description

The STARC05-2.8.4.1b Tag flags casex and casez constructs using improper coding style of don't-care condition in branches.

It is recommended that the pattern of don't-care condition in casex or casez construct clauses should be monotonically decreased, which is very helpful when considering overlapped condition.

Following two patterns of the triangle is good design practice:

- Right upper to lower triangle as in the following example:

```
casex(in)
  5'b1????:
    ...
  5'b01???:
    ...
  5'b001?:
    ...
  default:
    ...
endcase
```

- Left upper to lower triangle as in the following example

```
casex(in)
  5'b???1:
    ...
  5'b???10:
    ...
  5'b???100:
    ...
  default:
    ...
endcase
```

The STARC05-2.8.4.1b Tag flags a message if any of following sub-Tags is not satisfied:

1. Length of don't-care string is monotonically decreased.
2. No don't-care only clause exists.
3. Right/left end of don't-care string is same bit-position for all clauses except the default clause.
4. Check for triangle only if there are two or more adjacent lines of run.

If the 4th sub-Tag is not true, the STARC05-2.8.4.1b Tag is not checked.

NOTE: *The STARC05-2.8.4.1b Tag does not work for case constructs with multiple strings of don't-care conditions in any of the case clauses. In such cases, the [STARC05-2.8.4.4](#) is applicable.*

Following a proper coding style of the don't-care conditions in case constructs helps effective verification with organized coding pattern because random don't-care pattern would lead test pattern development to be more difficult.

Message Details

The following message appears at the location of a casex or a casez construct that does not follow the proper don't care condition coding style:

Improper coding style of dont-care condition in casex or casez branches

Severity

Caution

STARC05-2.8.4.3

Design should not use casex or casez constructs. (Verilog)

Description

The STARC05-2.8.4.3 Tag flags casex and casez constructs.

Use of casex or casez statements may increase chances of overlapping conditions and may deteriorate circuit quality. Such constructs may also lead to mismatches between pre- and post-synthesis simulation.

By default, the STARC05-2.8.4.3 Tag reports violation for both casex and casez constructs. You can set the value of the [check_case_keyword](#) parameter to either casex or casez to report violation for either casex or casez constructs, respectively.

Message Details

The following message appears at the starting line of a casex construct:

Avoid using <case_type> constructs

where,

the value of <case_type> is either casex or casez.

Severity

Recommended

STARC05-2.8.4.4

casex or casez constructs must not have complex clause conditions. (Verilog)

Description

The STARC05-2.8.4.4 Tag flags complex casex or casez constructs.

The STARC05-2.8.4.4 Tag flags casex and casez constructs that have clause conditions with multiple bit-slices set to Xs, Zs, or ?s.

A casex or casez construct where all clause conditions contain a maximum of one substring of Xs, Zs, or ?s is not flagged. For example, a case clause condition 11xx1100 is not flagged as it has only one substring of Xs. The case clause condition 11xx1xx0 is flagged as it has more than one substring of Xs.

Message Details

The following message appears at the location where a complex casex or casez construct is encountered:

Complex '<type>' clause condition used

Where <type> can be casex or casez.

Severity

Recommended1

STARC05-2.8.5.2

Do not use fixed values in case construct condition. (Verilog)

Description

The STARC05-2.8.5.2 Tag flags case constructs where the case construct condition is a fixed value.

case constructs with fixed value as its condition are essentially complex if constructs. It is recommended to use if constructs in such cases as they are easier to understand.

NOTE: *This Tag considers the return value of functions as non-constant even for functions returning constant values in all branches while evaluating the case select expression for fixed value.*

Message Details

The following message appears at the location of a case construct that has a fixed value as its condition:

Fixed value used as case construct condition

Severity

Recommended1

STARC05-2.8.5.3

Do not use variables or expressions as case clause conditions. (Verilog)

Description

The STARC05-2.8.5.3 Tag flags case construct clauses that have variables or expressions as clause conditions.

case constructs with variables or expressions as clause conditions are essentially complex if constructs. It is recommended to use if constructs in such cases as they are easier to understand.

Consider the following case construct that uses expressions as clause conditions:

```
case (1)
  expr1: ...
  expr2: ...
  ...
end case
```

This construct is functionally equivalent to the following if construct that is easier to understand:

```
if (expr1)
  ...
else if (expr2)
  ...
else if ...
```

Parameter(s)

- [*ignore_vector_bitselect*](#): By default, the parameter is set to `no`. Set the value of this parameter to `yes` to not report violations for the following cases:

- ☐ Case expression is 1 bit constant (for example, `1'b1`)
- ☐ Case item expression is bit select of a vector signal (for example, `state[0]`)

NOTE: *Index should be constant.*

Message Details

The following message appears at the location where a variable or an expression is used in the case construct clause condition:

variables or expressions used in the case clause condition

Severity

Mandatory

STARC05-2.8.5.4

Logical, arithmetic, bit wise or reduction operations should not be used in case construct condition. (Verilog)

Description

The STARC05-2.8.5.4 Tag flags case constructs that have logical, arithmetic, bit-wise, or reduction operations in their select condition.

It is recommended to use only simple signal names as the case construct select condition.

Message Details

The following message appears at the location of a case construct that has a logical, arithmetic, bit-wise, or reduction operation in its select condition:

Logical, arithmetic or bitwise operation used in case construct condition

Severity

Recommended

STARC05-2.9.1.1

"for" constructs must be used only for simple repeating statements. (Verilog)
FOR constructs must be used only for simple repeating statements (VHDL)

Description

Verilog

The STARC05-2.9.1.1 Tag flags complex for constructs.

The STARC05-2.9.1.1 Tag flags for constructs that have the following situations:

- The value of a scalar signal is being calculated as an expression using the value of the signal itself. For example:

```
for (i = 0; i <= 3; i = i + 1)
  x = x & a[i];
  c = expr1(a, b);
  a = expr2(c, d);

for (i = 0; i <= 3; i = i + 1) begin
  c = a & b[i];
  a = c ^ d;
end
```

- Bit-select of a vector variable indexed on the loop variable is being read when another bit-select of the same vector variable indexed on the loop variable is being assigned in the same for construct.

A bit-select of a vector variable is assumed to be read if the bit-select is used as part of the RHS expression of any assignment statement or in the index of a vector signal on LHS of any assignment statement. A bit-select of a vector variable is assumed to be set if the bit-select appears on LHS of any assignment statement as an isolated signal or as part of a concatenation or part-select of the vector signal.

For example, the STARC05-2.9.1.1 flags a message if bit-select `sig1[expr1(i)]` is read and `sig1[expr2(i)]` is set within the same for construct where `expr1(i)` is not equal to `expr2(i)`.

NOTE: *Expressions involving loop variables are considered to be equal only if the*

expressions are specified in an identical manner.

- The value of a scalar signal is being calculated as an expression using another signal whose value in turn is dependent on the value of the original signal as in the following example:

```
c = expr1(a, b);  
a = expr2(c, d);
```

Here, signal `c` is calculated as an expression of signal `a` which in turn is calculated as an expression of signal `c`.

for constructs where one iteration depends on the result of a previous iteration are synthesized to a long cascading logic and may have timing issues.

VHDL

The STARC05-2.9.1.1 Tag flags complex for constructs, which generates priorities.

The STARC05-2.9.1.1 Tag flags following types of for constructs:

- Constructs where the value of a scalar signal is being calculated as an expression using the value of the signal itself as in the following example:

```
for i in 0 to 7 loop  
    x <= x and a(i);  
end loop;  
for i in 0 to 7 loop  
    z <= x and a(i);  
    x <= z or w;  
end loop;
```

- Constructs where a bit-select of a vector variable indexed on the loop variable is being read when another bit-select of the same vector variable indexed on the loop variable is being assigned in the same for construct.

A bit-select of a vector variable indexed on the loop variable is being read when another bit-select of the same vector variable indexed on the loop variable is being assigned in the same for construct.

A bit-select of a vector variable is assumed to be read if the bit-select is used as part of an expression on RHS of any statement, in conditional expression or in the index of a vector signal on LHS of any statement. A bit-select of a vector variable is assumed to be set if the bit-select appears on LHS of any statement as an isolated signal.

For example, the STARCO5-2.9.1.1 Tag flags a message if bit-select `sig_a[expr1(i)]` is read and `sig_a[expr2(i)]` is set within the same for construct where `expr1(i)` is not equal to `expr2(i)`.

NOTE: *Expressions involving loop variables are considered to be equal only if the expressions are specified in an identical manner.*

- The value of a scalar signal is calculated as an expression using another signal whose value in turn is dependent on the value of the original signal as in the following example:

```
sig_c = expr1(sig_a, sig_b);
sig_a = expr2(sig_a, sig_d);
```

Here, signal c is calculated as an expression of signal sig_a which in turn is calculated as an expression of signal sig_d.

for constructs where one iteration depends on the result of a previous iteration are slower.

Message Details

The following message appears at the location where a complex for construct is encountered:

Verilog

Complex for construct used

VHDL

Complex for-loop used

Severity

Mandatory

STARC05-2.9.1.2a

**Loop variable of "for" construct must have constant initial value.
(Verilog)**

Description

The STARC05-2.9.1.2a Tag flags for constructs where the loop variable's initial value is not a constant.

The initial value of the loop variable of `for` constructs must be a constant value or an expression involving constant values only.

By default, the Tag does not report violation when input or inout of a function/task is used as initial condition of `for` construct. However, when the *strict* parameter is set, violation is reported for the same when corresponding actual argument of the function/task is non-constant.

NOTE: *Parameters and 'define macros are also supported as constants.*

Message Details

The following message appears at the location of a `for` construct where the loop variable does not have a constant initial value:

```
for loop variable has non-constant initial value
```

Severity

Mandatory

STARC05-2.9.1.2b

Terminating condition of "for" constructs must be a constant. (Verilog)

Description

The *STARC05-2.9.1.2b* Tag reports `for` constructs with non-constant terminating condition.

The termination expression of a `for` construct must be a simple comparison to a constant value or an expression involving constant values only.

By default, Tag does not report violation when input or inout of a function or task is used as termination condition of `for` construct. However, when the *strict* parameter is set, violation is reported for the same when corresponding actual argument of the function or task is non-constant.

NOTE: *Parameters and 'define macros are also supported as constants.*

Message Details

The following message appears at the location of a `for` construct with non-constant terminating condition:

for construct has non-constant terminating condition

Severity

Mandatory

Examples

Example 1

The *STARC05-2.9.1.2b* Tag reports a violation, by default, for the `for` loop appearing in the following example:

```
module mod_2_9_1_2b_S_Test1 (out, in, enable, end_cond);  
  
    output [3:0] out;  
    input [3:0] in;  
    input enable, end_cond;
```

```
reg [3:0] out;
integer i;

always @ (enable or in or end_cond)
    for (i = 0; i <= end_cond; i = i + 1)
        out[i] = in[i];

endmodule
```

Example 2

Consider the following example:

```
module tc;
integer x,y;
function integer mp_log2;
input integer value;
integer k, result;
begin
    result = 0;
    for (k = 0; 2**k <= value; k = k + 1)//violation if
strict=yes
        result = k;
        mp_log2 = result;
end
endfunction
always@(*)
    y = mp_log2(x);
endmodule
```

By default, no violation is reported in the above example because the variable `value` is an input of the function `mp_log2`. However, the *STARCO5-2.9.1.2b* Tag reports a violation if the *strict* parameter is set to *yes*.

STARC05-2.9.1.2c

The value of loop variable must not be modified inside the "for" construct. (Verilog)

Description

The STARC05-2.9.1.2c Tag flags for constructs where the loop variable is modified inside the for construct.

Such descriptions are logical errors.

Parameters and 'define macros are also supported as constants. That is, the STARC05-2.9.1.2c Tag also flags a violation, if the loop variable is modified using the parameters or 'define macros. For example, consider the following example:

```
module test(out_var, in_var, enable);
    parameter temp = 2;

    output [3:0] out_var;
    input [3:0] in_var;
    input enable;

    reg [3:0] out_var;
    integer i;

    always @ (enable or in_var)
        for (i = 0; i <= 3; i = i + 1) begin
            out_var[i] = in_var[i];
            i = temp;
        end

endmodule
```

By default, the tag reports violations when increment and decrement by 1 is done within a for-loop. Set the value of the [ignore_incr_decr_by_one](#) parameter to yes to ignore such violations.

Message Details

The following message appears at the location of a for construct where the loop variable is modified inside the loop:

for loop variable is modified within the loop

Severity

Mandatory

STARC05-2.9.1.2d

The loop termination condition must not be a constant. (Verilog)

Description

The STARC05-2.9.1.2d Tag flags for constructs where the loop termination condition evaluates to a constant.

In such cases, the for construct is either never executed or never terminated. Hence, such descriptions are logical errors.

Message Details

The following message appears at the location of a for construct where loop termination condition evaluates to a constant:

The condition in for statement is constant - the loop will either never execute or never terminate

Severity

Mandatory

STARC05-2.9.2.1

Arithmetic operations on non-constants within for-loop must not be performed

Description

Verilog

The STARC05-2.9.2.1 Tag flags arithmetic operations involving non-constants in for constructs.

The STARC05-2.9.2.1 Tag flags a message wherever a non-constant participates in any of the following types of arithmetic operations in a for construct:

Subtraction (-)	Addition (+)	Multiplication (*)
Division (/)	modulus (mod)	Shift-Left (<<)
Shift-Right (>>)		

By default, the STARC05-2.9.2.1 Tag flags all non-constants involved in arithmetic operations in for constructs. Set the [flag_only_induction_var](#) Tag parameter to flag only those non-constants that are induction variables and are involved in arithmetic operations in for constructs. In the following example, signal `b[i]` is an induction variable:

```
for (i=0;i<=10;i=i+1)
  a[i] = b[i]+1;
```

As the induction variable `b[i]` is involved in an arithmetic operation in a for construct, the STARC05-2.9.2.1 Tag flags this operation.

During synthesis, arithmetic operations with non-constants inside a for construct lead to a large area penalty because they generate replicated circuits.

The STARC05-2.9.2.1 Tag does not flag arithmetic operations on index of arrays in a for construct. For example, the Tag does not flag the following code in a for construct:

```
  a[i] = b[i+j];
```

By default, no Tag checking is done for expressions that are part of array indexes. When [strict](#) parameter is set to `yes`, array index expressions are checked and violation is reported, if the arithmetic expression contains any

non-static variable other than for-loop index.

VHDL

The STARCO5-2.9.2.1 Tag flags arithmetic operations involving non-constants in for constructs.

The STARCO5-2.9.2.1 Tag flags a message wherever a non-constant participates in any of the following types of arithmetic operations in a for construct:

Subtraction (-)	Addition (+)	Multiplication (*)
Division (/)	modulus (mod)	Remainder (rem)
Shift-Arithmetic (sla, sra)	Concatenation (&)	

By default, the STARCO5-2.9.2.1 Tag flags all non-constants involved in arithmetic operations in for constructs. Set the [flag_only_induction_var](#) Tag parameter to flag only those non-constants that are induction variables and are involved in arithmetic operations in for constructs. In the following example, signal `b[i]` is an induction variable:

```
for i in 0 to 10 loop
  a[i] <= b[i] + 1;
```

As the induction variable `b[i]` is involved in an arithmetic operation in a for construct, the STARCO5-2.9.2.1 Tag flags this operation.

During synthesis, arithmetic operations with non-constants inside a for construct lead to a large area penalty because they generate replicated circuits.

The STARCO5-2.9.2.1 Tag does not flag arithmetic operations on index of arrays in a for construct. For example, the Tag does not flag the following code in a for construct:

```
a[i] <= b[i+j];
```

By default, no Tag checking is done for expressions that are part of array indexes. When [strict](#) parameter is set to `yes`, array index expressions are checked and violation is reported, if the arithmetic expression contains any non-static variable other than for-loop index.

NOTE: *The STARCO5-2.9.2.1 Tag cannot process value propagation. In such cases, SpyGlass cannot determine whether the value is a constant or not.*

Message Details

Verilog

The following message appears at the location of an arithmetic operation in a for construct that involves a non-constant:

Arithmetic operation performed with non-constant in expression '*<expr>*' within for-loop

VHDL

The following message appears at the location of an arithmetic operation expression *<expr>* involving a non-constant in a for construct:

Arithmetic operation performed with non-constant in expression '*<expr>*' within for-loop

Severity

Mandatory

STARC05-2.9.2.2

Logical and Relational operations on non-constants within for-loop should not be performed.

Description

Verilog

The STARC05-2.9.2.2 Tag flags logical or relational operations involving non-constants in for constructs.

The STARC05-2.9.2.2 Tag flags a message wherever a non-constant participates in any of the following types of logical and relational operations in a for construct:

logical negation (!)	logical equality (==)
logical inequality (!=)	case equality (===)
case inequality (!==)	relational operation(>, >=, <, <=)
logical and (&&)	logical or ()
logical xor (^)	bit-wise negation (~)
bit-wise and (&)	bit-wise or ()

By default, the STARC05-2.9.2.2 Tag flags all non-constants involved in logical or relational operations in for constructs. Set the [flag_only_induction_var](#) Tag parameter to flag only those non-constants that are induction variables and are involved in logical or relational operations in for constructs. In the following example, signal `b[i]` is an induction variable:

```
for (i=0;i<=10;i=i+1)
  if (b[i] || 1)
    ...
```

As the induction variable `b[i]` is involved in a logical or relational operation in a for construct, the STARC05-2.9.2.2 Tag flags this operation. During synthesis, logical and relational operations with non-constants inside a for construct lead to a large area penalty because they generate replicated circuits.

VHDL

The STARCO5-2.9.2.2 Tag flags logical and relational operations involving non-constants in for constructs.

The STARCO5-2.9.2.2 Tag flags a message wherever a non-constant participates in any of the following types of logical and relational operations in a for construct:

Relational operations (=, \=, >, >=, <, <=)
Logical operations (and, or, nand, nor, xor, xnor)

By default, the STARCO5-2.9.2.2 Tag flags all non-constants involved in logical or relational operations in for constructs. Set the [flag_only_induction_var](#) Tag parameter to flag only those non-constants that are induction variables and are involved in logical or relational operations in for constructs. In the following example, signal `b[i]` is an induction variable:

```
for i in 0 to 10 loop
  if (b[i] or 1) then
    ...
```

As the induction variable `b[i]` is involved in a logical or relational operation in a for construct, the STARCO5-2.9.2.2 Tag flags this operation.

During synthesis, logical and relational operations with non-constants inside a for construct lead to a large area penalty because they generate replicated circuits.

Message Details

Verilog

The following message appears at the location of a logical or relational operation in a for construct that involves a non-constant:

Logical or relational operation performed with non-constant within for-loop

VHDL

The following message appears at the location of a logical or relational operation expression `<expr>` involving a non-constant variable in a for construct:

STARCO5 Tags

Logical or relational operation performed with non-constant in expression '<expr>' within for-loop

Severity

Recommended

STARC05-2.9.2.3

If logical or relational operation of non-constant variables exists in a "for" construct, the number of loop iterations must not be more than the number specified by the parameter 'maxLoopIter' (Default value 10).

Description

Verilog

The STARC05-2.9.2.3 Tag flags logical and relational operations involving non-constants in for constructs that have more iterations than the number specified by the [maxLoopIter](#) Tag parameter.

By default, the [maxLoopIter](#) Tag parameter is set to 10. You can change the default value of `maxLoopIter` Tag parameter to any positive integer.

The STARC05-2.9.2.3 Tag can evaluate the number of loop iterations only for those for constructs that satisfy the following conditions:

1. Loop initialization expression assigns a constant value to a single scalar loop variable.
2. Loop termination condition is a simple relational or logical expression where the loop variable is compared to a constant value.
3. Loop step condition is a simple arithmetic operation on the loop variable involving a single arithmetic operator which has the loop variable and a constant as its operands.

By default, the STARC05-2.9.2.3 Tag flags all non-constants involved in logical or relational operations in for constructs with more than 10 iterations. Set the [flag_only_induction_var](#) Tag parameter to flag only those non-constants that are induction variables and are involved in logical or relational operations in for constructs with more than 10 iterations. In the following example, signal `b[i]` is an induction variable:

```
for (i=0;i<=100;i=i+1)
    if (b[i] || 1)...
```

As the induction variable `b[i]` is involved in a logical or relational operation in a for construct with more than 10 iterations, the STARC05-2.9.2.3 Tag flags this operation.

During synthesis, logical and relational operations with non-constants inside a for construct lead to a large area penalty because they generate replicated circuits.

VHDL

The STARC05-2.9.2.3 Tag flags logical and relational operations involving non-constants in for constructs that have more iterations than the number specified by the [maxLoopIter](#) Tag parameter.

By default, the [maxLoopIter](#) Tag parameter is set to 10. You can change the default value of maxLoopIter Tag parameter to any positive integer.

The STARC05-2.9.2.3 Tag flags a message wherever a non-constant participates in any of the following types of logical and relational operations in a for construct with more than 10 iterations:

Relational operations (=, \=, >, >=, <, <=)
Logical operations (and, or, nand, nor, xor, xnor)

By default, the STARC05-2.9.2.3 Tag flags all non-constants involved in logical or relational operations in for constructs with more than 10 iterations. Set the [flag_only_induction_var](#) Tag parameter to flag only those non-constants that are induction variables and are involved in logical or relational operations in for constructs with more than 10 iterations. In the following example, signal `b[i]` is an induction variable:

```
for i in 0 to 100 loop
  if (b[i] or 1) then
    ...
```

As the induction variable `b[i]` is involved in a logical or relational operation in a for construct with more than 10 iterations, the STARC05-2.9.2.3 Tag flags this operation.

During synthesis, logical and relational operations with non-constants inside a for construct lead to a large area penalty because they generate replicated circuits.

Message Details

Verilog

The following message appears at the location of a logical or relational operation involving a non-constant in a for construct that has `<num>` (more than 10) iterations:

Logical or relational operation performed with non-constant

within for-loop. Loop has '<num>' (greater than 10) iterations

VHDL

The following message appears at the location of a logical or relational operation involving non-constants in a for construct that has *<num>* (more than 10) iterations:

Logical or relational operation performed with non-constant within for-loop. Loop has '<num>' (greater than 10) iterations [Hierarchy: '<entity-name>(arch-name):']

Severity

Recommended1

STARC05-2.9.2.4

Separate for loops must be used in reset and logic parts of flip-flop descriptions. (Verilog + VHDL)

Description

The STARC05-2.9.2.4 Tag flags a violation when it encounters a 'for' loop containing code that leads to the inferring of a flip-flop with asynchronous set or reset. This form of description is not accepted by synthesis tool. These tools require an asynchronous reset clause "if (<reset_condition>)" beginning the always/process block. However, in this form, the for-loop statement begins the always/process statement and causes an error in synthesis tool compilation.

The STARC05-2.9.2.4 Tag also reports a violation when both part of if-else construct contains a static part.

Message Details

Verilog

The following message appears at the location of a for loop construct that contains both reset part and logic part of flip-flop description *<flop-name>* under it:

Reset and logic parts for flip-flop '*<flop-name>*' under same for loop

VHDL

The following message appears at the location of a for loop construct that contains both reset part and logic part of a flip-flop description:

Reset part and logic part for a flip-flop mixed in for-loop

Severity

Mandatory

Verilog Examples

Example 1

Consider the following example where both reset part and logic part of flip-flop description `out[i]` are under the same for loop:

```
module mod_2_9_2_4_S_Test1 (clk, reset);
    input clk;
    input reset;

    integer i;
    reg [3:0]in [3:0];
    reg [3:0]out [3:0];

    always @ (posedge clk or negedge reset)
        for (i = 0; i < 4; i = i + 1)//violation
            if ( !reset )
                out[i] = 4'b0000;
            else
                out[i] = in[i];
endmodule
```

For this example, SpyGlass generates the following message:

Reset and logic parts for flip-flop 'out[i]' under same for loop.

To remove this message, rewrite the example to move the reset part out of the for loop construct as follows:

```
module mod_2_9_2_4_S_Test1 (clk, reset);
    input clk;
    input reset;

    integer i;
    reg [3:0]in [3:0];
    reg [3:0]out [3:0];

    always @ (posedge clk or negedge reset)
        if ( !reset )
            for (i = 0; i < 4; i = i + 1)
                out[i] = 4'b0000;
            else
                for (i = 0; i < 4; i = i + 1)
                    out[i] = in[i];
endmodule
```

Consider the following example where both part of if-else construct contain the static part:

```
always @ (posedge clk or negedge reset)
for (i = 0; i < 4; i = i + 1)
  if ( !reset )
    out[i] = 1'b0;
  else
    out[i] = 1'b1;
```

VHDL Examples

Example 1

Consider the following example where flip-flop description Q with asynchronous reset is contained in for loop LOOP_1:

```
library ieee;
use ieee.std_logic_1164.all

PROC1: process( CLK, RST )
begin
  LOOP_1:for I in 1 to 10 loop
    if ( RST = 0 ) then
      Q(I) <= 0;
    elsif ( CLK'event and CLK = '1' ) then
      Q(I) <= D(I);
    end if;
  end loop;
end process PROC1;
```

Example 2

Consider the following example where both part of if-else construct contain the static part:

```
PROC1:process(CLK,RST)
begin
  LOOP_1:for I in 1 to 10 loop
    if (RST = 0 ) then
      Q(I)<= 0;
    elsif (CLK  event and CLK = '1') then
```

```
        Q(I) <= 1;  
    end if;  
end loop;  
end process PROC1;
```

STARC05-3.1.2.7

Do not use embedded dc_shell scripts. (Verilog)
Do not use embedded dc_shell scripts in the design (VHDL)

Description

The STARC05-3.1.2.7 Tag flags embedded `dc_shell` scripts in the design.

The STARC05-3.1.2.7 Tag flags the following embedded `dc_shell` script commands:

■ For Verilog:

```
//synopsys dc_script_begin  
//synopsys dc_script_end
```

■ For VHDL:

```
-- synopsys dc_script_begin  
-- synopsys dc_script_end
```

Embedding Design Compiler `dc_shell` scripts in the design makes it technology dependent. Hence, it is recommended to create separate files for `dc_shell` scripts.

The STARC05-3.1.2.7 Tag ignores pragmas other than `synopsys` pragmas.

Message Details

Verilog

The following message appears at the location where an embedded `dc_shell` script command `<pragma>` is encountered.

Pragma '`<pragma>`' used in design

VHDL

The following message appears at the location where an embedded `dc_shell` script command `<pragma>` is encountered.

Synopsys Pragma '`<pragma>`' used in design

Severity

Mandatory

STARC05-3.1.3.1

The order of module port declarations and instance port connection lists should be same as the order in the module port map.(Verilog)

The order of entity port declarations and instance port map should be same as the order in the entity port map. (VHDL)

Description

Verilog

The STARC05-3.1.3.1 Tag flags port declarations and instance port connection lists that are not in the same order as the order in the corresponding module's port map.

To avoid excessive messages, the STARC05-3.1.3.1 Tag reports a message for the first offending port declaration or instance port connection only and ignores subsequent offending port declaration or instance port connection.

Maintaining consistency in ordering of ports across module port maps, port declarations, and instance port connection lists allows for better readability and error-free instantiations.

VHDL

This Tag flags port declarations and instance port map that are not in the same order as the order in the corresponding entity's port map.

Maintaining consistency in ordering of ports across entity port maps, port declarations, and instance port map allows for better readability and error-free instantiations. To avoid excessive violations, this Tag reports a violation only for the first offending port declaration or instance port connection.

Message Details

Verilog

The following message appears at the location of a port declaration `<port-name>` in a module or a port connection `<port-name>` in a module instance that is in an order different from the order specified in module port list:

Port declaration or port connection for '`<port-name>`' not in correct order

VHDL

The following message appears at the location of a port declaration `<port-name>` in an entity or a port connection `<port-name>` in an entity instance that is in an order different from the order specified in entity port list:

Order of specification of port `<signal name>` not consistent with port declaration.

Severity

Recommended

Examples

Verilog

Consider the following example where ports `dd` and `cc` are declared in module `test` in an order different from their order in the module port list:

```
module MOD(aa, bb, cc, dd, ee, ff); //port map
    input aa; //port declaration
    input bb; //port declaration
    input dd; //port declaration
    input cc; //port declaration

    output ee;
    output ff;
    ...
endmodule

module top
    ...
    MOD U1(.aa(in0), .bb(in1), .cc(in2), .dd(in3), .ff(out0),
    .ee(out1)); //instance port connection list
    ...
endmodule
```

Similarly, ports `ff` and `ee` are connected in module instance `U1` in an order different from their order in the module port list.

For this example, SpyGlass generates the following messages:

order of specification of port 'dd' not consistent with port declaration

order of specification of port 'ff' not consistent with port declaration

VHDL

```

library ieee;
use ieee.std_logic_1164.all

architecture RTL of t3131 is
  component tcpict69c4
    port (CLK33M      : in  std_logic;           -- PCI clock
          CLK19M      : in  std_logic;           -- Main clock
          RSTN        : in  std_logic;           -- reset
          READ_TRG     : in  std_logic;           -- CPU read timing
          TST_DATA     : in  std_logic_vector(31 downto 0); -- test data
    input
      CNT_ENB        : in  std_logic;           -- count enable sig.(33M)
      CLR_ADR        : in  std_logic;           -- CPU address selection
    -- 02/19 add

    TST_LOAD        : in  std_logic;           -- test data load enable

    DRG_DATA        : out std_logic_vector(31 downto 0) -- REG data
    output
    );

  end component;

  -- Signal Declaration
  signal cpurdy_p0 : std_logic;
  signal TXCOLDIS0 : std_logic;
  signal colnum0_c : std_logic;
  signal KILL_L : std_logic_vector (31 downto 0);
  signal colnum0_dat : std_logic_vector (31 downto 0);

begin -- RTL

```

```
-- Instance Call for above component
U_COLNUM0 : tcpict69c4
  port map
    CLK33M    => CLK33M ,
    CLK19M    => CLK19M ,
    RSTN      => RSTN ,
    READ_TRG  => cpurdy_p0 ,
    CNT_ENB   => TXCOLDIS0 , -- Detected for STARC05-3.1.3.1
    CLR_ADR   => colnum0_c ,
    TST_LOAD  => KILL_L(0) ,
    TST_DATA  => KILL_L ,
    DRG_DATA  => colnum0_dat);

end RTL;
```

STARC05-2.9.3.1

Exit or next statement must not be used in a for loop (VHDL)

Description

The STARC05-2.9.3.1 Tag flags exit or next statements in for loop constructs. Using exit or next statement in for loop may generate wrong synthesis results. Synthesis of such constructs may also generate large circuits.

Message Details

The following message appears at the location where an exit statement is encountered in a for loop construct:

Exit statement used in FOR loop

The following message appears at the location where a next statement is encountered in a for loop construct:

Next statement used in FOR loop

Severity

Mandatory

STARC05-3.1.3.2vc

Ports not declared in recommended order. (VHDL)

Description

The STARC05-3.1.3.2vc Tag flags port declarations that are not specified in recommended order, for an entity that does not have any architecture.

It is recommended to specify the ports in the following order:

Input ports, output ports, inout ports, buffer, and then the linkage ports.

Under the recommended port definition order, a message is generated for the following cases:

- If every input port is specified after an output, inout, buffer, or linkage port
- If every output port is specified after an inout, buffer, or linkage port
- If every inout port is specified after a buffer or linkage port
- If every buffer port is specified after a linkage port

Buffer ports should be declared after inout ports and before linkage ports.

Specifying ports in a consistent order aids understanding and reduces the likelihood of misconnects.

Message Details

The following message appears at the location of a entity port map where the port `<port-name>` is not defined in the recommended order:

Port '`<port-name>`' not defined in recommended order

Severity

Recommended3

Suggested Fix

Reorder port definitions to comply with this convention.

STARC05-3.1.3.4a

**Only one port or signal declaration should be in one line. (Verilog)
Define one signal per line in I/O, reg declaration, and wire
declaration (VHDL)**

Description

Verilog

The STARC05-3.1.3.4a Tag flags those port or signal declarations that are not in their own line.

Declaring ports and signals in separate lines improves readability and provides space for individual comments.

VHDL

The requirement of this Tag is covered by the [STARC05-3.5.6.3a](#) Tag.

Message Details

Verilog

The following message appears at the location of a port or signal declaration that is not in its own line:

Multiple port or signal declarations in one line

Severity

Recommended

STARC05-3.1.3.4b

Port and signal declarations should have comments. (Verilog) Declarations should have comments (VHDL)

Description

Verilog

The STARC05-3.1.3.4b Tag flags port and signal declarations without corresponding comments.

All port and signal declarations must have corresponding comments in the same or immediately preceding line.

To avoid excessive messages, number of messages has been restricted to 30 per file and 600 per design, by default. Set the [allviol](#) Tag parameter to *yes* or *<Tag-name>* to report all violation messages.

A comment explains clearly the purpose of the port or signal, thus increasing understanding for future reuse.

VHDL

The requirement of this Tag is covered by the [STARC05-3.5.6.3b](#) Tag.

Message Details

Verilog

The following message appears at the location where a port or signal is declared without a corresponding comment:

Port or signal declaration does not have comments

Severity

Recommended

STARC05-3.1.3.5

Description order of declarations should be type/subtype declaration, constant declaration, signal declaration and component declaration. (VHDL)

Description

The STARC05-3.1.3.5 Tag flags object declarations that are not in the recommended order.

The STARC05-3.1.3.5 Tag requires that object declaration should be in the following order:

type, subtype, constant, attribute declaration, attribute specification, signal, variable, alias, component

The STARC05-3.1.3.5 Tag checks for declarations inside the following constructs:

- Entities
- Architectures
- Package declarations
- Package bodies
- Procedure bodies and function bodies
- Block constructs
- if-generate and for-generate statements
- Process blocks

NOTE: *The STARC05-3.1.3.5 Tag supports generate-for block.*

To avoid excessive messages, the STARC05-3.1.3.5 Tag reports only the first offending declaration inside a scope. Set the value of the [strict](#) Tag parameter to `yes` or `STARC05-3.1.3.5` to report all messages.

Message Details

The following message appears at the location where an object declaration `<name>` is not in the recommended order `<order>` in the current scope:
Declaration '`<name>`' not described in correct order '`<order>`'

Severity

Recommended

STARC05-3.1.4.2

Code is not correctly indented. (Verilog)
Standardize the number of indents (two spaces). (VHDL)

Description

Verilog

The STARC05-3.1.4.2 Tag flags incorrectly indented code.

Indentation of code improves readability of the design. The STARC05-3.1.4.2 Tag checks for indent of two spaces per indent level.

For example, the if statements in a block are to be indented by one means, you need to shift the first statement by two spaces.

Follow the indentation guidelines given below:

- All ports, signal, variable declaration should be indented by four spaces.
- begin statement should be in the same line where the construct starts or if it is on the next line, it should be indented by four spaces.

Consider the following example:

```
always (posedge clk ) begin
```

OR

```
always (posedge clk )
    begin // 2 spaces
```

If a statement is broken in more than one line, then it should be indented by two spaces.

Set the [starc2005_number_for_indent](#) Tag parameter to a different value, to specify the allowed number of spaces at the first character in a line in an always construct.

By default, only 20 violations per file and 500 violations per design is reported. Set the [allviol](#) Tag parameter to *yes* or *<Tag-name>* to report all violation messages.

The STARC05-3.1.4.2 Tag checks all HDL constructs.

VHDL

The STARC05-3.1.4.2 Tag flags incorrectly indented code lines of the design.

Using effective indentation improves the readability of RTL code with complex constructs, like deeply nested if-else constructs, case constructs, wire or net declarations, comments, function descriptions, task descriptions, and so on.

Following checks are performed by the STARC05-3.1.4.2 Tag:

- All compiler directives, module definitions, and UDP definitions start at the same indent level as previous line
- Parameter, port, module, UDP, net declaration should be indented by two spaces from the previous indent level
- Table keyword in a UDP should be indented by two spaces
- always, if, repeat, forever, for, named-block, while, task, and function constructs should be indented by two spaces
- case statements should be indented by two spaces from corresponding case statement
- Statements that exceed the line length (and therefore have to fold to multiple lines) should start at the same indent level as previous line
- begin keyword should be at the same line as corresponding block construct
- One blank line should be there before and after every always, initial, function and task blocks
- Rest of the statements should be indented by two spaces from the previous indent level

Set the [starc2005_number_for_indent](#) Tag parameter to a different value, to specify the allowed number of spaces at the first character in a line in a process construct.

By default, only 20 violations per file and 500 violations per design is reported. To report all the violations, set the [allviol](#) Tag parameter to yes.

Severity

Reference

Suggested Fix

Indent the code according to the above specification.

STARC05-3.1.4.3

Tabs should not be used. (Verilog)
Tabs should not be used in the RTL description. (VHDL)

Description

The STARC05-3.1.4.3 Tag flags first occurrence of the tab character found in the design file.

It is recommended to use spaces instead of tabs.

Following this guideline enhances readability of the design across different text editors and editor settings.

Message Details

The following message appears at the location where a tab character is found:

Replace tabs with equivalent spaces

Severity

Recommended

STARC05-3.1.4.5

The maximum number of characters in one line should not be more than 110

Description

The STARC05-3.1.4.5 Tag flags long code lines.

By default, the STARC05-3.1.4.5 Tag flags code lines that are longer than 110 characters. Use the [line_length_max](#) Tag parameter to specify a different number.

Longer lines can be difficult to read on small terminals.

Message Details

The following message appears at the code line that is longer than \$line_length_max characters:

Line length is more than \$line_length_max characters

Severity

Reference

STARC05-3.1.5.2

Vector signal bit-widths should be specified using parameters. (Verilog)

Description

The STARC05-3.1.5.2 Tag flags non-port vector signals with directly specified bit-widths.

It is recommended to use parameters to specify vector signal bit-widths.

Specifying bit-width for vector objects using parameters makes the design reusable and easily maintainable.

To avoid excessive messages, the STARC05-3.1.5.2 Tag reports only first five messages for a module. To view all messages, set the [strict](#) Tag parameter to `yes` or STARC05-3.1.5.2.

NOTE: *The STARC05-3.1.5.2 Tag also flags non-port vector signals where the bit-width is specified using text macros (``define` directive).*

Message Details

The following message appears at the location where a vector signal `<sig-name>` is declared without using parameters for bit-width specification:

Bit-width specification of signal `<sig-name>` is not parameterized

Severity

Reference

STARC05-3.1.5.2v

Non-port vectors bit-width should be specified using generics. (VHDL)

Description

The STARC05-3.1.5.2v Tag flags signal or variable vectors with directly specified bit-widths.

The STARC05-3.1.5.2v Tag checks for declarations inside the following constructs:

- Entities
- Architectures
- Block constructs
- if-generate and for-generate statements
- Process blocks

To avoid excessive messages, the STARC05-3.1.5.2v Tag reports only the first five offending declarations inside a scope. Set the [strict](#) Tag parameter to report all messages.

NOTE: *The STARC05-3.1.5.2v Tag does not be check in top-level design units as the [STARC05-1.1.4.9](#) requires that generics should not be used at top-level.*

Message Details

The following message appears at the location where a signal or variable `<name>` is declared with bit-width specified without using generics:

Bit-width specification of signal '`<name>`' does not use generic

Bit-width specification of variable '`<name>`' does not use generic

Severity

Reference

STARC05-3.1.6.1

Generic should be used in conditional expression of if-generate statement (VHDL)

Description

The STARC05-3.1.6.1 Tag flags if-generate statements that do not use generics in their conditional expression.

When the values given to conditional expressions are given as generics from the upper layer, the circuits being generated can be switched. It is, therefore recommended to use generics in conditional expression of if-generate statement.

NOTE: *The STARC05-3.1.6.1 Tag supports generate-for block.*

NOTE: *The STARC05-3.1.6.1 Tag will not be check in top-level design units, as the [STARC05-1.1.4.9](#) Tag requires that generics should not be used at top-level.*

Message Details

The following message appears at the location where an if-generate statement does not use generics in its conditional expression:

Generic not used in conditional expression of if-generate statement

Severity

Recommended

STARC05-3.1.6.2

Generic should be used in loop parameter range specification of for-generate statement (VHDL)

Description

The STARC05-3.1.6.2 Tag flags for-generate statements that do not use generics in their loop parameter range specification.

Using generics in loop parameter range specification of for-generate statement makes the design reusable.

NOTE: *The STARC05-3.1.6.2 Tag will not be check in top-level design units, as the [STARC05-1.1.4.9](#) Tag requires that generics should not be used at top-level.*

Message Details

The following message appears at the location where an for-generate statement does not use generics in its loop parameter range specification:

```
Generic not used in loop parameter range specification of for-  
generate statement
```

Severity

Caution

STARC05-3.2.2.1

Parameters should be used to specify constants (Verilog)

Description

The *STARC05-3.2.2.1* Tag reports non-parameterized constants used in the design.

To facilitate change and improve shared code maintenance, constants should be specified by using parameters.

The Tag ignores the following constants:

- Numeric constants 0 to 7 (leading zeros, if any are ignored)
- Binary constants 1'b0 and 1'b1
- 0 and 1 expressed as multi-bit constants (n'b0000..0 and n'b1111..1 respectively, where *n* is an integer)

Therefore, the numeric constant 2 is allowed whereas the same number expressed as a multi-bit constant (for example, 8'd2) is not allowed.

The Tag ignores constants that are used to describe vector bit-widths because such constants are covered by the [STARC05-3.1.5.2](#) and [STARC05-1.1.4.7](#) Tags.

To avoid excessive messages, the Tag reports only first five messages for a module. Set the *strict* parameter to *yes* or *STARC05-3.2.2.1* to view all messages.

By default, the Tag does not report violations for non-parameterized constants used as array indexes. Set the *check_index* parameter to *yes* to report violations for such cases.

Message Details

The following message appears at the location where a non-parameterized constant *<const-value>* is encountered:

Constant *<const-value>* is not parameterized

Severity

Recommended3

STARC05-3.2.2.1v

Do not use hard-coded constant in the design. (VHDL)

Description

The STARC05-3.2.2.1v Tag flags hard-coded constants used in the design. To facilitate change and improve shared code maintenance, constants should be specified using generic or constant constructs.

The STARC05-3.2.2.1v Tag ignores the following constants:

- Numeric constants 0 to 7
- Constants 'Z', 'X', 'W', 'H', 'U', 'L', '-', '0' or '1'
- Vector constants having
 - ☐ All 0's or all 1's if binary
 - ☐ All 0's or all 7's if octal
 - ☐ All 0's or all F's if hexadecimal
 - ☐ All Z's, all X's, all W's, all H's, all L's, all U's, all '-'s, all 0's or all 1's
- Constants used in declaration statements
- Constant used in report statements
- Constant used in assert statements
- Constants used as labels (case label, process label, and so on)
- Constants used in index name, slice name, and selected name
- Constants used to describe vector bit-widths

Message Details

The following message appears at the location where a hard-coded constant `<const-value>` is encountered:

Hard Constant '`<const-value>`' used in the design

Severity

Recommended3

STARC05-3.2.2.3

Parameters should be read using include files. (Verilog)

Description

The STARC05-3.2.2.3 Tag flags parameter declarations in source files.

It is recommended to declare parameters in a separate file and then include this file in the source files.

To avoid excessive messages, the STARC05-3.2.2.3 Tag reports a message only for the first found parameter declaration in a source file and ignores other parameter declarations in the source file.

Message Details

The following message appears at the location of the first found parameter declaration in a source file:

Parameter(s) declared in source file

Severity

Recommended

STARC05-3.2.2.4

Include files must be specified with more than 1 level higher relative path. (Verilog)

Description

The STARC05-3.2.2.4 Tag reports violation when the include path is specified as follows:

- path is specified as full path name
- path is not specified (current directory)
- path is specified down the current directory

Message Details

The following message appears if an include path *<path-name>* does not return to directory that is more than 1 level higher:

Include path \"*<path-name>*\" does not return to directory that is more than 1 level higher

Severity

Recommended1

STARC05-3.2.2.5

Text macros should not be nested. (Verilog)

Description

The STARC05-3.2.2.5 Tag flags nested text macros.

The defined value of a text macro should not be used in the declaration of another text macro.

For example, the following RTL segment would be flagged as a message:

```
`define macro1 1  
`define macro2 0 `macro1
```

Message Details

The following message appears at the location of a text macro definition *<macro-name>* that contains another text macro:

Nested macro '*<macro-name>*' used

Severity

Recommended

STARC05-3.2.2.7

Constants should be defined using parameters only. (Verilog)

Description

The STARC05-3.2.2.7 Tag flags constants declared using ``define` constructs.

The STARC05-3.2.2.7 Tag flags ``define` constructs where a macro is defined as a numeric value (potentially replaceable by a parameter declaration) and ignores ``define` constructs that define string constants and argument-based ``define` constructs.

While both ``define` and parameters can be used to specify constants, it is a good practice to consistently use one method. Using parameters is the better method as parameters do not, by default, become visible throughout the entire source file.

Message Details

The following message appears at the location where a macro `<macro-name>` is used to declare a constant:

Use parameter instead of macro '`<macro-name>`'

Severity

Recommended

STARC05-3.2.3.3

Do not use entity instantiation in the design (VHDL)

Description

The STARC05-3.2.3.3 Tag flags entity instantiations.

Libraries are fixed at the time of instantiation if direct entity instantiation is used. Hence, such constructs reduce the reusability of the design.

Message Details

The following message appears at the location of a direct entity instantiation `<inst_name>`:

Instance '`<inst-name>`' uses entity instantiation

Severity

Mandatory

STARC05-3.2.4.1

Non-integer type used in the declaration of a generic may be unsynthesizable. (VHDL)

Description

The STARC05-3.2.4.1 Tag flags non-integer types used in generic declarations.

Such usage may be unsynthesizable. However, using non-integer types may be required in testbenches or simulation models. Therefore, code targeted for synthesis should be changed to use only integer generics.

By default, the Tag reports violations for non-integers, such as `time` and `real`, if used in generic declaration. Set the [report_all_datatype](#) parameter to `yes` to report violations for all data types used in a generic declaration, except integer.

Message Details

VHDL

The following message appears at the location of a generic declaration `<name>` that uses non-integer type `<data-type>` is used:

Non-integer type '`<data-type>`' used in declaration of generic '`<name>`' may be unsynthesizable

Severity

Mandatory

STARC05-3.5.2.1

File references must be relative path names including the master directory. (Verilog)

Description

The STARC05-3.5.2.1 Tag flags file references in the design that are not relative path names and/or do not include the name of the specified master directory.

The STARC05-3.5.2.1 Tag checks file references in the following:

- ``include` directives
- `$fopen` system function call
- `$readmemh` and `$readmemb` system tasks.

For maintainability considerations, file references in the design must be relative path references and each reference must include name of a master directory.

The STARC05-3.5.2.1 Tag accepts the name of the master directory as an argument using the [src_master_dir](#) Tag parameter. If master directory argument is not specified, the STARC05-3.5.2.1 Tag only checks for relative file path names.

Message Details

The following message appears at the location of a file reference statement to file `<file-name>` that is not relative and/or does not include the name of the master directory specified using the `src_master_dir` Tag parameter:

File reference for `<file-name>` is either not a relative path or does not include name of the specified master directory

Severity

Recommended1

STARC05-3.5.2.1v

Each file's reference should be specified by a relative path (return to the master directory once). (VHDL)

Description

The STARC05-3.5.2.1v Tag flags file references in the design that are not relative path names and/or do not include the name of the specified master directory.

For maintainability considerations, file references in the design must be relative path references and each reference must include name of a master directory.

Use the [src_master_dir_vh](#) Tag parameter to specify the name of the master directory. If master directory is not specified, this Tag only checks for relative file path names. Have file name references return to the master directory in the relative path before specifying file names. If file names are specified by an absolute path, then it will have to be changed when the data are moved to another directory.

Message Details

The following message appears at the location of a file reference statement that is not relative and/or does not include the name of the master directory specified using the `src_master_dir_vh` Tag parameter:

File reference is either not a relative path or does not include name of the specified master directory

Severity

Recommended1

STARC05-3.5.6.2

Add comments for sub-programs, always construct and condition statements, indicating objective and contents. (Verilog)

Description

The STARC05-3.5.6.2 Tag flags uncommented constructs.

The STARC05-3.5.6.2 Tag flags tasks, functions, always constructs, and `if` statements that do not contain comments either on the same or the immediately preceding line.

Use of comments improves the readability of the source code. Comments should be added for sub-programs, always constructs, and condition statements, indicating their objective and contents.

Parameter(s)

- **`check_all_always`**: By default, the parameter is set to `no`. Set the value of the parameter to `yes` to enable the tag to check all `always` (`always_ff`, `always_comb` and `always_latch`) blocks and report violations if they do not have a corresponding comment in the same or preceding lines.
- **`ignore_empty_lines`**: By default, the parameter is set to `-1` and the tag ignores any number of white spaces/blank lines before the construct to check the comment. Set the value of the parameter to `0` or any positive integer to restrict the number of blank lines before the construct to match the comment. That is, by setting the value to `0` or any positive integer, the tag ignores that number of empty lines as specified in the value.

Message Details

The following message appears at the first line of an uncommented construct:

Description does not have comments

Severity

Recommended2

STARC05-3.5.6.2va

Process blocks should have comments. (VHDL)

Description

The STARC05-3.5.6.2va Tag flags process constructs that do not have comments on the same line or immediately preceding lines.

Using comments indicating the objective of the statement improves the readability of the source code and hence, becomes easier to maintain the source code.

Message Details

The following message appears at the first line of an uncommented process construct:

```
Process block does not have comments
```

Severity

Recommended2

STARC05-3.5.6.2vb

Sub-program descriptions should have comments. (VHDL)

Description

The STARC05-3.5.6.2vb Tag flags those function and procedure declarations and descriptions that do not have comments on the same line or immediately preceding lines.

Using comments indicating the objective of the statement improves the readability of the source code and hence, becomes easier to maintain the source code.

Message Details

The following message appears at the first line of an uncommented sub-program description:

```
Sub-program description does not have comments
```

Severity

Recommended2

STARC05-3.5.6.3a

Each declaration should be described in separate line

Description

The STARC05-3.5.6.3a Tag flags if more than one from the following type of declarations occur on the same line:

- Port declaration
- Generic declaration
- Constant declaration
- Signal declaration
- Variable declaration
- Type declaration
- Subtype declaration

Declaring declarations in separate lines improves readability and provides space for individual comments.

Message Details

The following message appears at the location where multiple declarations are found in one line:

Multiple declarations in one line

Severity

Recommended

STARC05-3.5.6.3b

Declarations should have comments (VHDL)

Description

The STARC05-3.5.6.3b Tag flags following type of declarations without corresponding comments:

- Port declaration
- Constant declaration
- Signal declaration
- Variable declaration
- Type declaration
- Subtype declaration

All declarations should have corresponding comments in the same or immediately preceding line.

To avoid excessive messages, the STARC05-3.5.6.3b Tag flags only 30 messages per source file and 600 messages per design. To generate all messages, set the [allviol](#) Tag parameter to `yes` or `1`.

A comment explains clearly the purpose of the declaration, thus increasing understanding for future reuse.

Message Details

The following message appears at the first line of an uncommented declaration:

```
Declaration does not have comments
```

Severity

Recommended

STARC05-3.5.6.4

Comments should use English characters only.

Description

The STARC05-3.5.6.4 Tag flags non-english characters used in comment lines.

The STARC05-3.5.6.4 Tag flags a message in each file where a comment includes a non-english character (that is, a character other than A-Z a-z 0-9 _ - # \$ % & * + ! < > () = @ ^ { } [] | \ ~ ` ` " , . ? /).

Use of English characters in comments allows for easier understanding of source code for projects distributed over geographical locations.

Message Details

The following message appears for a comment line where non-english character(s) are used:

Non-english character(s) used in comment line

Severity

Recommended

STARC05-3.5.6.6

Japanese comment should be described in EUC code.

Description

The STARC05-3.5.6.6 Tag flags Japanese codes ISO-2022-JP, SHIFT-JIS and UTF-8 when used as comments in RTL design.

Message Details

The following message appears where Japanese code `<code>` is used in the RTL design:

Code '`<code>`' used in comment in RTL design

Severity

Recommended1

STARC05-3.5.6.7

Comments should start with "//" (Verilog)

Description

The STARC05-3.5.6.7 Tag flags multi-line comments in the design.

NOTE: *The STARC05-3.5.6.7 Tag is available only if you set the [strict](#) Tag parameter to yes or STARC05-3.5.6.7.*

It is recommended that you use only single-line comments (// comments) and avoid multiline comments (/**/ comments).

Message Details

The following message appears for a multi-line comment line:
Multiline comment specified

Severity

Recommended

STARC05-1.1.5.2b

Reset signal names should follow the recommended naming convention

Description

The STARC05-1.1.5.2b tag flags those reset signal names from reset source to point-of-use that do not match the regular expression specified by the *starc2005_reset_string* tag parameter.

The default value of the *starc2005_reset_string* tag parameter is */RST_X/i* or */RESET_X/i*. Therefore, by default, the STARC05-1.1.5.2b tag flags those signal names from reset source to point-of-use that do not contain the string *RESET_X* or the string *RST_X*.

All RTL nets from the source of asynchronous reset signal to the location where the asynchronous reset signal is used in a flip-flop description are checked, including the intermediate nets connected across the hierarchical boundaries.

For example, the STARC05-1.1.5.2b tag checks *rst* (source), *rst1* (intermediate), and *rst2* (point-of-use) in the hierarchical design shown in the following figure:

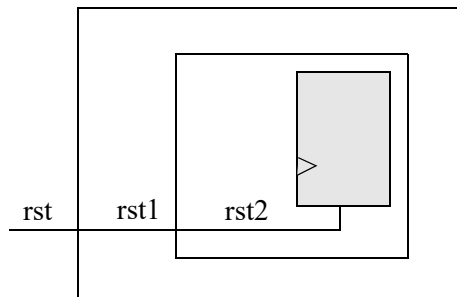


FIGURE 22. Hierarchical design

For synchronous reset signals, the STARC05-1.1.5.2b tag checks the name of the synchronous reset signal at the point-of-use only. Thus, the STARC05-1.1.5.2b tag checks the name *rst2* (point-of-use) only if the reset signal *rst2* is a synchronous reset signal.

Following this naming convention improves the readability of the design and helps in debugging.

By default, the value of the [report_sync_rst_only](#) parameter is set to no. In this case, the tag does not report for synchronous reset names. When the [report_sync_rst_only](#) parameter is set to yes, the tag processes synchronous reset name only to validate if the names are matching the format specified using the [starc2005_reset_string](#) parameter.

NOTE: *The STARC05-1.1.5.2b tag is case-insensitive.*

Message Details

The following message appears at the location where a tag-violating reset signal name `<rst-name>` is first encountered:

Reset signal name '<rst-name>' does not follow recommended naming convention

Severity

Recommended3

STARC05-1.3.2.1a

Asynchronous resets or sets must not be gated

Description

The STARC05-1.3.2.1a tag flags logical operators used in the asynchronous reset/set line.

The STARC05-1.3.2.1a tag checks that asynchronous reset/set signals are not generated from combinational gates or black boxes. Buffers and inverters in the asynchronous reset/set line are ignored.

This tag does not traverse on blocked path and ignores set/reset pin driven by a constant value which can be propagated through assignment or by using the SGDC file.

By default, the STARC05-1.3.2.1a tag checks for both set and reset signals. You can set the [priority_set_reset](#) parameter to either set or reset. The tag checking is done for the specified control signal to report a gated set or gated reset signal.

In a design using gated resets/sets, any fault in the circuit involving reset/set signal can propagate to reset or set a register. Hence, it is recommended to avoid connecting any signal other than the initial reset/set signal to the asynchronous reset/set pins.

By default, the [show_connected_net](#) parameter is set to no. Set the parameter value to yes to change the name of gated set/reset net to the name of immediate net of the gate.

By default, the [ignoreCellName](#) parameter is not set. Set the [ignoreCellName](#) parameter to a PERL regular expression to specify the asynchronous resets or sets gated through lib cell instances, which need to be ignored.

Message Details

The following message appears at the location where a gated <sig-type> (reset/set) signal <sig-name> of flip-flop <flop-name> is encountered:

Gated <sig-type> '<sig-name>' used for flip-flop '<flop-name>'

Severity

Recommended1

Verilog Examples

Consider the following example where the reset signal `intrRST` of flip-flop `OUT` has been generated by gating `RESET` and `IN2` signals:

```
module test (CLK, RESET, IN1, IN2, OUT);  
    input CLK, RESET;  
    input IN1, IN2;  
    output OUT;  
  
    reg     OUT;  
    wire    intrRST;  
  
    assign intrRST = RESET & IN2;  
  
    always@(posedge CLK or negedge intrRST)  
        if (!intrRST)  
            OUT =1'b0;  
        else  
            OUT = IN1;  
  
endmodule
```

For this example, SpyGlass generates the following message:

Gated reset 'test.intrRST' used for flip-flop 'OUT'

STARC05-2.1.2.2a

Do not use nonblocking assignment in task description. (Verilog)

Description

The STARC05-2.1.2.2a tag flags the use of nonblocking assignment in task description.

Using nonblocking assignment statements in a task construct may not be supported by some synthesis tools.

Message Details

The following message appears at the location where a nonblocking assignment is encountered in a task construct:

nonblocking assignment <assignment> used in task description

Example

Consider the following example:

```
module test (clk, a, b, c, d);  
  input  clk;  
  input  a, b;  
  output c, d;  
  
  reg c, d;  
  
  always @(posedge clk)  
  begin  
    rtl_check(a, b, c, d);  
  end  
  
  task rtl_check;  
  input a, b;
```

```
output c, d;  
if (a < b) begin  
    c = b;  
    d <= a; // nonblocking statement is used in TASK  
end  
endtask  
endmodule
```

In the above example, the *STARC05-2.1.2.2a* tag reports a violation because nonblocking assignment is used in the task description.

STARC05-2.3.2.1

Blocking assignment used in flip-flop description. (Verilog)

Description

The STARC05-2.3.2.1 tag flags flip-flop descriptions using blocking assignments.

The behavior and optimization of nonblocking assignments and blocking assignments are not same. An always construct using nonblocking assignments that model a shift register, is interpreted as a pair of parallel flip-flops when blocking assignments are used.

The STARC05-2.3.2.1 tag also flags only those nonblocking assignments in an always construct which infer sequential logic (flip-flop) in the design.

You can set the [ignore_blocking_hanging_flop](#) parameter to `yes` to enable the tag to ignore hanging flip-flops. By default this parameter is not set.

Message Details

The following message appears at the location where a blocking assignment is encountered in the flip-flop description:

Blocking assignment used in flip-flop description

Severity

Caution

STARCO5-2.4.1.3

**Latches should not have asynchronous set or asynchronous reset.
(Verilog)**
**Latches must not have asynchronous set or asynchronous reset
(VHDL)**

Description

The STARCO5-2.4.1.3 tag flags latches with asynchronous sets or resets. Latches with asynchronous set/reset description are usually not found in ASIC libraries. Also, different synthesis tools may infer such latches differently making the designs unreliable.

Parameters

[waiver_compat](#): Default value is no. If you set the value of this parameter to yes or <tag-name>, it ensures that the tag does not generate the line number information in the first run itself. Thus waivers work correctly even if the line numbers of the RTL gets changed in the subsequent runs.

Message Details

The following message appears at the location of a latch <latch-name> if a signal <sig-name> is encountered at line <line-num> that is the asynchronous set/reset of the latch:

Latch '<latch-name>' has asynchronous set/reset '<sig-name>'
(line: '<line-num>')

Severity

Mandatory

STARC05-2.4.1.5

Do not use two level latches in the same phase clock

When to Use

Use this tag to detect cascaded latches in the same phase clock.

Description

The STARC05-2.4.1.5 tag flags cascaded latches with same phase enable pins.

tag Exceptions

This tag does not traverse on blocked path and ignore latches with enable pin driven by a constant value which can be propagated through assignment or by using an SGDC file.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[RECOMMENDED1] Two-level latches '<latch1>' and '<latch2>' are in the same phase enable

Potential Issues

Violation may arise when two cascaded latches have the same phase enable pins.

Consequences of Not Fixing

If there is more than one latch in the same phase, it would mean that there are paths that pass through multiple latches. Circuits like this may be

acceptable in terms of their behavior but they create problems in timing analysis and should not be used.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic shows the cascaded latches and their common enable path. You can trace through the output of the one latch to the D pin of the other latch.

To fix the violation, avoid using same phase enable pin for two-level latches.

Example Code and/or Schematic

Consider the following example:

```
module mod (in1, in2, out1, out2, en);
  input in1, en;
  input [1:0] in2;
  output out1;
  output [1:0] out2;
  reg out1;
  reg [1:0] out2;
  reg [1:0] w1;

  always @(en or in1)
    if (en) w1 <= in2;

  always @(en or in2)
    if (en) out2 <= w1;
endmodule
```

In the above example, the STARCO5-2.4.1.5 tag reports a violation as the design uses two level latches in the same phase enable pins.

Default Severity Label

Recommended1

tag Group

RTL Description Technique

STARCO5 Tags

Reports and Related Files

No related reports or files

STARC05-2.5.1.2

Ensure that logic does not exist in tristate enable conditions

When to Use

Use this tag to ensure that logic does not exist in tristate enable conditions.

Description

The STARC05-2.5.1.2 tag reports violation for logic in tristate enable conditions. The tristate buffer enable (select) condition must have only a signal name described without using more complex logic. Complex logic like any combo logic, sequential cell, etc., except inverters and buffers between tristate enable and module input ports, if used in tristate enable condition is reported.

Tag Exceptions

Following are the exceptions to the STARC05-2.5.1.2 tag:

- Supports block path analysis, that is, it supports constant value propagation. Therefore, this tag does not traverse on the blocked path.
- Ignores the tristate enable pin driven by a constant value, which can be propagated through assignment or through [set_case_analysis](#) constraint.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

- [ignore_flops](#): The default value is no. Set the value of the parameter to yes to ignore violations where flip-flop is connected to the tri-state logic.
- [strict](#): The default value is no. Set the value of the parameter to yes to report logic in tristate enable conditions outside the module. You can also provide a comma or space-separated list of tags as an input.

Constraint(s)

- [*set_case_analysis*](#): Use this constraint to set the initial values for block path analysis or to propagate the tristate enable pin driven by a constant.
- [*assume_path*](#): Use this constraint to specify the paths that exist between the input pins and the output pins of black boxes.

Messages and Suggested Fix

The following message appears at the location where logic is encountered in enable condition of tristate buffer *<sig-name>* in hierarchy *<hier-path>*:

[MANDATORY] Tristate buffer '*<sig-name>*' has logic in enable condition. [Hierarchy: '*<hier-path>*']

Potential Issues

Violation may arise when a tristate buffer has combinational logic in enable condition.

Consequences of Not Fixing

Placing complex logic in the enable line of the tristate buffer might change the timing of the switching of the buffer owing to potential changes in logic sequence after synthesis. Such timing changes may cause bus contentions resulting in malfunction or higher power consumption.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic shows the logic connected to the enable pin of tristate buffer.

To fix the violation, ensure that the tristate enable pin is connected to the input port directly or via buffers or inverters in the same design unit.

Example Code and/or Schematic

Example 1

In the following example, the tristate enable condition for signal out is the result of a logical operation and thus violates the STARCO5-2.5.1.2 tag:

```
assign out = in1&&in2? in2 : 1'bz;
```

Example 2

In the following example, the tristate enable signal `enb` has been specified by a `case` construct, thus violating the STARC05-2.5.1.2 tag:

```
case(enb)
  1'b0 : tso = 4'bzzzz;
  1'b1 : tso = in;
endcase
```

Example 3

In the following example, the tristate enable signal `enb` has been specified by an `if-case` construct, thus violating the STARC05-2.5.1.2 tag:

```
if(enb == 1'b0)
  tso = 4'bzzzz;
else
  tso = in;
```

Example 4

Consider the following example:

```
module mod_2_5_1_2_test1(in,enb1, enb2,tso);
  input [3:0] in;
  input enb1, enb2;
  output [3:0] tso;
  reg [3:0] tso;

  always@(enb1 or enb2 or in)
  begin
    case(enb1 && enb2)
      1'b0 : tso = 4'bzzzz;    //tristate buffer
      1'b1 : tso = in; //violation (Tristate buffer
'tso[3:0]' has logic in enable condition)
    endcase
  end

endmodule
```

In the above example, the STARC05-2.5.1.2 tag reports a violation as the tristate buffer, `tso[3:0]`, has logic in enable condition.

Example 5

Consider the following example:

```
module R(D,in1,in2 ,clk,Q);
input D, in1,in2; // Data input
input clk; // clock input
output reg Q; // output Q
always @(posedge clk)
begin
    if(in1&&in2)
        Q <= D;
    else
        Q <= 1'bz; //no violation under new parameter (ignore_flops)
    end
endmodule
```

Example 6

Consider the following example:

```
module TOP( D,in1,in2 ,clk,Q);
input D, in1,in2; // Data input
input clk; // clock input
output reg Q; // output Q
R    d-flop(D,in1,in2 ,clk,Q);
endmodule

module d-flop(D,in1,in2 ,clk,Q);
input D, in1,in2; // Data input
input clk; // clock input
output reg Q; // output Q
always @(posedge clk)
```

```
begin
  if(in1&&in2)
    Q <= D;
  else
    Q <= 1'bz; //no violation under new parameter (ignore_flops)
  end
endmodule
```

Example 7

Consider the following example:

```
module mod1 (in1 ,out1 , clk );

  input in1,clk ;
  output out1;
  reg out1,out2;
  reg a , b;

  always @(posedge clk)
    if(a||b)
      out1<=in1;
    else
      out1<= 1'bz; //no violation under new parameter (ignore_flops)
  endmodule
```

Example 8

Consider the following example:

```
module R(D,in1,in2 ,clk,Q);
  input D, in1,in2; // Data input
  input clk; // clock input
```

```

output reg Q; // output Q
always @(posedge clk)
begin
    if(in1==in2)
        Q <= D;
    else
        Q <= 1'bz; //no violation under new parameter (ignore_flops)
end
endmodule

```

Example 9

Consider the following example:

```

module R(D,in1,in2 ,clk,Q);
input D, in1,in2; // Data input
input clk; // clock input
output reg Q; // output Q
always @(posedge clk)
begin
    if(in1>in2)
        Q <= D;
    else
        Q <= 1'bz; //no violation under new parameter (ignore_flops)
end
endmodule

```

Example 10

```

module top(en1,en2,din,qout);
input en1,en2,din;
output qout;

```

```
wire en = en1 & en2;

tristate t1(en,din,qout);

endmodule

module tristate(e,d,q);
  input e,d;
  output q;

  assign q = e? d : 1'bz; //violation under strict
parameter
```

Example 11

```
module top(en1,en2,din,qout);
  input en1,en2,din;
  output qout;

  wire en = en1 & en2;

  tristate t1(en,din,qout);

endmodule

module tristate(e,d,q);
  input e,d;
  output q;

  assign q = e? d : 1'bz; //violation under strict
parameter
```

STARCO5 Tags

endmodule

Default Severity Label

Mandatory

tag Group

RTL Description Technique

Reports and Related Files

No related reports or files

STARC05-2.5.1.4

Identifies the tristate bus that is driven by more than specified number of drivers

When to Use

Use this tag to detect tristate buses that are driven by more than specified number of drivers.

Description

The STARC05-2.5.1.4 tag reports violation for the tristate buses that are driven by more than specified number of drivers.

A bus qualifies as a tristate bus, if it is driven by tristatable signals only.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

- ***tristate_count***: The default value is 5. Set the value of the parameter to any positive integer value to set the threshold number of drivers for tristate bus checking.
- ***MAX_DRIVER_COUNT***: The default value is 200. Set the value of the parameter to a positive integer value to limit the extraction of drivers in fan-in cone to a specified value.

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location of a tristate signal *<sig-name>* that has more than \$tristate_count drivers:

[RECOMMENDED] Tristate bus '*<sig-name>*' has more than \$tristate_count drivers

Potential Issues

Violation may arise when a tristate bus is driven by more than the specified number of drivers.

Consequences of Not Fixing

Heavily loaded bi-directional buses result in drastic increase in wire area during layout, which creates contention issues.

How to Debug and Fix

Double-click the violation message that you want to debug. View the corresponding Incremental Schematic. The Incremental Schematic window highlights the multiple driven tristate bus.

Example Code and/or Schematic**Verilog**

Consider the following example:

```
module mod_2_5_1_4_test1(inp,ts);
input [5:0] inp;
output ts;
reg ts;
reg en;

always @(inp[0] or en)
ts = (!en) ? inp[0] : 1'bz;

always @(inp[1] or en)
ts = (!en) ? inp[1] : 1'bz;

always @(inp[2] or en)
ts = (!en) ? inp[2] : 1'bz;
always @(inp[3] or en)
ts = (!en) ? inp[3] : 1'bz;

always @(inp[4] or en)
ts = (!en) ? inp[4] : 1'bz;
always @(inp[5] or en)
ts = (!en) ? inp[5] : 1'bz;
```

```
endmodule
```

In the above example, the STARC05-2.5.1.4 tag reports a violation because the tristate bus, `mod_2_5_1_4_test1.ts`, has more than five drivers.

VHDL

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all;

entity top is
  port(
    en1 : in std_logic_vector(5 downto 0);
    a : in std_logic(5 downto 0);
    y : out std_logic
  );
end top;

architecture arc_top of top is
begin
  y <= a[0] when (en[0] ='1') else 'Z';
  y <= a[1] when (en[1] ='1') else 'Z';
  y <= a[2] when (en[2] ='1') else 'Z';
  y <= a[3] when (en[3] ='1') else 'Z';
  y <= a[4] when (en[4] ='1') else 'Z';
  y <= a[5] when (en[5] ='1') else 'Z';
end arc_top;
```

In the above example, the STARC05-2.5.1.4 tag reports a violation as the tristate bus, `top.y`, has more than five drivers.

Default Severity Label

Recommended

tag Group

RTL Description Technique

STARCO5 Tags

Reports and Related Files

No related reports or files

STARC05-2.5.1.5a

Nets other than tristate nets must not be driven by multiple drivers. (Verilog)
Net that is not tristated has multiple drivers. (VHDL)

Description

Verilog

The *STARC05-2.5.1.5a* tag reports nets that are not tristate nets but have multiple drivers. A net is considered as a tristate net if all the drivers in its immediate fan-in are tristate elements. Such nets are expected to have multiple drivers. For other nets, having multiple drivers is an error condition.

Use the [IGNORE_SAME_DRIVERS](#) parameter to ignore drivers of same type.

This tag ignores all the black boxes that are driving a net. The signals driven by black boxes are not considered as having multiple drivers.

By default, this tag does not consider inout ports and bus holders as net drivers. Set the value of the [strict](#) parameter to *yes* or *STARC05-2.5.1.5a* to consider inout ports and busholders as net drivers.

By default, this tag does not report a violation for the following:

- Unloaded signals or variables
- Hanging driver nets and drivers of net through the gray-box element
- Undriven nets connected to the output port of an instance

Use the [assume_driver_load_srp](#) parameter to report violations for such cases.

VHDL

The *STARC05-2.5.1.5a* tag reports nets that are not tristate nets and are driven from more than one sequential block, and/or an instance, and/or a concurrent statement.

A net is considered as a tristate net if all the drivers in its immediate fan-in are tristate elements. This tag does not report violations for such tristate nets.

This tag ignores all the black boxes that are driving a net. The signals driven by black boxes are not considered as having multiple drivers.

NOTE: *The STARC05-2.5.1.5a tag reports those data types, which have built-in resolution function (such as `std_logic`). If the data type has no built-in resolution function, the STARC05-2.5.1.5b tag is reported.*

If the [strict](#) parameter is not set, the tag ignores bus-holders and inout ports as drivers.

Currently, this tag does not report a violation if a net connected to the output port of an instance is not driven inside that instance. But if the [checkDriverInDu](#) parameter is set, this tag considers the net connected to the output port of an instance as driven.

By default, the tag extracts all the drivers in the fan-in cone. Set the value of the [MAX_DRIVER_COUNT](#) parameter to a positive integer value to limit the extraction of drivers in fan-in cone to a specified value.

Use the [ignore_inout](#) parameter to yes to not report any violation for the inout terminals.

Message Details

Verilog

The following message appears at the location where the signal `<sig-name>` is not a tristate signal but has multiple drivers:

Non-tristate signal '`<sig-name>`' has multiple drivers

VHDL

The following message appears at the location of a signal declaration `<sig-name>` that has multiple drivers:

signal '`<sig-name>`' has multiple drivers

Severity

Mandatory

Verilog Examples

Example 1

Consider the following example where the signal `OUT[3]` has multiple drivers (IN1 and IN2):

```
module test (IN1, IN2, CLK, OUT);
```

```

input [7:0] IN1;
input IN2, CLK;
output reg [7:0] OUT;

always@(posedge CLK)
    OUT = IN1;
always @(posedge CLK)
    OUT[3] = IN2;
endmodule

```

For this example, SpyGlass generates the following message:

Non-tristate signal 'test.OUT[3]' has multiple drivers

Example 2

Now consider the following example where the tristate signal `ts[2]` signal has multiple drivers:

```

module test (IN1, IN2, CLK, OUT);
    input IN1, CLK;
    input [3:0] IN2;
    output OUT;
    reg [7:0] ts;
    reg en;
    always (en or IN1 or IN2)
    begin
        ts[2] = en ? IN1 : 1'bz;
        ts[4:1] = en ? IN2 : 4'bzzzz;
        ts[5:2] = en ? IN2 : 4'bzzzz;
    end
endmodule

```

The *STARCO5-2.5.1.5a* tag does not report any violation in this example.

Example 3

Consider the following example:

```

module tristate (input in1, in2, in3, in4, in5, output out1,
out2);
wire a1;
orr or_1( .in1(in1), .in2(in2), .out(a1));
orr or_2( .in1(in3), .in2(in4), .out(a1));

```

STARC05 Tags

```
y y_1( .in1(a1), .in2(in5), .out(out1));
endmodule
module orr (input in1, in2, output out);
endmodule
```

The *STARC05-2.5.1.5a* tag reports a violation for the net a1, when the [assume_driver_load_srp](#) parameter is set to yes.

Example 4

Consider the following example:

When `-strict=yes`

```
module test(input in , output out1 , inout out2 );
assign out1 = in;
assign out2 = in; // Violation
endmodule
```

When `-strict=no` (default flow)

```
module test(input in , output out1 , inout out2 );
assign out1 = in;
assign out2 = in; // No Violation
endmodule
```

The *STARC05-2.5.1.5a* tag reports a violation when the `-strict` parameter is set to yes and does not report violation when the `-strict` parameter is set to no.

STARC05-2.5.1.7

Do not use tristate output in the conditional expression of an if statement

When to Use

Use this tag to detect tristate outputs that are used in an if construct conditional expression in the same module.

Description

The STARC05-2.5.1.7 tag reports violation for the conditional expression of if statement only inside the same module.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location of an if construct condition expression when it uses one or more bits of a tristate output *<name>*:

[RECOMMENDED2] Tri State output '<name>' bit(s) used in the conditional expression of if statement

Potential Issues

Violation may arise when a tristate signal is used in a conditional expression inside the same module.

Consequences of Not Fixing

For tristate signals, propagation of x should be considered in order to match simulation results in RTL and gate-level. If a signal value that includes z is used in a conditional expression of if statements, the else part

gets executed. Normally, a value z becomes x after passing through a logic gate, but in this case it becomes a fixed value. This may result in inconsistencies between RTL and gate level.

How to Debug and Fix

The violation message shows the tristate output name. Double-click the violation message. The HDL Viewer window highlights the if condition where the tristate output is used in the conditional expression of the if statement.

To fix the violation, avoid using tristate signals for conditional expression of if statements.

Example Code and/or Schematic

Example 1

The STARC05-2.5.1.7 tag reports a violation in the following Verilog example in which tristate signal, w , is used in the if condition:

```
module m (in1, in2, en1, en2, out);
    input [7:0] in1, in2;
    input en1, en2;
    output out;
    reg [3:0] out;
    reg a;
    assign w = in1 ? 1'bz: 1'bz;
    always @ (w) begin
        if (w)
            out = 1'b1;
        else out = 1'b0;
    end
endmodule
```

Hence, you should avoid using tristate signals for conditional expression of if statements.

Example 2

Consider the following VHDL example:

```
library IEEE;
use IEEE.std_logic_1164.all;
```

```
entity test11 is
  port(
    in1: in std_logic_vector(2 downto 0);
    en: in std_logic;
    out2_z: out std_logic_vector(2 downto 0)
  );
end test11;

architecture behav of test11 is

  signal out1: std_logic_vector(2 downto 0);
begin
  process(in1, en)
  begin
    if (en = '0') then
      out1 <= in1;
      out2_z <= "0XZ";
    else
      out2_z <= "1ZZ";
      out1 <= "00Z";
    end if;

    if(out1(1 downto 0) = "11") then
      out2_z <= "1ZZ";
    end if;
    if(out1(0) = '1') then
      out2_z <= "1ZZ";
    end if;
    if(out1 = "11") then
      out2_z <= "1ZZ";
    end if;

  end process;
end behav;
```

In the above example, the STARCO5-2.5.1.7 tag reports a violation as the following tristate outputs are used in the conditional expression of `if` statement:

- out1(1 downto 0)
- out1(0)
- out1

Example 3

Consider the following example:

```
module TOP(input in1, input in2, input sel, output out);
wire tri_state_output;
reg out;
tri_state U001 (.in(in1), .sel(sel), .out(tri_state_output));
always @(tri_state_output)
begin
    if (tri_state_output)
        begin
            out = in2;
        end
    end
endmodule
module tri_state(input in, input sel, output out);
assign out = (sel == 1) ? in : 1'b0;
endmodule
```

In the above example the STARC05-2.5.1.7 tag will not report a violation as the `tri_state_output` net is a tristate net which comes from the other module `tri_state`.

Default Severity Label

Recommended2

tag Group

RTL Description Technique

Reports and Related Files

No related reports or files

How to Debug

The violation message shows the tristate output name.

Double-click the violation message. The HDL Viewer window highlights the if condition where the tristate output is used in the conditional expression of the if statement.

Severity

Recommended2

Examples

Verilog

The STARCO5-2.5.1.7 tag reports a violation in the following example in which tristate signal, w, is used in the if condition:

```
module m (in1, in2, en1, en2, out);
  input [7:0] in1, in2;
  input en1, en2;
  output out;
  reg [3:0] out;
  reg a;
  assign w = in1 ? 1'bz: 1'bz;
  always @ (w) begin
    if(w)
      out = 1'b1;
    else out = 1'b0;
  end
endmodule
```

Hence, you should avoid using tristate signals for conditional expression of if statements.

VHDL

Consider the following example:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity test11 is
  port(
    in1: in std_logic_vector(2 downto 0);
    en: in std_logic;
    out2_z: out std_logic_vector(2 downto 0)
```

STARCO5 Tags

```

        );
end test11;

architecture behav of test11 is

signal out1: std_logic_vector(2 downto 0);
begin
    process(in1, en)
    begin
        if (en = '0') then
            out1 <= in1;
            out2_z <= "0XZ";
        else
            out2_z <= "1ZZ";
            out1 <= "00Z";
        end if;

        if(out1(1 downto 0) = "11") then
            out2_z <= "1ZZ";
        end if;

        if(out1(0) = '1') then
            out2_z <= "1ZZ";
        end if;

        if(out1 = "11") then
            out2_z <= "1ZZ";
        end if;

    end process;
end behav;

```

In the above example, the STARCO5-2.5.1.7 tag reports a violation as the following tristate outputs are used in the conditional expression of `if` statement:

- out1(1 downto 0)
- out1(0)
- out1

STARC05-2.5.1.8

Tristate output should not be used in the selection expression of a case statement that assigns a fixed value in others choice

Description

The STARC05-2.5.1.8 tag flags tristate outputs used in case construct select expression when the default/OTHERS clause assigns a fixed value.

For tristate signals, propagation of x should be considered in order to match simulation results in RTL and gate-level. When a signal is entered in a selection expression of a case statement that is not assigned 'x' in default clause, 'x' will not propagate.

Avoid using tristate signals for selection expression of case statements.

Message Details

Verilog

The following message appears at the location of a case construct select expression that uses one or more bits of tristate signal *<name>* when the default clause of the case construct assigns a fixed value only:

Tri State output '<name>' bit(s) used in the selection expression of case statement

VHDL

The following message appears at the location of a case construct select expression that uses the tristate signal *<name>* when the OTHERS clause of the case construct assigns a fixed value only:

Tristate output '<name>' used in the selection expression of a case statement

Severity

Recommended

STARC05-2.5.1.9

Do not enter a tristate output in the selection expression of casex and casez statement.

When to Use

Use this tag to detect casex and casez constructs that use tristate outputs in the selection expression.

Description

The STARC05-2.5.1.9 reports violation for casex/casez constructs where tristate signals are used in the case selection expressions.

Default Weight

5

Language

Verilog, VHDL

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the location of a casex/casez construct whose select expression has a tristate signal <name>:

[RECOMMENDED2] Tri State output '<name>' bits(s) used in the selection expression of casex or casez statement

Potential Issues

Violation may arise when a tristate signal is used in the selection expression of a casex or casez statement.

Consequences of Not Fixing

The value, Z, is usually treated as a don't care value, and propagation as X is not assured across all the tools. This can potentially result in a mismatch

between gate-level and RTL-level simulations.

How to Debug

Double-click the violation message. The HDL Viewer window highlights the select expression of the casex/casez where tristate output is used in the select expression of the casex/casez statement.

View the Incremental Schematic of the violation message. The Incremental schematic displays the offending tristate output net.

Although most of the current tools can handle this situation correctly, but it is generally advisable to avoid such a scenario to ensure high quality and portable design.

Example Code and/or Schematic

Consider the following example:

```
module top (z, a, b, c);
output [2:0] z,c;
input [1:0] a;
input [2:0] b;
reg ci_var;
reg [2:0]c,z;
  always @ (a or b) begin

    if (a)
      z = 4'bz;
    else
      c = 4'bz;

  end
always @ (a or b)
begin
  casex (z)
    a * b      : ci_var = 0;
    a[0] + b[2] : ci_var = 0;
    a          : ci_var = 1;
    default:    ci_var = c;
  endcase
```


STARCO5 Tags

```
end
```

```
endmodule
```

In the above example the STARCO5-2.5.1.9 tag reports a violation as the tristate output, `z`, is used in a `casex` or `casez` selection.

Default Severity Label

Recommended

tag Group

RTL Description Technique

Reports and Related Files

No related reports or files

STARC05-3.3.1.4b

A flip-flop should have an asynchronous set or an asynchronous reset

Description

The STARC05-3.3.1.4b tag flags flip-flop descriptions that have neither asynchronous set nor asynchronous reset. The STARC05-3.3.1.4b tag also flags set/reset signals that are tied to supply or ground pins.

All flip-flops used in the design have initial set/reset pins for initializing the state of the design. If a flip-flop is not controllable from external pin, it would have an undefined value after initialization and may produce undesirable results.

The STARC05-3.3.1.4b tag does not report violation for the following cases:

- Flip-flops whose output is unconnected
- Gate-level cells, if all the outputs of a flip-flop are unconnected

Parameter(s)

[*flag_unconnected_out_flops*](#): Default value is no. Set the value of the parameter to yes to report violation when flop output is unconnected.

Message Details

The following message appears at the location where the output of a flip-flop *<flop-name>* (in hierarchy *<hier-path>*) that is described without either asynchronous set or asynchronous reset, first assigned:

Flip-flop '*<flop-name>*' has neither asynchronous set nor asynchronous reset. [Hierarchy: '*<hier-path>*']

Severity

Recommended

Verilog Examples

Consider the following example where a flip-flop OUT has been described without a set or reset line:

STARCO5 Tags

```
module test (OUT, IN, CLK);  
  input IN, CLK;  
  output OUT;  
  
  reg OUT;  
  
  always @ (posedge CLK)  
    OUT <= IN;  
endmodule
```

For this example, SpyGlass generates the following message:

Flip-flop 'OUT' has neither asynchronous set nor asynchronous reset.[Hierarchy: 'test']

STARC05-3.3.3.1

A clock must not be connected to the D input of a FF

Description

The STARC05-3.3.3.1 tag flags clock signals that are connected to data pins of flip-flops.

The STARC05-3.3.3.1 tag supports block path analysis, that is, it supports constant value propagation. Therefore, this tag does not traverse on the blocked path. The [set_case_analysis](#) constraint is used to set the initial values for block path analysis. For black boxes, this tag traverses further only if the [assume_path](#) constraint is specified in the SGDC file. This tag does not flag a violation for the clock driven by a constant value, which can be propagated through an assignment or using an SGDC file.

Inputting a clock into the input of a flip-flop may cause a problem for the ATPG tool. Generally, the ATPG tool generates a test pattern while performing the simulation with zero delays. Therefore, inputting a clock into the D input of a flip-flop runs a risk of generating an incorrect test pattern because of the racing problem.

Message Details

The following message appears at the location where a clock *<clk-name>* is connected to the data pin of a flip-flop:

Clock '*<clk-name>*' is connected to input pin of a flip-flop

Severity

Mandatory

STARC05-1.1.4.6b

Fixed value should not be connected to the input ports

Description

The STARC05-1.1.4.6b tag flags input ports of a module that are connected to a fixed value.

After synthesis optimization, ports that are directly connected to fixed value may become unconnected ports. This situation may cause problems during logic equivalency checks.

The STARC05-1.1.4.6b tag traverses from the input port of the module and ignores buffers and inverters during the fan-in traversal.

The STARC05-1.1.4.6b tag keeps track of the number of inverters during fan-in traversal and reports the actual value, `tiedhigh` or `tiedlow`, reaching the input port.

Message Details

The following message appears at the location where input port `<port-name>` of instance `<inst-name>` is connected to a fixed value `<value>` (tied high or tied low):

Input '`<port-name>`' of `<module-name>` [Instance `<inst-name>`] is tied `<value>`

where `<value>` can be `tiedhigh` or `tiedlow`.

Severity

Recommended1

STARC05-3.3.6.2

Do not mix clock lines and reset lines

Description

The STARC05-3.3.6.2 tag flags clock signals and reset signals that have logic in common.

In circuits where clock lines and reset lines are mixed, all flip-flops to which the clock signal is connected would be excluded from scanning. Therefore, mixing of clock lines and reset lines should be avoided.

The STARC05-3.3.6.2 tag does not require that the clock signal and reset signal belong to the same flip-flop. Any pair of clock signal and reset signal is flagged if they have common logic.

Parameter(s)

- [*allow_combo_logic_base*](#): Default value is `no`. Set this parameter to `yes` to ignore combinational logic in the clock/reset path.
- [*report_internal_netname*](#): Default value is `no`. Set this parameter to `yes` to enable the tag to display the hierarchical names of the violating internal nets.

Message Details

The following message appears when a clock signal *<clk-name>* and a reset signal *<rst-name>* have common logic:

Clock '*<clk-name>*' and reset '*<rst-name>*' have logic in common

Example

Consider the following example:

```
module DFF(input CLK, CLR_N_DFF, D, RST,output reg Q);
    always @(posedge CLK or negedge CLR_N_DFF or negedge
RST) begin
        if (CLR_N_DFF == 1'b0)
            Q <= 1'b0;
        else if (RST == 1'b0 )
            Q <= 1'b1;
```

STARCO5 Tags

```

        else
            Q <= D;
        end
    endmodule

module DUT(input CLK, CLR_N, RST, D1, D2, output Q1, Q2);
    DFF dff1(.CLK(CLK), .CLR_N_DFF(CLR_N), .D(D1),
        .Q(Q1), .RST(RST));

    DFF dff2(.CLK(CLR_N), .CLR_N_DFF(1'b1), .D(D2),
        .Q(Q2), .RST(RST));
endmodule

```

For the above example, the tag reports this following message:

clock 'DUT.CLR_N' and reset 'DUT.CLR_N' have logic in common

Severity

Mandatory

STARC05-3.1.3.2a

Ports should be declared in recommended order. (Verilog)

Description

The STARC05-3.1.3.2a tag flags module port declarations where the ports are not specified in the recommended order.

It is recommended to specify the input ports first, followed by output ports, and then inout ports. Within each port type, the ports should be specified as clock type ports first, followed by reset type ports, and then other ports.

Under the recommended port definition order, a violation message is generated for the following cases:

- If every input port specified after an output or inout port
- If every output port specified after an inout port

Within each port type, a violation message is generated for the following cases:

- If every clock type port specified after a reset type port or a port that is neither a clock nor a reset
- If every reset type port specified after a port that is neither a clock nor a reset.

Specifying ports in a consistent order aids understanding and reduces the likelihood of disconnects.

NOTE: *The STARC05-3.1.3.2a tag is not checked if a module is empty. Port order of such modules is checked by the [STARC05-3.1.3.2b](#) tag.*

Message Details

The following message appears at the location of a module port map where the port `<port-name>` is not defined in the recommended order:

Port '`<port-name>`' not defined in recommended order

Severity

Recommended2

Suggested Fix

Reorder port definitions to comply with this convention.

STARC05-3.1.3.2b

Ports should be declared in recommended order. (Verilog)

Description

The STARC05-3.1.3.2b tag flags port maps in empty modules, where ports are not specified in recommended order.

It is recommended to specify the input ports first, then output ports, and then the inout ports.

Under the recommended port definition order, a violation message is flagged in the following cases:

- If every input port is specified after an inout or output port
- If every output port is specified after an inout port

Specifying ports in a consistent order aids understanding and reduces the likelihood of disconnects.

Message Details

The following message appears at the location of a module port map where the port *<port-name>* is not defined in the recommended order:

Port '*<port-name>*' not defined in recommended order

Severity

Recommended2

Suggested Fix

Reorder port definitions to comply with this convention.

Starcad-21 Tags

This section explains the Starcad-21 tags in the VC SpyGlass Lint.

ALLOW_CELL

Allowed cells should be used in the design

Language

Verilog+VHDL

Description

The *ALLOW_CELL* tag flags instantiations of cells other than the allowed cells.

The *ALLOW_CELL* tag requires you to specify the names of the allowed cells in a control file that is specified by using the *matchtype* tag parameter. Then, the *ALLOW_CELL* tag flags the instantiations of the cells other than the specified cells.

Message Details

The following message appears at the location where a non-allowed cell *<cell-name>* is instantiated:

cell <cell-name> should not be used.

tag Severity

ERROR

CHIP_MODULE_NAME_LENGTH

The name lengths of chip-level modules should not be longer than 16

Language

Verilog+VHDL

Description

The *CHIP_MODULE_NAME_LENGTH* tag flags chip-level design unit names that are longer than 16 characters.

Message Details

The following message appears for chip-level design unit name *<du-name>* that is longer than 16 characters:

Top level module name <du-name> is longer than 16.

tag Severity

Info

BOTH_SETRESET

Flip-flops or latches must not have both asynchronous set and asynchronous reset.

Language

Verilog+VHDL

Description

The `BOTH_SETRESET` Tag reports the flip-flops or latches that have both asynchronous set and asynchronous reset pins.

The `BOTH_SETRESET` Tag does not report flip-flops that have both the set and reset pins and any one of them is inactive.

Message Details

The following message appears for the flip-flop or latch `<output-name>` that has both asynchronous set and asynchronous reset descriptions:

Use of Flip-flop/Latch with Set/Reset `<output-name>` is prohibited.

Severity

Warning

CHIP_PORT_NAME_LENGTH

The name lengths of chip-level ports should not be longer than 16

Language

Verilog+VHDL

Description

The *CHIP_PORT_NAME_LENGTH* tag flags the chip-level port names that are longer than 16 characters.

Message Details

The following message appears for chip-level port name *<port-name>* that is longer than 16 characters:

Top level port name *<port-name>* is longer than 16.

tag Severity

Info

DISALLOW_CELL

Disallowed cells that are specified by user are used in the design

Language

Verilog + VHDL

Description

The *DISALLOW_CELL* tag flags disallowed cells instantiated in the design.

The *DISALLOW_CELL* tag requires you to specify the names of the disallowed cells in a control file that is specified by using the *matchtype* tag parameter. Then, the *DISALLOW_CELL* tag flags the instantiations of the specified cells.

NOTE: *The DISALLOW_CELL tag supports the generate-for block.*

Message Details

The following message appears at the location where disallowed cell *<cell-name>* is instantiated:

cell <cell-name> should not be used.

tag Severity

Error

RESERVE_NAME

VDD/VSS/VCC/GND/VREF are used as the name of modules/instances/nets/ports

Language

Verilog+VHDL

Description

The *RESERVE_NAME* tag flags object names that contain special signal names.

The *RESERVE_NAME* tag flags design unit, instance, port, and net names that contain any of the strings VDD, VSS, VCC, or VREF.

The *RESERVE_NAME* tag is case insensitive.

By default, the *RESERVE_NAME* tag flags objects names that are VDD, VSS, VCC, or VREF.

Set the *matchtype* tag parameter to *forward* to have the *RESERVE_NAME* tag flag those object names that start with VDD, VSS, VCC, or VREF. Set the *matchtype* tag parameter to *backward* to have the *RESERVE_NAME* tag flag those object names that end with VDD, VSS, VCC, or VREF.

Message Details

The following message appears for object name *<name>* that contains VDD, VSS, GND, VCC, or VREF:

Name *<name>* contains VDD/VSS/GND/VCC/VREF.

tag Severity

Error

CELL_INFO

Informations of all cell names and the number of each cell

Description

The *CELL_INFO* tag reports name and the number of instances of each cell instantiated in the design.

Message Details

The following message appears for cell *<cell-name>* that is instantiated *<num>* times in the design:

'*<num>*' Instances of the master '*<cell-name>*' exists in the circuit.

Tag Severity

Info

MODULE_PORTS

Information about number of ports in each module

Description

The MODULE_PORTS tag is a prerequisite tag and reports the number of ports in each module.

Tag Severity

Info

Timing Tags

This section explains the Timing Tags in VC SpyGlass Lint.

DeepMux

Deep MUX detected - potential performance problem

Description

The *DeepMux* Tag flags nested if statements that have more than the specified number of levels and can be rewritten as a case construct.

Nested if statements infer priority encoders that can impact performance, particularly if the priority encoder is deep. Using a case statement will cause a parallel structure to be inferred, which will, in general, result in a higher performance path.

Tag Arguments

By default, the *DeepMux* Tag flags nested if statements that are more than 4 levels deep. Use the *bigmux_max* Tag parameter to specify a different number.

Message Details

The following message appears at the first line of a deeply nested if statement construct:

Deeply nested 'if' looks like it might be better written as a case statement

Tag Severity

Warning

Verilog Examples

Consider the following example where the multiplexer construct (with a [2:0] as the select expression) has 5 levels:

```
module test(clk);  
    input clk;  
  
    reg [2:0] a, b, c;  
  
    always @ (posedge clk)  
        begin
```

Timing Tags

```

    if( a[2:0] == c[2:0])
        b[2:0] = c[2:0];
    else if ( a[2:0] == b[2:0])
        c[2:0] = b[2:0];
    else if ( a[2:0] == (b[2:0] + c [2:0] ))
        a[2:0] = 3'b000;
    else if ((a[2:0]==3'b111) || (a[2:0]==3'b000))
        b[2:0] = 3'b101;
    else if ( ( a[2:0] == 3'b101) )
        b[2:0] = 3'b001;
    end
endmodule

```

For this example, SpyGlass generates the following message:

Deeply nested 'if' looks like it might be better written as a case statement

To fix this message, rewrite the multiplexer construct using the case construct as follows:

```

module test(clk);
    input clk;

    reg [2:0] a, b, c;

    always @ (posedge clk)
        begin
            case (a[2:0])
                c[2:0] :
                    b[2:0] = c[2:0];
                b[2:0] :
                    c[2:0] = b[2:0];
                b[2:0] + c[2:0] :
                    a[2:0] = 3'b000;
                3'b000, 3'b111 :
                    b[2:0] = 3'b101;
                3'b101 :
                    b[2:0] = 3'b001;
            endcase
        end
end

```

```
endmodule
```

VHDL Examples

Consider the following example where the multiplexer construct (with `in1` as the select expression) has 5 levels:

```
library ieee;
use ieee.std_logic_1164.all

architecture logic of test is
  signal s : bit;
  begin
    process ( s )
      variable I1 : integer;
      begin
        if in1 = 1 then I1 := 0;
        elsif (in1 = in3) then I1 := 1;
        elsif (in2 + in3 = in1) then I1 := 2;
        elsif (0 = in1) then I1 := 3;
        elsif (in1 = in2) then I1 := 4;
        end if;
      end process;
    end logic;
```

For this example, SpyGlass generates the following message:

Deeply nested 'if' looks like it might be better written as a case statement

To fix this message, rewrite the multiplexer construct using the case construct as follows:

```
architecture logic of test is
  signal s : bit;
  begin
    process ( s )
      variable I1 : integer;
      begin
        case (in1) is
          when (1)           => I1 := 0;
          when (in3)         => I1 := 1;
```

Timing Tags

```
        when (in2 +in3) => I1 := 2;  
        when (0)        => I1 := 3;  
        when (in2)      => I1 := 4;  
    end case;  
end process;  
end logic;
```

ClockEnableRace

Reports a race condition between clock and enable pins of a flip-flop

When to Use

Use this Tag to detect a race condition between clock and enable pins of a flip-flop.

Description

The *ClockEnableRace* Tag reports the flip-flops in which the clock and enable pins are driven by flip-flops that are triggered by the same clock signal. In such cases, a race condition exists between clock and enable signals of a flip-flop.

A common clock is detected by traversing backward from clock and enable pins of the flipflop. In the traversal, the Tag marks the components passed by two separate functions.

While it works at RTL description level, the *ClockEnableRace* Tag reports design layout-based clock-enable race conditions rather than simulation-related race conditions.

Parameters

- *cell_check_for_race*: Default value is NULL. Specify a comma-separated list of cell names specified in the following format to report a race condition between clock pins and enable pins of custom cells as well.:
'<cell name1>:<pin1>,<pin2>;<cell name2>:<pin1>,<pin2>'
- *checkcomblogic*: Default value is no. This indicates the *ClockEnableRace* Tag does not check for the combinational cells between clock and enable paths. Set this parameter to yes to check for such cases.
- *ignore_hanging_flop*: Default value is no. When you set this parameter to yes, the Tag does not report a violation for the flip-flop if it's output is hanging.
- *ignore_icg_cell*: Default value is not set. Set this parameter to the ICG cell name(s) to not report race through those ICG cells.
- *report_common_source*: Default value is no. Set the value of the parameter to yes to report violations when the clock and enable pin of the flip-flop is connected to a common source.

Constraints

None

Messages and Suggested Fix

The following message appears at the first line of a module that contains the Tag-violating flip-flop `<flop-name>`:

[WARNING] Potential Race between Clock and Enable for flop
<flop-name>

Potential Issues

This violation appears if your design contains a flip-flop in which the clock and enable pins are driven by flip-flops that are triggered by the same clock signal.

Consequences of Not Fixing

The race condition between clock and enable pins of a flip-flop may lead to potential timing issues in your design.

How to Debug and Fix

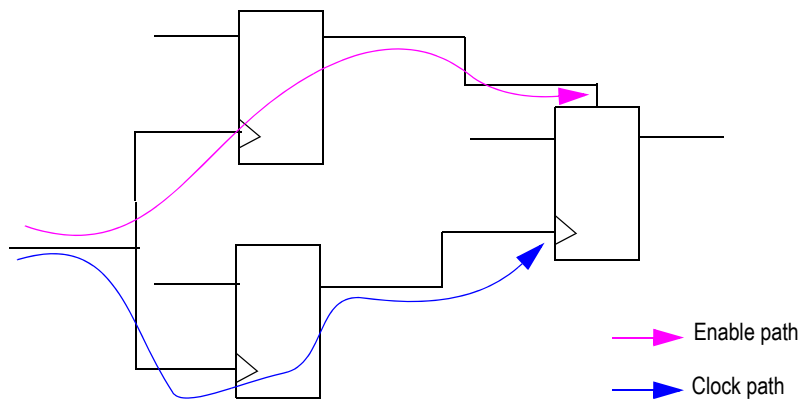
View the Incremental Schematic of the violation message. It shows two paths originating from one common point. The two paths are highlighted in two different colors. One path reaches the `Clock` terminal of a flip-flop. Another path reaches the `Enable` terminal of the same flip-flop. This indicates the race condition.

Fix the problem by ensuring that the clock and enable signals are not generated from flip-flops that are triggered by the same clock signal.

Example Code and/or Schematic

Example 1

In the following figure, the timing paths, based on layout, to the clock and enable pins of a flip-flop may be approximately same and therefore may result in a clock-enable race:



Example 2

Consider the following example where the clock signal w2 and the enable signal w1 of the flip-flop out are driven by flip-flops temp1 and temp2, respectively, that are triggered by the same clock signal clk1:

```
module test ( q1, q2, clk1, clk2, out);
  input q1, q2, clk1, clk2;
  output out;

  reg out;
  reg temp1, temp2;
  wire w1, w2;

  always @(posedge clk1)
    temp1 <= q1;

  assign w1 = temp1;

  always @(posedge clk1)
    temp2 <= q2;

  assign w2 = temp2;

  always @(posedge w2)
    if (w1)
```

Timing Tags

```
        out <= w1;  
    endmodule
```

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

LogicDepth

Reports logic paths where depth exceeds the specified number of levels

When to Use

Use this Tag to detect logic paths where the number of delay levels exceeds the specified value. The logic paths detected are those that originate or terminate at flip-flops or primary pins.

Description

The *LogicDepth* Tag reports logic paths, which originate or terminate at flip-flops or primary pins, where the number of delay levels exceeds the specified value. This Tag performs a simplified version of the critical path analysis.

For a library cell in the design, the Tag behavior is as follows:

- If any library cell is found in a path, the depth for that cell is 1.
- If a library cell has more than one input, all inputs to this cell are considered as different paths and cell depth is 1 for each path.
- If a cell is a flip-flop, the cell is treated as an end point for that particular path and no depth for the cell is counted to calculate the logic depth.

Delay Depth Calculation

The *LogicDepth* Tag calculates the delay depths of each type of logic gate as described in the following table:

Gate Type	Delay ^a	Associated Tag Parameter	Synthesis Primitive
Buffer	1	delay_Buf	(RTL INT)_BUF
Tristate buffer	1	delay_Tribuf	(RTL INT)_BUFIZ
Inverter	1	delay_Inv	(RTL INT)_NOT
N-input AND	$\log(\$size)/\log(2)$	delay_And	(RTL INT)_AND, M_RTL_AND_
N-input NAND	$\log(\$size)/\log(2)$	delay_Nand	(RTL INT)_NAND
N-input OR	$\log(\$size)/\log(2)$	delay_Or	(RTL INT)_OR, M_RTL_OR_

Timing Tags

Gate Type	Delay ^a	Associated Tag Parameter	Synthesis Primitive
N-input NOR	$\log(\text{\$size})/\log(2)$	delay_Nor	(RTL INT)_NOR
N-input XOR	$(\text{\$size}/4)+1$	delay_Xor	(RTL INT)_XOR, M_RTL_XOR_
N-input XNOR	$(\text{\$size}/4)+1$	delay_Xnor	(RTL INT)_XNOR
MUX with N select lines of which only one is active at a time and N inputs	2	delay_Mux	(RTL INT)_MUX
LogN MUX with N select lines and 2^N inputs	$(\text{\$size}/4)+1$	delay_LogN_mux	M_RTL_MUX_, M_RTL_SELECT_BOX_,
Decode operation, decoding from N bits to 2^N bits	$\log(\text{\$size})/\log(2)+1$	delay_Decoder	M_RTL_DEC_
Latch	2	delay_Latch	(RTL INT)_LD
Unsigned full-adder with 2 N-bit inputs, plus carry-in and N+1-bit output plus carry-out	$\text{\$size}*2$	delay_Add_unsigned	M_RTL_ADD_UN_
Binary subtract (a-b) where a and b are both N-bit	$\text{\$size}*2$	delay_Subtract_binary	M_RTL_SUB_BIN
Unary subtract (-a) where a is N-bit	$\text{\$size}*2$	delay_Subtract_unary	M_RTL_SUB_UNARY_
Incrementor (a+1) where a is N-bit	$\text{\$size}*2$	delay_Increment	M_RTL_INCR_
Decrementor (a-1) where a is N-bit	$\text{\$size}*2$	delay_Decrement	M_RTL_DECR_
Fixed-point multiplier (a*b) where a and b are both N-bit	$\text{\$size}*4$	delay_Multiply_unsigned	M_RTL_MULT_UN_
Fixed-point multiplier (a*b) where a and b are both N-bit	$\text{\$size}*4$	delay_Multiply_signed	M_RTL_MULT_SIGN_

Gate Type	Delay ^a	Associated Tag Parameter	Synthesis Primitive
Unsigned fixed-point greater-than comparison (a>b) where a and b are both N-bit	\$size*2	delay_GT_unsigned	M_RTL_GT_UNSIGN
Signed fixed-point greater-than comparison (a>b) where a and b are both N-bit	\$size*2	delay_GT_signed	M_RTL_GT_SIGN
Unsigned fixed-point greater-than/equals comparison (a>=b) where a and b are both N-bit	\$size*2	delay_GE_unsigned	M_RTL_GE_UNSIGN
Signed fixed-point greater-than/equals comparison (a>=b) where a and b are both N-bit	\$size*2	delay_GE_signed	M_RTL_GE_SIGN
Unsigned fixed-point less-than/equals comparison (a<=b) where a and b are both N-bit	(\$size/4)*2	delay_LEQ_unsigned	M_RTL_LEQ_UNSIGN
Signed fixed-point less-than/equals comparison (a<=b) where a and b are both N-bit	(\$size/4)*2	delay_LEQ_signed	M_RTL_LEQ_SIGN
Equality comparison (a==b) where a and b are both N-bit	\$size*2	delay_Equal	M_RTL_EQ
Non-equality comparison (a!=b) where a and b are both N-bit	\$size*2	delay_Not_equal	M_RTL_NEQ

Timing Tags

Gate Type	Delay ^a	Associated Tag Parameter	Synthesis Primitive
Shift operation ($a \gg b$ or $a \ll b$) where a and b are both N-bit	$\$size * 2$	<code>delay_Shift</code>	<code>M_RTL_LSHIFT_</code> , <code>M_RTL_RSHIFT_</code>
Absolute value evaluation operation ($abs(a)$) where a is N-bit	$((\$size - 1) * 2) + 1$	<code>delay_Abs</code>	<code>M_RTL_ABS_</code>

a. \$size is the number of inputs to a gate.

On the basis of your synthesized netlist, the logic depth values reported by VC SpyGlass Lint may differ from the expected values. It is because when you synthesize a source design using a synthesis tool or library, the generated netlist is different from the netlist generated by VC SpyGlass Lint.

For a better correlation between the reported and actual logic depth values, use the `delay_*` Tag parameters to tweak the VC SpyGlass Lint delay calculations to follow your synthesis tool or library. For example, VC SpyGlass Lint defines the logic depth value of an N-input AND gate as $\log_2 N$ assuming that the gate is realized as a cascade of 2-input AND gates. Therefore, VC SpyGlass Lint realizes an 8-input AND gate as three levels of 2-input AND gate cascade. However, depending on the richness of your synthesis library, the same 8-input AND gate may be realized differently. It may be realized as a single 8-input AND gate. Therefore, the expected logic depth value is different. Set the `delay_And` parameter to a value that follows your synthesis tool or library more closely to deal with the difference in realization for AND gates.

Parameters

- **`show_viol_count`**: Default value is -2. Set this parameter to any integer value to show the violation number as passed by the user for each start point. The following table describes the Tag behavior for negative values of this parameter:

Value	Tag Behavior
-1	Reports all the paths present in the design. This may result in noise. Therefore, it is not a recommended option.
-2	Reports the longest path per start point. It means if there are multiple paths from a start point, only the longest path is reported.
-3	Reports the longest path per end point. It means if there are multiple paths from an end point, only the longest path is reported.

- ***ldto and ldfrom***: By default, the *LogicDepth* Tag reports logic depth for paths between all combinations of flip-flops, primary ports, and memory elements. Use the *ldto and ldfrom* parameters to customize the *LogicDepth* Tag checking by specifying the paths that need to be checked. For these parameters to work, the clock information must be propagated.

For this, you need to specify clocks by using the *clock* SGDC constraint, in an SGDC file. Alternatively, you can use the *use_inferred_clocks* parameter to specify clock signal.

- ***delaymax***: By default, the *LogicDepth* Tag reports paths that have more than 40 delay levels. Set this parameter to a positive integer value to change the default maximum limit. The Tag, by default, reports the following types of logic paths that have more delay levels than the number specified by this parameter:

- ☐ From a primary input pin to a primary output pin
- ☐ From a primary input pin to a flip-flop data pin
- ☐ From a flip-flop output pin to a flip-flop data pin
- ☐ From a flip-flop output pin to a primary output pin

The *LogicDepth* Tag assumes instances of library cells with the *memory()* attribute set as memory elements.

- ***delaymax_memtoflop***: Default value is -1. This indicates the value of this parameter is same as that specified for the *delaymax* parameter. Set this parameter to a positive integer to specify a different value for the maximum number of logic levels. Beyond this limit, the *LogicDepth* Tag reports a violation while checking paths from a memory element output pin to a flip-flop input pin or a primary output.

Timing Tags

- ***delaymax_floptomem***: Default value is -1. This indicates the value of this parameter is same as that specified for the ***delaymax*** parameter. Set this parameter to a positive integer to specify a different value for the maximum number of logic levels. Beyond this limit, the *LogicDepth* Tag reports a violation while checking paths from a flip-flop output pin or a primary input pin to a memory element input pin.
- ***delaymax_memtomem***: Default value is -1. This indicates the value of this parameter is same as that specified for the ***delaymax*** parameter. Set this parameter to a positive integer to specify a different value for the maximum number of logic levels. Beyond this limit, the *LogicDepth* Tag reports a violation while checking paths from a memory element output pin to a memory element input pin.
- ***delaymax_err***: Default value is 200. This indicates the *LogicDepth* Tag reports an error message for all logic levels greater than 200. In addition, a warning message is reported for all logic levels ranging from 40 (default value of the *delaymax* parameter) to 200 (default value of the *delaymax_err* parameter). Set this parameter to a different positive integer to change the default setting.
- ***delaymax_inputtoflop***: By default, this parameter is set to the value of the *delaymax* parameter. Set this parameter to a positive integer to specify the maximum number of logic levels between a primary input pin and a flip-flop data pin.
- ***delaymax_inputtooutput***: By default, this parameter is set to the value of the *delaymax* parameter. Set this parameter to a positive integer to specify the maximum number of logic levels between a primary input pin and a primary output pin.
- ***delaymax_floptooutput***: By default, this parameter is set to the value of the *delaymax* parameter. Set this parameter to a positive integer to specify the maximum number of logic levels between a flip-flop output pin and a primary output pin.
- ***delaymax_floptoflop***: By default, this parameter is set to the value of the *delaymax* parameter. Set this parameter to a positive integer to specify the maximum number of logic levels between a flip-flop output pin to a flip-flop data pin.

Constraints

None

Messages and Suggested Fix

Message 1

The following message appears at the location where the logic path from an object *<objs-type> <objs-name>* to an object *<objd-type> <objd-name>* that has more delay levels than the number specified by the *delaymax* parameter is first set:

[WARNING] *<path-type>*:Logic delay from *<objs-name>* to *<objd-name>* (*<num>* levels) exceeds allowed max (\$delaymax)

Where, the values are as follows:

Source Object Type (<i><objs-type></i>)	Destination Object Type (<i><objd-type></i>)	<i><path-type></i>
Primary input pin	Primary output pin	INPUT->OUTPUT
Primary input pin	Flip-flop data pin	INPUT->FLOP
Flip-flop output pin	Flip-flop data pin	FLOP-FLOP
Flip-flop output pin	Primary output pin	FLOP->OUTPUT

Potential Issues

For potential issues, click [Potential Issues](#).

Consequences of Not Fixing

For consequences of not fixing this violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For information on debugging, click [How to Debug and Fix](#).

Message 2

The following message appears at the location where the logic path from an object *<objs-type> <objs-name>* to an object *<objd-type> <objd-name>* that has more delay levels (*<num>*) than the number specified by the *delaymax_floptomem* parameter is first set:

[WARNING] *<path-type>*:Logic delay from *<objs-name>* to *<objd-name>* (*<num>* levels) exceeds allowed max (\$delaymax_floptomem)

Where, the values are as follows:

Source Object Type (<i><objs-type></i>)	Destination Object Type (<i><objd-type></i>)	<i><path-type></i>
Flip-flop output pin	Memory element input pin	FLOP-MEM
Primary input pin	Memory element input pin	INPUT->MEM

Potential Issues

For potential issues, click [Potential Issues](#).

Consequences of Not Fixing

For consequences of not fixing this violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For information on debugging, click [How to Debug and Fix](#).

Message 3

The following message appears at the location where the logic path from an object *<objs-type>* *<objs-name>* to an object *<objd-type>* *<objd-name>* that has more delay levels (*<num>*) than the number specified by the *delaymax_memtoflop* parameter is first set:

[WARNING] *<path-type>*:Logic delay from *<objs-name>* to *<objd-name>* (*<num>* levels) exceeds allowed max (\$*delaymax_memtoflop*)

Where, the values are as follows:

Source Object Type (<i><objs-type></i>)	Destination Object Type (<i><objd-type></i>)	<i><path-type></i>
Memory element output pin	Flip-flop output pin	MEM->FLOP
Memory element output pin	Primary input pin	MEM->INPUT

Potential Issues

For potential issues, click [Potential Issues](#).

Consequences of Not Fixing

For consequences of not fixing this violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For information on debugging, click [How to Debug and Fix](#).

Message 4

The following message appears at the location where the logic path from a memory element `<mems-name>` to a memory element `<mem2-name>` that has more delay levels (`<num>`) than the number specified by the `delaymax_memtmem` parameter is first set:

[WARNING] MEM->MEM:Logic delay from `<mem1-name>` to `<mem2-name>` (`<num>` levels) exceeds allowed max (`$delaymax_memtmem`)

Potential Issues

For potential issues, click [Potential Issues](#).

Consequences of Not Fixing

For consequences of not fixing this violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For information on debugging, click [How to Debug and Fix](#).

Message 5

The following message appears when a clock `<clk-name>` specified using the `ldto` and `ldfrom` parameters is not found in the design:

[WARNING] Clock Domain `<clk-name>` not found

Potential Issues

These violations appear if your design contains logic paths, which originate or terminate at flip-flops or primary pins, where the number of delay levels exceeds the specified value.

Consequences of Not Fixing

The logic paths, which originate or terminate at flip-flops or primary pins, where the number of delay levels exceeds the specified value may lead to potential timing issues in your design. If a path has too many logic levels, it is difficult to meet timing goals even with aggressive synthesis optimization.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the paths that exceed the threshold logic depth.

Fix this problem by reviewing such paths for possible timing issues.

Example Code and/or Schematic

Example 1

Consider the following example:

```
configure_lint_tag_parameter -tag LogicDepth -parameter  
show_viol_count -value 5
```

The above declaration reports 5 violations for each start point.

Example 2

Consider the following example:

```
configure_lint_tag_parameter -tag LogicDepth -parameter  
show_viol_count -value 100
```

The above declaration reports 100 violations for each start point.

Example 3

Consider the following example:

```
configure_lint_tag_parameter -tag LogicDepth -parameter  
show_viol_count -value -1
```

The above declaration reports all violations for each start point.

Example 4

Consider the following example:

```
configure_lint_tag_parameter -tag LogicDepth -parameter  
show_viol_count -value -2
```

The above declaration reports the longest path for each start point.

Example 5

Consider the following example:

```
configure_lint_tag_parameter -tag LogicDepth -parameter  
show_viol_count -value -3
```

The above declaration reports the longest path for each end point.

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

LogNMux

Reports multiplexer constructs with select inputs more than the specified number

When to Use

Use this Tag to detect multiplexer constructs in which log of the number of select input pins is greater than the specified number.

Description

The LogNMux Tag reports multiplexer constructs in which log of the number of select input pins is greater than the specified number, that is, the `logmux_max` value that is 3 by default.

Detection of select inputs is dependent on data inputs and not on the select input width. This means the algorithm calculates the number of select inputs that would be needed to accommodate all data inputs, and that select input count is reported if it is found greater than the value specified for `logmux_max`.

By default, the Tag reports a violation when $\log(\text{number of select input pins in a LogN MUX}) > \text{Limit specified by the } \text{logmux\_max} \text{ parameter}$.

Parameters

- `check_mux_select_pins`: Default value is `no`. If you set this parameter to `yes`, the Tag also reports a violation when:
Number of select input pins in a LogN MUX > Limit specified by the `logmux_max` parameter

Refer to [Example 3](#) for an example related to this parameter.

NOTE: When the `check_mux_select_pins` parameter is not specified, the number of select inputs are identified from the number of data inputs that can accommodate the data. For example, if there are 21 data inputs, then the select pins would be considered as five, irrespective of select input width. The parameter has no effect for cases when violation has already been reported, that is, the select input pins are already exceeding the default value, as in the above case. Refer to [Example 5](#) for details.

- `show_connected_net`: Default value is `no`. This indicates that the LogNMux Tag does not show the immediate net connected to the mux.

Set the value of the parameter to *yes* to enable the *LogNMux* Tag to show the immediate net connected to the mux.

- ***strict***: Default value is *no*. This indicates the *LogNMux* Tag does not report MUXes inferred from variable used in bit-select or part-select. Set this parameter to *yes* or *LogNMux* to report such violations. Refer to [Example 4](#) for an example related to this parameter.
- ***logmux_max***: Default value is 3. This indicates the *LogNMux* Tag reports MUX constructs that have more than three select lines. Use this parameter to specify a different number.

Constraints

None

Messages and Suggested Fix

The following message appears at the location where the output *<sig-name>* of a LogN MUX with *<num>* select lines is first set when *<num>* is more than the value of the *logmux_max* parameter.

[WARNING] LogN mux (output *<sig-name>*) has *<num>* select inputs (greater than recommended limit of *\$logmux_max*)

Potential Issues

This violation appears if your design contains multiplexer constructs with more select lines than the specified number.

Consequences of Not Fixing

The multiplexer constructs with more select lines than the specified number creates congested decode and MUXing logic in the design. The congestion issues around MUX may lead to potential timing issues in your design.

How to Debug and Fix

View the Incremental Schematic of the violation message. It shows the violating multiplexer and its select inputs.

Fix this problem by reviewing the design and timing impact of the multiplexer with large select pins.

Example Code and/or Schematic

Example 1

In the following Verilog example, the output `mux.qout` of a LogN MUX has 4 input pins, which is greater than the recommended limit of 3:

```
module mux(din, sel, qout);
  input [7:0]din;
  input [3:0]sel;
  output qout;
  reg qout;
  always @(din or sel)
    case(sel)
      4'b0000 : qout = din[0];
      4'b0010 : qout = din[1];
      4'b0011 : qout = din[2];
      4'b0100 : qout = din[3];
      4'b0101 : qout = din[4];
      4'b0110 : qout = din[5];
      4'b0111 : qout = din[6];
      4'b1000 : qout = din[7];
      default : qout = 1'b0;
    endcase
endmodule
```

Example 2

In the following VHDL example, the output `mux_gate.SIG` of a LogN MUX has 4 input pins, which is greater than the recommended limit of 3:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity mux_gate is

  port (SEL: in STD_LOGIC_VECTOR (3 downto 0);
        A: in STD_LOGIC_VECTOR (7 downto 0);
        SIG: out STD_LOGIC);
end mux_gate;
```

```

architecture RTL of mux_gate is
begin
  SEL_PROCESS: process (SEL,A)
  begin
    case SEL is
      when "0000" => SIG <= A(0);
      when "0010" => SIG <= A(1);
      when "0100" => SIG <= A(2);
      when "0110" => SIG <= A(3);
      when "0101" => SIG <= A(4);
      when "0011" => SIG <= A(5);
      when "0111" => SIG <= A(6);
      when "1000" => SIG <= A(7);
      when others => SIG <= '0';
    end case;
  end process SEL_PROCESS;
end RTL;

```

Example 3

In the following Verilog example, the output `mux.qout` of a LogN MUX has 4 select input pins, which is greater than the recommended limit of 3:

```

module mux(din,sel,qout);
  input [3:0]din;
  input [3:0]sel;
  output qout;
  reg qout;
  always @(din or sel)
    case(sel)
      4'b0001 : qout = din[0];
      4'b0010 : qout = din[1];
      4'b0100 : qout = din[2];
      4'b1000 : qout = din[3];
    endcase
endmodule

```

For this example, by default, the *LogNMux* Tag does not report a violation. However, if the [check_mux_select_pins](#) parameter is set to `yes`, the Tag

reports a violation because it has 4 select input pins.

Example 4

Consider the following cases:

Case 1

```
wire [7:0] index;
mux_z = data [index +:6]
```

In the above scenario, the Tag reports a violation with the *strict* parameter.

Case 2

```
wire [7:0] index;
mux_z = data [index]
```

In the above scenario, the Tag reports a violation with the *strict* parameter.

Example 5

Consider the following example:

```
module mux4(din, sel, qout);
input [15:0]din;
    input [31:0]sel;
    output qout;
    reg qout;
    always @(din or sel)
    begin
        case(sel)
            1 : qout = din[0];
            2 : qout = din[1];
            3 : qout = din[1];
            4 : qout = din[2];
            5 : qout = din[3];
            6 : qout = din[4];
            7 : qout = din[5];
            8 : qout = din[6];
            9 : qout = din[7];
            default : qout = 1'b0;
        endcase
    end
endmodule
```

In the above example, the select input width is 32 bits and depending upon the data inputs, that is, nine including default, the number of select inputs are calculated as four. The [check_mux_select_pins](#) parameter has no effect in this case.

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

MaxFanout

Fan-out of a net exceeds a critical threshold

Description

The *MaxFanout* tag flags nets with fan-out exceeding the specified limit.

High fan-out nets are likely to have severe delay problems unless the nets are broken into buffered trees, which significantly complicates the physical design.

Arguments

By default, the *MaxFanout* tag flags nets with fan-out of more than 20. Use the *fanoutmax* parameter to specify a different number.

The *MaxFanout* tag ignores inverters in the fan-out of a net. It means the fan-out of an inverter found in the net fan-out is added to the fan-out of the net itself.

The *MaxFanout* tag considers the fan-out of nets at RTL level. However, some nets may be removed during synthesis optimizations. Therefore, the fan-out of a net in the netlist is generally less than its fan-out in RTL description. To take care of this situation, the *MaxFanout* tag scales the fan-out of RTL nets by assuming fan-out factor of 0.5 (default). Therefore, when a net is connected to 2 AND gates, the effective fan-out is 1 (0.5×2). Use the *fanoutmax* parameter to specify a different fan-out factor (generally less than 1.0).

The portion of the fan-out due to sequential elements is not scaled as these elements are not optimized out during synthesis.

It is recommended to exclude certain high fan-out net types, such as clock nets and reset nets, from detection using the *ignore_nets* parameter. Clock nets are handled specially in Place and Route tools, which insert buffering to manage timing. Reset nets generally do not require special buffering since they are not performance sensitive.

The *infertop* parameter allows specifying net names with the *ignore_nets* parameter without including the name of top-level design unit for net hierarchical names. If you use the *infertop* parameter, omit the top-level design unit name from the net names specified with *ignore_nets* parameter.

By default, the *infertop* parameter is set to yes.

Message Details

The following message appears at the location where a net `<net-name>` with fan-out `<num>` is first driven when the net fan-out exceeds the fan-out limit specified by the `fanoutmax` parameter:

Net `<net-name>` fanout (`<num>`) exceeds threshold `$fanoutmax`

Severity

Warning

ShiftReg

Large shift register detected

Description

The *ShiftReg* tag flags large shift registers.

Large shift registers are generally used to describe memories. Such constructs are inferred as connected individual flip-flops and are difficult to manage while achieving timing goals in physical design. Such constructs should be implemented using vendor-supplied memory macros.

NOTE: *This tag does not report some cases of circular shift registers.*

By default, the *ShiftReg* tag reports shift operations on signals or part-selects of signals of more than 20 bits. Use the *sr_max* parameter to specify a different number.

Message Details

The following message appears at the location where a shift operation of depth *<num>* on a signal *<sig-name>* is encountered:

Large shift register detected: last signal *<sig-name>*, depth *<num>*

Where, *<sig-name>* is the last signal bit found to be involved in the shift operation.

Severity

Warning

Verilog Examples

Consider the following example that describes large shift operations:

```
module test(in)
  input in;

  reg [25:0] r1, r2;
  wire [25:0] w;
  integer i;
```

```

always @(posedge in)
begin
    r1[23:2] = r1[24:3];
end

always @(posedge in)
begin
    for(i = 1; i < 24; i = i + 1)
        r2[i] = r2[i + 1];
    end
endmodule

```

The first always construct has a large shift operation on register `r1`. Therefore, VC SpyGlass Lint generates the following message at the location of the shift operation:

Large shift register detected: last signal test.r1[2], depth 22

The second always construct has a for loop describing a large shift operation on register `r2`. Therefore, VC SpyGlass Lint generates the following message at the first line of the always construct:

Large shift register detected: last signal test.r2[1], depth 23

VHDL Examples

Consider the following example that describes shift operations on very wide signals:

```

library ieee;
use ieee.std_logic_1164.all

entity test is
    port (in1, clk : in BIT; out1: out BIT);
end entity;

architecture arch_test of test is
    type array_type is array (0 to 25) of BIT;
    signal sig1 : array_type;

    begin
        process (clk, sig1)

```


Timing Tags

```
begin
    if(clk' event and clk = '1') then
        sig1(1 to 25) <= sig1(0 to 24);
    end if;
end process;
end architecture;
```

The `process` construct has a shift operation on a very wide array-type signal `sig1`. Therefore, VC SpyGlass Lint generates the following message at the first line of the `process` construct:

Large shift register detected: last signal test.sig1[0], depth 25

Netlist Tags

This section explains the Netlist Tags in VC SpyGlass Lint.

Rule_1-1

Checks for the floating primary port

Description

The tag reports violations when top level primary input/inout/output port is left unconnected.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when top level primary input/inout/output port is left unconnected :

[ERROR] Primary <input | output | inout> port '<port-name>' is left unconnected

Default Severity Label

Error

Tag Group

None

Reports and Related Files

None

Rule_1-2

Checks if top level primary port is not connected to the pad cell

Description

This tag reports violations when top level primary input/inout/output port is connected to the internal logic without going through the pad cell.

This tag now considers cell as PAD cell if `sec_class = "acore_with_pad"` attribute is set.

A pin is considered a pad pin when either of the following conditions are true:

- `sec_pin_type="pad"`
- `is_pad:true`
- `pad_cell:true`

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when top level primary input/inout/output port is connected to the internal logic without going through the pad cell:

[ERROR] Primary <input | inout | output> port '<port-name>' is connected to the internal logic without going through pad-cell

Default Severity Label

Error

Netlist Tags

Tag Group

None

Reports and Related Files

None

Rule_1-3

Checks if pad pin is connected properly

Description

The tag reports violations when input/output pad pin is connected to wrong primary port/pad pin. Input pad pin should be connected to the input primary port/pad pin. Output pad pin should be connected to the output primary port/pad pin.

A pin is considered a pad pin when either of the following conditions are true:

- `sec_pin_type="pad"`
- `is_pad:true`
- `pad_cell:true`

NOTE: *The tag does not report violation for inout pad pins and inout primary ports.*

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when input/output pad pin is connected to wrong primary port:

```
[ERROR] <input | output> pad pin '<pin-name>' (Inst:
'<instance-name>') is connected to <output | input> <port |
pad-pin> '<port-name | pad-pin-name>'
```

Default Severity Label

Error

Netlist Tags

Tag Group

None

Reports and Related Files

None

Rule_1-4

Checks if primary port is connected to multiple pad pins

Description

The tag reports violations when primary input/output/inout pad pin is connected to multiple pad pins.

A pin is considered a pad pin when either of the following conditions are true:

- `sec_pin_type="pad"`
- `is_pad:true`
- `pad_cell:true`

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when primary input/output/inout pad pin is connected to multiple pad pins:

[ERROR] Primary <input | input | output> port '<port-name>' is connected to multiple (<pad-pin-count>) pad pins

Default Severity Label

Error

Tag Group

None

Netlist Tags

Reports and Related Files

None

Rule_1-5

**Checks if inout pad pin is connected to a wrong inout primary port/
pad pin**

Description

This tag reports a violation when an inout pad pin is connected to the wrong primary port/pad pin. The inout pad pin should be connected to the inout primary port/pad pin.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when an inout pad pin is connected to a wrong primary port:

[ERROR] <inout> pad pin '<pin-name>' (Inst: '<instance-name>')
is connected to <inout> <port | pad-pin> '<port-name | pad-pin-
name>'

Default Severity Label

Error

Tag Group

None

Reports and Related Files

None

Rule_2

Detects open input terminals

When to Use

Use this tag to detect open input terminals.

Description

This tag reports violation for structurally unconnected nets/signals.

Default Weight

5

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when unconnected input terminal of library cells are found:

```
[ERROR] Detected undriven input terminal '<terminal-name>'
```

Default Severity Label

Error

Tag Group

Not Applicable

Reports and Related Files

No related reports or files.

Rule_2-1

Detects open inout terminal

When to Use

Use this tag to detect open inout terminals.

Description

The *Rule_2-1* tag reports violation for unconnected inout terminals of library cells.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when unconnected inout terminals of library cells are found:

[ERROR] Detected undriven inout terminal '<terminal-name>'

Default Severity Label

Error

Tag Group

Not Applicable

Reports and Related Files

None

Rule_3

Reports variable/signals that do not infer a tristate and have multiple simultaneous drivers

When to Use

Use this tag to identify variable/signals that do not infer a tristate and have multiple simultaneous drivers.

Tag Description

The *Rule_3* tag reports variables/signals that do not infer tristate nets and have multiple simultaneous drivers.

The *Rule_3* tag does not report violation for tristate nets whose immediate fan-in drivers are tristate elements. However, the tag reports a violation when an enable signal for two tristate elements is same as both the elements get enabled at the same point of time.

This tag reports variables/signals that are driven in more than one sequential always/process constructs, instances, and/or concurrent statements. Such cases are errors unless the variable infers a tristate net.

Language

Verilog

Tag Exceptions

Following are the exceptions to the *Rule_3* tag:

- The *Rule_3* tag fails to run if you set the `fast` tag parameter to `yes`.
- By default, this tag does not report any violation if the multiple driven signals/variables are not loaded or they have hanging driver input.
- The *Rule_3* tag does not consider black boxes as drivers for multiple-driven nets.

Parameters

- *strict*: By default, the *Rule_3* tag ignores bus-holders and top-level inout ports as drivers. Set this parameter to consider bus-holders and top-level inout ports as drivers. For all other modules, inout ports are always checked for tag violation irrespective of the value of this parameter.

NOTE: A violation is reported even without the *strict* parameter, if two or more output

terminals are connected together with an inout port. No violation is reported (even if the strict parameter is set), if an inout port is connected with the non-enabled tristate (disabled or the tristate having unknown value) driver.

- **[assume_driver_load](#)**: Default value is no. This indicates the *Rule_3* tag does not flag any violation, if the multiple driven signal/variable is not loaded or have hanging driver input. Set the value of the parameter to yes to report violations for such cases.
- **[checkconstassign](#)**: Default value is no. This indicates the *Rule_3* tag ignores nets that are driven multiple times by the same constant value. Set this parameter to yes to report such cases.
- **[check_bbox_driver](#)**: Default value is no. Set this parameter to yes to consider the black-box as a valid driver.
- **[ignore_equivalent_drivers](#)**: Default value is yes. Set the value of the parameter to no to report violations for a net, which is driven by two identical instances of basic two input gates of type OR/NOR/AND/NAND and both instances are driven by same input signals.
- **[ignore_greybox_drivers](#)**: Default value is no. Set the value of the parameter to yes to consider greybox as a valid driver, if this parameter is set to 'yes' then Tag will not consider greybox as a valid driver.
- **[handle_equivalent_drivers](#)**: The default value is no. Set the value of the parameter to yes to report violations when a net is simultaneously driven by two identical instances of two input gates of type OR, NOR, AND, or, NAND and both instances are driven by same input signals.
- **[show_connected_net](#)**: The default value is no. Set the value of the parameter to yes to report adjacent nets to a fan-in terminal of a multi-driven net and help in preventing hierarchical waiver breaks.

Constraints

None

Messages and Suggested Fix

The following message appears at the location where a signal `<sig-name>` is found to be driven by more than one sequential always/ process construct, instances, and/or concurrent statements:

[ERROR] signal '`<sig-name>`' has multiple simultaneous drivers

Potential Issues

Violation may arise when a signal has multiple simultaneous drivers.

Consequences of Not Fixing

A net that is not a tristate net should have a unique driver. If such nets are driven by more than one net, there is a possibility of an electrical conflict. For example, consider a net driven by outputs of two different AND gates.

If one output is 0 and other output is 1, the driven net is in a non-digital state. Such cases lead to excessive current flows from one AND gate to another AND gate causing chip electrical failures, causing chip malfunction.

The effect of a non-digital net can also propagate to its fan-out points. For such fan-out points, the net behaves as if it was floating resulting in further chip electrical and functional failures.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic shows the multiple driven net and a corresponding load for that net.

Most probably, there is an error in your logic. To fix the violation, ensure that variable/signal does not have multiple drivers, unless you actually intended tristate logic.

Example Code and/or Schematic

Example 1

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
port(
in1,in2 : in std_logic;
output  : out std_logic
);
end test;
architecture rtl of test is
begin
output <= in1 ;
output <= in2 ;
```


Netlist Tags

```
end rtl;
```

In the above example, the *Rule_3* tag reports a violation as the signal, *output*, has multiple simultaneous drivers, *in1* and *in2*.

Default Severity Label

Error

Tag Group

Not applicable

Reports and Related Files

None

Rule_4

Checks if output pin of an instance is tied to VDD/GND

Description

This tag reports violation if output pin of an instance of a cell is tied to VDD/GND. Such a connection may cause short circuit or ESD damage.

The tag does not report violation for:

- Pad pins (with the `sec_pin_type` : "pad" attribute).
- Pins tied inside the library cell (function: 1/0).
- Output of a Buffer/Inverter whose input is tied to VDD/GND. See Example 3 for details.

NOTE: *The Rule_4 tag also reports violation if output of a cell is connected to multi-driven net, and this net is connected to a buffer/inverter whose input is tied to VDD/GND. See Example 2, below, for details.*

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears if output pin of an instance of a cell is tied to VDD/GND:

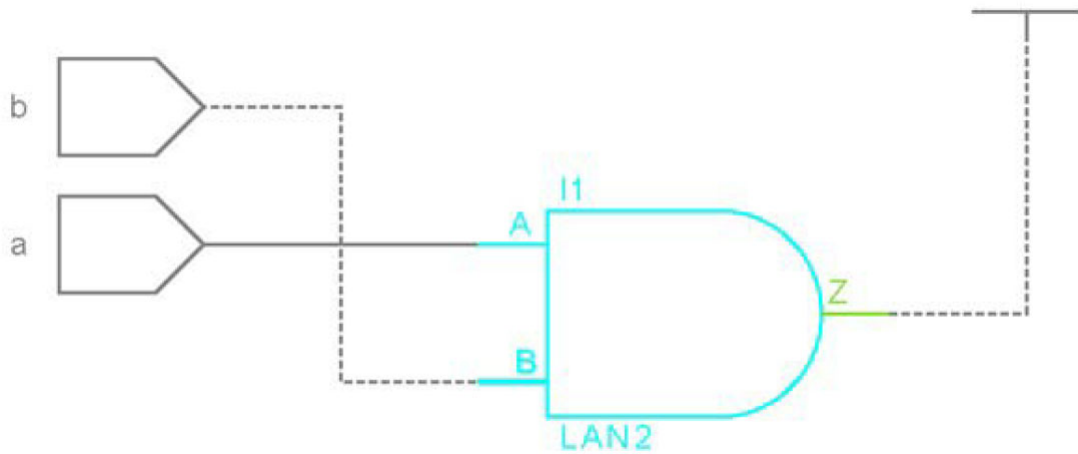
[ERROR] output pin '<pin-name>' of instance '<instance-name>' is tied to VDD/GND

Example Code and/or Schematic

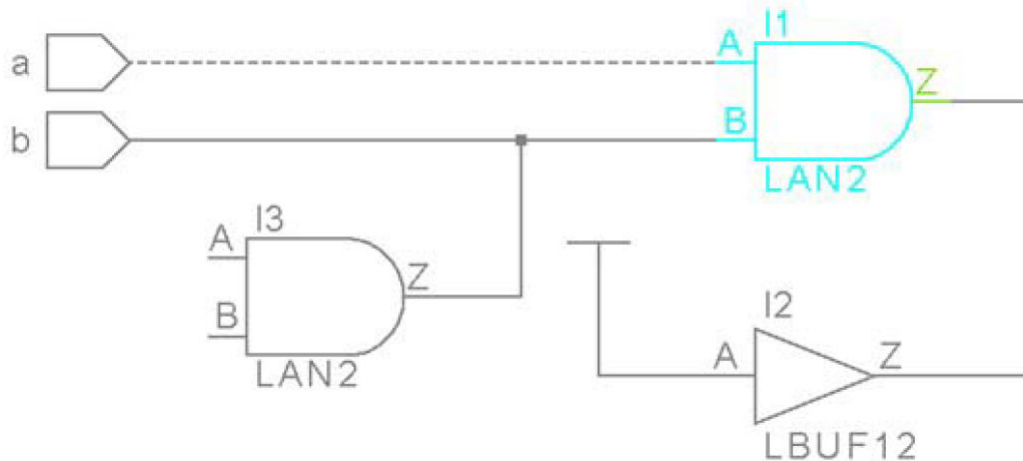
Example 1

A violation is reported in the following scenario:

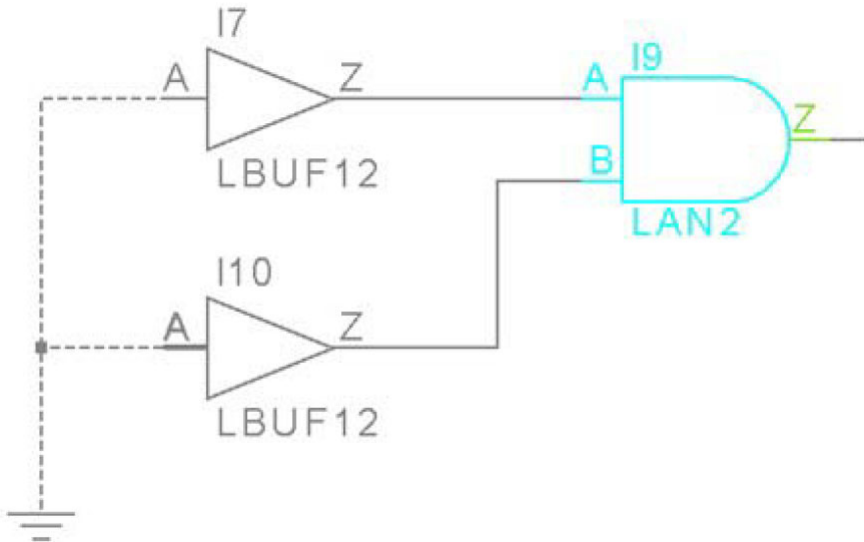
Netlist Tags

**Example 2**

A violation is reported in the following scenario:

**Example 3**

No violation is reported in the following scenario:

**Default Severity Label**

Error

Tag Group

None

Report and Related File

None

Rule_5

Check pin voltage difference between source and destination.

Description

The tag reports violation if there are connections between pins of lib cells that have different power/ground supply voltages. The traversal from the source pin to the destination pin is done only through buffers.

The Rule_5 tag also reports violation if any of the below condition is satisfied:

- The power pin name of source and destination do not match.
- The ground pin name of source and destination do not match

Also, the pin voltage level is determined using the `voltage_map`, `related_ground_pin`, and `related_power_pin` attributes of the liberty file.

Language

Verilog, VHDL

Parameter(s)

margin_value: The default value is 0.0. You can specify a floating point value as an input to this parameter to specify the permissible difference in the voltage values of source and destination pins. The tag will not report violations if the difference of voltage values of source and destination pins are within the range of this parameter value.

Constraint(s)

If the same cell name is available in different libraries the tag will pick up the library cell from library given through this constraint for a particular instance.

- **use_library_group**: Use the library cell from a given group for a particular instance. The following is the syntax of the constraint:

```
use_library_group -name <library group> -instname
<hierarchical path of instance name>
```

- **define_library_group:** Defines the library for a library group. The following is the syntax of the constraint:

```
define_library_group -name <library group name> -libname
<library name>
```

Violation Messages

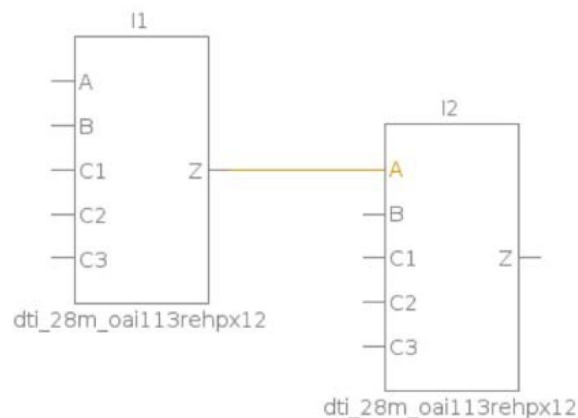
The tag displays the following Warning message when there is a difference in the power voltage level for the source and destination pins:

Source pin <hierarchical path name of pin> (cell:<cell name>, voltage: <voltage name> <voltage value>) and destination pin <hierarchical path name of pin> (cell:<cell name>, voltage: <voltage name> <voltage value>) connected through net <hierarchical path name of net> are operating at different <power/ground> voltage levels.

Example Code and/or Schematic

Example 1

Consider the following schematic:



In the above example, if power voltage of test.I1.Z is different from power voltage of test.I2.A, the tag reports the following violation message:

Source pin 'test.I1.Z' (cell: 'dti_28m_oai113rehpx12', voltage: VDD0 0.770000) and destination pin 'test.I2.A' (cell:

Netlist Tags

'dti_28m_oai113rehpx12', voltage: VDD 0.990000) connected through net 'test.w1' are operating at different power voltage levels

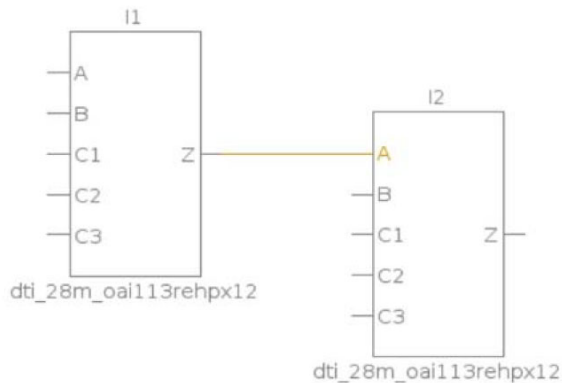
If ground voltage of test.I1.Z is different from power voltage of test.I2.A, the following violation message is reported:

Source pin 'test.I1.Z' (cell: 'dti_28m_oai113rehpx12', voltage: VDD1 0.000000) and destination pin 'test.I2.A' (cell: 'dti_28m_oai113rehpx12', voltage: VDD 0.500000) connected through net 'test.w1' are operating at different ground voltage levels

Also, if power/ground voltage of both test.I1.Z and test.I2.Z is the same, then the tag does not report any violation.

Example 2

Consider the following schematic:



If the value of the `margin_value` parameter is set to 0.5, the tag reports the following violation message if the difference between power/ground voltage of test.I1.Z and test.I2.Z pins is higher than 0.5.

Source pin 'test.I1.Z' (cell: 'dti_28m_oai113rehpx12', voltage: VDD2 0.330000) and destination pin 'test.I2.A' (cell: 'dti_28m_oai113rehpx12', voltage: VDD1 0.990000) connected through net 'test.w1' are operating at different power voltage levels

Default Severity Label

FATAL

Tag Group

Netlist

Report and Related File

None

Rule_8

Rule detects if transmission gate signal or instance are connected in series.

Description

When transmission gates are connected in series without buffer in between, change in delay or glitch may occur depending on the clock skew. This tag checks for only lib - cell transmission gates and ignores tool-generated (non-libcell) buffer / inverter only between any two libcell transmission gates.

If a libcell buffer or libcell inverter is specified using the [TRANSMISSION_GATE_CELL_LIST](#) parameter, the tag does not consider them as transmission gates.

Parameter(s)

[TRANSMISSION_GATE_CELL_LIST](#): Default value is ' '. Set the value of the parameter to comma-separated transmission gates to provide names of transmission gate cells.

Constraint(s)

None

Violation Messages

[FATAL] Transmission gate [DesignObjSignal] and [DesignInstance] are connected in series.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Fatal

Tag Group

Netlist

Report and Related File

None

Rule_9

Checks for incorrect parametric NAND connections

Description

This tag reports violations when the following conditions are not met:

- No two PIs/POEs should be connected to each other
- No two POs should be connected to each other
- The PI/POE and PO of the same cell should not be connected to each other.
- Every PI/POE should either be connected to one PO, VDD/GND, or primary port.
- Every PO should either be connected to one PI/POE, VDD/GND, or primary port.
- Every PO should be connected to only one PI/POE or it should be connected to one/many non-PIs/POEs. It should have a load. It should not be left unconnected.

All PI, POE, and PO pins have the following attributes set:

- `sec_pin_type : pndtree`

NOTE: *The tag checking is done assuming that only one of the PI or POE pins exists in a cell.*

Language

Verilog

Parameter(s)

[*PNDTREE_CELL_LIST*](#): Default value is " ". Specify a comma-separated list of pins to specify PI/PO pins.

Constraint(s)

None

Messages and Suggested Fix

The following message is reported by this tag :

[ERROR] Incorrect connections of parametric nand tree detected.
Pin '<pin-name>' is connected with <number> PI/POE pins ,
<number> PO pins , <number> other pins. Details of pin
connections (<pin-info>)

Default Severity Label

Error

Tag Group

None

Reports and Related Files

None

Rule_12

Checks if output of a combinational cell is connected to its own input

Description

This tag reports violation when output of a combinational cell is connected to its own input.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when output of a combinational cell is connected to its own input:

[ERROR] output '<pin-name>' of the combinatorial cell '<cell-name>' [Master: <cell-master-name>] is connected to its own input

Default Severity Label

Error

Tag Group

None

Reports and Related Files

None

Rule_14

Checks if PI/POE/PO pins of parametric NAND tree are unconnected

Description

This tag reports violations when the following condition is not met:

Every PI/POE/PO pin, having `sec_pin_type: pndtree`, should not be left unconnected. Either it should be connected to the supply or it should be having proper driver/load.

NOTE: *The tag checking is done assuming that only one of the PI or POE pins exists in a cell.*

Language

Verilog

Parameter(s)

[*PNDTREE_CELL_LIST*](#): Default value is " ". Specify a comma-separated list of pins to specify PI/PO pins.

Constraint(s)

None

Messages and Suggested Fix

The following message appears when any PI/POE/PO pins are left unconnected :

[ERROR] <PI | PO | POE> pin '<instance-name/pin-name>' is left unconnected

Default Severity Label

Error

Tag Group

None

Netlist Tags

Reports and Related Files

None

Rule_16

Checks if fan-in cone of set and reset pin of a flip-flop/latch converge

Description

The *Rule_16* tag reports flip-flops/latches where the fan-in cones of their set and reset pins converge.

Same net driving both the set pin and the reset pin of a flip-flop/latch can cause glitches.

No violation is reported for flip-flop or latch if their set or reset pin is connected to tied-high/tied-low. Also, the tag does not back-traverse through the select pins of a multiplexer.

Language

Verilog

Parameters

- *allviol*: The default value is no. Set this parameter to yes or *<tag-name>* to report all the violation messages.
- *ignore_hanging_flop*: Default value is no. When you set this parameter to yes, the tag does not report a violation for the flip-flop/latch if it's output is hanging.
- *check_depth*: Default value is -1. When you set this parameter to 0, the tag reports a violation when set and reset pins of a flip-flop are converging through the same source pin/net, which can be present in any module boundary and no logic exists between the source and set/reset pins.

Messages and Suggested Fix

The following message appears at the location where the output of flip-flop *<flip-flop-name>* when the fan-in cones of its set pin *<set-name>* and reset pin *<rst-name>* converge at net *<net-name>*:

[ERROR] <Flop | Latch> '*<flop-name>*' having reset '*<rst-name>*' and set '*<set-name>*' converges at net '*<net-name>*'

How to Debug and Fix

Separate the common driver of set and reset to avoid glitches.

Example Code and/or Schematic

Consider the following example:

```
module setreset(in1, in2,in3, clk, out);
    input in1, in2,in3, clk;
    output out;

    reg out;

    wire rst, prst;

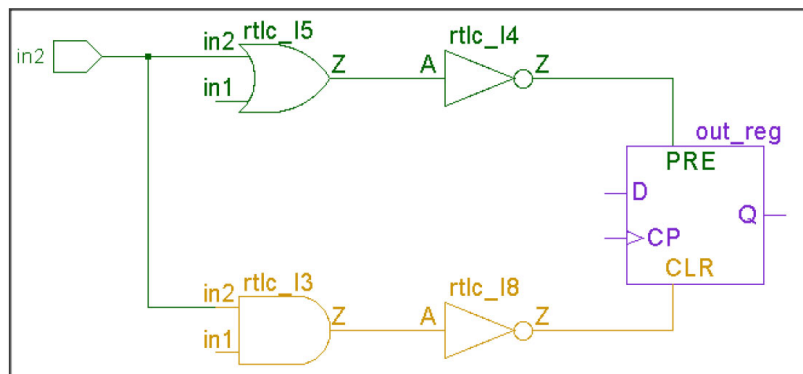
    assign rst = in1 & in2;
    assign prst = in1 | in2;

    always @ (posedge clk or negedge rst or negedge prst) begin
        if (~rst) out = 1'b0;
        else if (~prst) out = 1'b1;
        else out = in3;
    end
endmodule
```

In the above example, the *Rule_16* tag reports a violation because the same nets *in1* and *in2* are driving both the set pin and the reset pin of the flip-flop. The following message is reported by this example:

Flop 'setreset/out' having reset 'setreset/rst' and set 'setreset/prst' converges at net 'setreset/in2'

The schematic for the above example is shown below:



Default Severity Label

Error

Tag Group

None

Reports and Related Files

None

Rule_19

Checks if an invalid cell is used in the analog path

Description

This tag reports violation when there is a digital logic in the analog device path.

The following illegal chain is detected and reported:

```
Analog Cell <-> digital logic <-> Analog Pad
```

The tag does not report violation for the following chains:

```
Analog Cell <-> digital logic <-> Analog Cell
```

```
Analog Pad <-> digital logic <-> Analog Pad
```

```
Analog Cell or Pad <-> Analog Cell or Pad (no digital logic)
```

By default, a violation is reported for the following chains:

```
Analog Cell <-> digital pad <-> Analog Pad
```

```
Analog Cell <-> digital logic <-> digital pad <-> Analog Pad
```

However, when the [*ignore_digital_pad*](#) parameter is set to yes, the propagation is stopped at the digital pad and no violation is reported.

Identifying the analog/digital and cell/pad:

- A cell is analog cell if one of the following attributes are set:
 - ☐ `sec_class : analog`
 - ☐ `is_analog: true`
- A cell is analog pad if one of the following conditions are true:
 - ☐ `sec_class : pad and sec_io_internal_power_check : false`
 - ☐ `is_analog : true and is_pad : true`

- A cell is digital pad if it has 'sec_class : pad' attribute only or if it has 'sec_class : pad' and 'sec_io_internal_power_check : true' attributes.
- A digital logic is the one that does not qualify as an <analog cell>, <analog pad>, or <digital pad>.

Propagation will stop whenever a clock pin of a flip-flop is found during the fanout traversal.

Language

Verilog

Parameters

ignore_digital_pad: The default value is no. Set this parameter to yes to stop propagation at digital pad.

Messages and Suggested Fix

The following message is reported when invalid cells are detected in the analog path:

```
[ERROR] '<number>' Invalid cell(s) detected in the analog path  
[Start Inst: '<start-analog-inst-name>', Start Pin Name:  
'<start-pin-name>', End Inst: '<end-analog-inst-name>']
```

Default Severity Label

Error

Tag Group

None

Reports and Related Files

None

Rule_20

Checks if a hold fix delay cell is used in the clock/enable path

Description

This tag reports violation when cells having `sec_cell_type : hold_buffer` attribute value are used in the fanin cone of the clock/enable pin of the sequential element/latch.

To search for hold buffer cells, the tag traverses fan-in cone of clock/enable pin of the sequential element/latch till it reaches the primary port. The tag stops the traversal if any of the following is found in the path:

- MUX select pins
- Black-box

NOTE: *Only one violation per instance is reported for a delay cell.*

Language

Verilog

Parameter(s)

***delay_cell_list*:** By default, the *Rule_20* tag reports a violation when cells having the `sec_cell_type: hold_buffer` attribute value are used in the fanin cone of the clock/enable pin of the sequential element/latch.

Set the value of this parameter to a comma-separated list of PERL regular expressions, to report a violation when the name of a cell in the fanin cone matches the given name.

Constraint(s)

None

Messages and Suggested Fix

The following message is reported when a cell is used in the clock/enable path:

[ERROR] Cell <cell-master-name1>(<hier-cell-name1>) should not be used in the clock/enable path of the cell <cell-master-name2>(<hier-cell-name2>)

Default Severity Label

Error

Tag Group

None

Reports and Related Files

None

Rule_22

Checks if design has more than one parametric NAND trees

Description

This tag reports violation if the design has more than one parametric NAND tree. The definition and detection algorithm of parametric NAND tree is as follows:

Consider that the starting point of p-nand tree is checking for PI/POE pins. If PI/POE pin of a cell `cell_1` is found to be driven by VDD/GND, then it traverses further from the PO pin of that cell, `cell_1`, to check further connectivity and possible detection of p-nand tree. The PO pin of `cell_1` should be connected to the PI/POE pin of the cell `cell_2` and further PO of `cell_2` should be connected to:

- Either PI/POE pin of `cell_3`. In this case, the tag will recursively traverse further from PO pin of `cell_3` for possible detection of p-nand tree.
- Or it should be connected one or many non-PI/POE pins. The PO pin of `cell_2` should have a load. It should not be left unconnected. This connection will be considered as the detection of a valid p-nand tree.

The following points should also be considered while detection of a parametric NAND tree configuration:

- No two PIs/POEs should be connected to each other
- No two POs should be connected to each other
- The PI/POE and PO of the same cell should not be connected to each other
- Every PI/POE should either be connected to one PO or to VDD/GND
- Every PO should be connected to only one PI/POE or it should be connected to one or more non-PI/POE pins. It should have a load. It should not be left unconnected.

All PI, POE, and PO pins have the `sec_pin_type : pndtree` attribute set.

NOTE: *The tag checking is done assuming that only one of the PI or POE pins exists in a cell.*

Language

Verilog

Parameter(s)

[*PNDTREE_CELL_LIST*](#): Default value is " ". Specify a comma-separated list of pins to specify PI/PO pins.

Constraint(s)

None

Messages and Suggested Fix

The following message is reported when design has multiple parametric NAND tree configurations:

[WARNING] Multiple (<number>) parametric nand tree configurations detected

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

Rule_23

Checks if control pin of an instance is tied to VDD or GND

Description

This tag reports violation if control pin of an instance of a cell is tied to VDD or GND. The tag considers set or reset pins as control pins for flip-flop.

The tag does not report a violation for the spare cells. A spare cell is a cell:

- Whose all input pins are tied to power/ground, all output pins are floating, and all inout pins are either tied to power/ground or are floating.
- Whose name is specified using the [spare_cell_name](#) parameter. Default value of this parameter is `spare_.*`.

The tag will report viol, when any one of set or reset pins of a flip-flop is connected to either VDD or GND

The tag does not report violation for pad cell or spare cell.

Language

Verilog

Message Details

The following message appears if control pin of an instance of a cell is tied to VDD or GND:

[WARNING] Control pins '<set-pin-name>' and '<reset-pin-name>' of instance '<instance-name>' is tied to '<VDD | GND>'

Default Severity Label

Warning

Tag Group

None

Report and Related File

None

Rule_24

Detects combinational loops

When to Use

Use this tag to detect combination loops in the design.

Description

The *Rule_24* tag reports violation for the combinational loops in the design. A combinational loop is created when the output of a cell loops back onto its inputs directly or through many other combinational cells.

For complete list of nets in the loop, refer to the [Rule_24_Report](#) report. The report is generated only for the violating cases.

Language

Verilog

Parameter(s)

- [enableE2Q](#): The default value is `no`. Set the value of the parameter to `yes` to detect loops that pass through enable pin to Q (output) pin path of a latch.
- [overlappingLoops](#): The default value is `yes`. Set the value of the parameter to `no` to ignore the overlapping loops.
- [ignoreSRlatch](#): The default value is `no`. Set the value of the parameter to `yes` to ignore the SR-latch combinational loops.
- [allviol](#): The default value is `no`. Set the parameter to `yes` or `<tag-name>` to report all violation messages.
- [strict](#): The default value is `no`. Set the value of the parameter to `yes` to enable the *Rule_24* tag to report latches implemented through a MUX.
- [report_mux_select](#): The default value is `yes`. Set the value of the parameter to `no` to enable the *Rule_24* tag to report combinational loops existing through the select pin of a MUX.
- [report_flop_reset_loop](#): The default value is `no`. Set the value of the parameter to `yes` to enable the tag to report violation when a loop exists through set/reset pin and output pin of a flip-flop.

- ***effort_level***: The default value is 100. Set the value of the parameter to any positive integer to specify the number of times the *Rule_24* tag should traverse a net to detect a combinational loop.
- ***ignore_iopad***: The default value is no. If you set this parameter to yes, the tag does not report combinational loops that traverse input and output terminals of a pad cell.

Constraint(s)

set_case_analysis: This constraint specifies the case analysis conditions.

Messages and Suggested Fix

Message 1

[INFORMATION] Combinational loop exists at '<signal-name-list>'

In the above message, the <signal-name-list> contains the list of signals in the combinational loop.

NOTE: *The combinatorial loops reported by the Rule_24 tag may also contain synthesis tool-generated nets that do not have the HDL correspondence. Their names will always be displayed as <synth_gen_net> so you must not try to search those nets in the HDL file.*

Potential Issues

A combinational loop is created when the output of a cell loops back onto its inputs directly or through many other combinational cells.

Consequences of Not Fixing

The timing behavior and ultimately functional behavior of logic in a combinational loop is highly dependent on the target technology because its behavior depends on gate and interconnect delays. Such designs may not be portable and may even have reliability problems if the timing dependencies are tight enough to fall within manufacturing process variations.

The output of a cell, which causes a combinational loop can:

- Get into a locked state

For example, consider a two input AND gate with output connected to one of the inputs. When other input of the AND gate is 0, the AND gate output is locked to a 0.

■ Oscillate at potentially high frequency

For example, consider the output of an inverter being looped back to its input through an OR gate. When the OR gate is enabled, the inverter can oscillate between logic 0 and 1 (separated in time by the propagation delay of the loop). This can lead to unintended logic behavior and potentially high power consumption.

There can be variations in the scenarios specified above. Hence, it is best to avoid combinational loops.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic shows the combinational loop. You can view the details of all the nets for the violating loop (in text) using the [Rule_24_Report](#).

The violation message lists all the nets for the violating loop, if the number of nets is less than or equal to ten. However, the violation message lists the first ten nets for the violating loop, if the number of nets is more than ten.

For SR type latches, use the hard macros from the vendor library where the macro is guaranteed to have deterministic timing and functional behavior.

To fix the violation, search a place to break the combinational loop with a flip-flop, unless the loop is introduced by mistake.

Message 2

[INFO] Please refer to 'Rule_24_Report.rpt' for complete combinational loops

Potential Issues

The violation message appears only when **Message 1** is reported for the *Rule_24* tag.

Consequences of Not Fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

Double-click the violation message. The [Rule_24_Report](#) is displayed. This is an informational message. Therefore, no fix is required for this message.

Example Code and/or Schematic

Consider the following example:

```
module m0 (in1, out);  
  input [3:0] in1;  
  output [7:0] out;  
  wire a;  
  assign a = ~in1[2];  
  assign out[3] = a;  
  assign in1[2] = ~out[3];  
endmodule
```

In the above example, the *Rule_24* tag reports a violation as a combinational loop is present in the design.

Default Severity Label

Info

Tag Group

None

Report and Related File

[Rule_24_Report](#)

Rule_25

Checks if gated/generated clocks are generated from more than one block in the design

Description

This tag reports gated clock signals.

Gating clocks create skew and therefore timing problems. Such constructs also cause problems for ATPG tools.

While clock gating is necessary for low-power designs, this tag works as a tool to verify that the clock gating has been implemented only where intended.

The tag does not report violations for the ICG cells and multiplexers.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears at the first line of a module where a clock signal that defines the sequential element has been gated:

[INFO] Gated clock '<clock-name>' (flop: <flip-flop-name>) detected

How to Debug and Fix

No fix may be necessary, but review all gating to make sure it is where you expected. You should also consider whether it would be better to leave clock gating to the block user, rather than embedding it in the block.

Example Code and/or Schematic

Consider the following example where the clock signal, `clock1` has been gated using the ASIC gate `ND2`:

Netlist Tags

```

module test(clock, in1, in2, out1, out2);
    input  clock;
    input  [63:0] in1, in2;
    output [63:0] out1, out2;

    reg    [63:0] out1, out2;
    wire   clock1, clock2;

    ND2 n(.A(clock), .B(clock2), .Z(clock1));

    assign clock2 = 1'b1;

    always@(posedge clock1)
        out1 = in1;

    always@(posedge clock1)
        out2 = in2;
endmodule

```

For this example, SpyGlass generates the following message:
 Gated Clock 'clock1' (flop: test/out2[63]) detected

Default Severity Label

Info

Tag Group

None

Reports and Related Files

None

Rule_26

Checks if all the inputs of a cell are tied to VDD or GND

Description

This tag reports violation when the `sec_pin_type` : data attribute is set for an input pin. If set, the tag will check for all inputs of a cell, which are tied with VDD/GND or if 1/0 is reaching the input terminal through the `set_case_analysis` constraint.

The tag checking is not done for pad cells. A pad cell is the cell whose one pin has the following attributes set:

- `sec_pin_type="pad"`
- `is_pad:true`
- `pad_cell:true`

The tag does not report a violation for the spare cells. A spare cell is a cell:

- Whose all input pins are tied to power/ground, all output pins are floating, and all inout pins are either tied to power/ground or are floating.
- Whose name is specified using the [spare_cell_name](#) parameter. Default value of this parameter is `spare_.*`.

Language

Verilog

Parameter(s)

None

Constraint(s)

`set_case_analysis`: This constraint specifies the case analysis conditions.

Messages and Suggested Fix

The following message is reported when all the input pins of a cell are tied to VDD/GND :

Netlist Tags

[WARNING] All the inputs of the cell '<cell-name>' are tied to VDD/GND

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

Rule_26-1

Report library cell input pin's not connected to 'TIE' Cell.

Description

The Rule_26-1 tag reports violation for a library cell input pin, which is directly connected to supply or constant. This tag will not report violation for libcell input pins which are directly connected to TIE cell.

A cell is considered a TIE cell if it is a TIE cell and its name matches the input specified using the [allow_tie_cell_list](#) parameter.

Parameter(s)

[allow_tie_cell_list](#): The following is the default value of the parameter:

```
"EPTIELO.* EPTIEHI.* GPTIEHI.* GPTIELO.* TIEHI.*  
TIELO.*"
```

Specify a space-separated list of perl regular expressions to specify list of Lib cells, which are required to be considered as a Tie cell.

Constraint(s)

None

Messages and Suggested Fix

Pin <PIN_NAME> of instance <INSTANCE_NAME> is not connected to Tie Cell.

Potential Issues

The violation message is reported for a Library cell input pin that is not connected to a Tie cell.

Consequences of Not Fixing

Not fixing the violation may impact physical flow netlist validation.

How to Debug and Fix

To fix the violation, ensure that the Library cell input pin is connected to the Tie cell.

Netlist Tags

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

Rule_27

Checks if multiple inputs of the same cell are tied together

Description

This tag reports a violation when a cell has two or more input signals and all the inputs are tied to the same signal.

Language

Verilog

Parameters

allow_combo: By default, this parameter is set to `no` and the *Rule_27* tag ignores combinational elements while performing fan-in traversal of input signals of cells. When this parameter is set to `yes`, the tag does not ignore such elements, except buffer.

Constraint(s)

None

Messages and Suggested Fix

The following message is reported when multiple inputs of the same cell are tied together:

[WARNING] Input pins '<pin-name-list>' of cell '<cell-name>' are tied together

Default Severity Label

Warning

Tag Group

None

Reports and Related Files

None

Rule_28

Detects combinational loops, including loop through the set/reset and output pins of a flip-flop

When to Use

Use this tag to detect combination loops in the design.

Description

The *Rule_28* tag reports violation for the combinational loops in the design. A combinational loop is created when the output of a cell loops back onto its inputs directly or through many other combinational cells.

This tag also reports violation if the loop exists through set/reset and output pin of a flip-flop.

For complete list of nets in the loop, refer to the [Rule_28_Report](#) report. The report is generated only for the violating cases.

Default Weight

10

Language

Verilog

Parameter(s)

- [enableE2Q](#): The default value is `no`. Set the value of the parameter to `yes` to detect loops that pass through enable pin to Q (output) pin path of a latch.
- [overlappingLoops](#): The default value is `yes`. Set the value of the parameter to `no` to ignore the overlapping loops.
- [ignoreSRlatch](#): The default value is `no`. Set the value of the parameter to `yes` to ignore the SR-latch combinational loops.
- [allviol](#): The default value is `no`. Set the parameter to `yes` or `<tag-name>` to report all violation messages.
- [strict](#): The default value is `no`. Set the value of the parameter to `yes` to enable the *Rule_28* tag to report latches implemented through a MUX.

- ***report_mux_select***: The default value is *yes*. Set the value of the parameter to *no* to enable the *Rule_28* tag to report combinational loops existing through the select pin of a MUX.
- ***effort_level***: The default value is 100. Set the value of the parameter to any positive integer to specify the number of times the *Rule_28* tag should traverse a net to detect a combinational loop.
- ***ignore_iopad***: The default value is *no*. If you set this parameter to *yes*, the tag does not report combinational loops that traverse input and output terminals of a pad cell.

Constraint(s)

set_case_analysis: This constraint specifies the case analysis conditions.

Messages and Suggested Fix

Message 1

[INFO] Combinational loop exists at '<signal-name-list>'

In the above message, the <*signal-name-list*> contains the list of signals in the combinational loop.

NOTE: *The combinatorial loops reported by the Rule_28 tag may also contain synthesis tool-generated nets that do not have the HDL correspondence. Their names will always be displayed as <synth_gen_net> so you must not try to search those nets in the HDL file.*

Potential Issues

A combinational loop is created when the output of a cell loops back onto its inputs directly or through many other combinational cells.

Consequences of Not Fixing

The output of a cell, which causes a combinational loop can:

- Get into a locked state
For example, consider a two input AND gate with output connected to one of the inputs. When other input of the AND gate is 0, the AND gate output is locked to a 0.
- Oscillate at potentially high frequency
For example, consider the output of an inverter being looped back to its

input through an OR gate. When the OR gate is enabled, the inverter can oscillate between logic 0 and 1 (separated in time by the propagation delay of the loop). This can lead to unintended logic behavior and potentially high power consumption.

There can be variations in the scenarios specified above. Therefore, it is best to avoid combinational loops.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic shows the combinational loop. You can view the details of all the nets for the violating loop (in text) using the [Rule_28_Report](#).

The violation message lists all the nets for the violating loop, if the number of nets is less than or equal to ten. However, the violation message lists the first ten nets for the violating loop, if the number of nets is more than ten.

For SR type latches, use the hard macros from the vendor library where the macro is guaranteed to have deterministic timing and functional behavior.

To fix the violation, search a place to break the combinational loop with a flip-flop, unless the loop is introduced by mistake.

Message 2

[INFO] Please refer to 'Rule_28_Report.rpt' for complete combinational loops

Potential Issues

The violation message appears only when **Message 1** is reported for the *Rule_28* tag.

Consequences of Not Fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

Double-click the violation message. The [Rule_28_Report](#) is displayed. This is an informational message. Therefore, no fix is required for this message.

Example Code and/or Schematic

Consider the following example:

```
module m0 (in1, out);
```

```
input [3:0] in1;
output [7:0] out;
wire a;
  assign a = ~in1[2];
  assign out[3] = a;
  assign in1[2] = ~out[3];
endmodule
```

In the above example, the *Rule_28* tag reports a violation as a combinational loop is present in the design.

Default Severity Label

Information

Tag Group

None

Report and Related File

[Rule_28_Report](#)

Rule_29

Checks if output data pin is connected to input data pin of the same storage element

Description

This tag reports a violation if any output data pin (with the `sec_pin_type:"data"` or `nextstate_type:data` attribute) is connected to any input data pin (with the `sec_pin_type:"data"` or `nextstate_type:data` attribute) of the same cell.

Storage element should not have any reset/set pins. This design issue can cause initialization problem.

NOTE: *The tag checking is done for flip-flops only.*

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message is reported when output data pin is connected to input data pin of the same storage element:

```
[ERROR] output data pin <pin-name> is connected to the Input  
data pin <pin-name> of the same cell 'Hier-Inst-Name'
```

Default Severity Label

Error

Tag Group

None

Reports and Related Files

None

Rule_32

Reports variable/signals that do not infer a tristate and have multiple simultaneous drivers

When to Use

Use this tag to identify variable/signals that do not infer a tristate and have multiple simultaneous drivers.

Description

The Rule_32 tag reports variables/signals that do not infer tristate nets and have multiple simultaneous drivers.

The Rule_32 tag does not report violation for tristate nets whose immediate fan-in drivers are tristate elements. However, the tag reports a violation when an enable signal for two tristate elements is same as both the elements get enabled at the same point of time.

This tag reports variables/signals that are driven in more than one sequential always/process constructs, instances, and/or concurrent statements. Such cases are errors unless the variable infers a tristate net.

Language

Verilog, VHDL

Parameters

- **strict:** By default, the *Rule_32* rule ignores instances, bus-holders and top-level inout ports as drivers. Set this parameter to consider instances, bus-holders and top-level inout ports as drivers. For all other modules, inout ports are always checked for rule violation irrespective of the value of this parameter.

NOTE: A violation is reported even without the *strict* parameter, if two or more output terminals are connected together with an inout port. No violation is reported (even if the *strict* parameter is set), if an inout port is connected with the non-enabled tristate (disabled or the tristate having unknown value) driver.

- **assume_driver_load:** Default value is no. This indicates the *Rule_32* tag does not flag any violation, if the multiple driven signal/variable is not loaded or have hanging driver input. Set the value of the parameter to yes to report violations for such cases.

- [*checkconstassign*](#): Default value is `no`. This indicates the *Rule_32* tag ignores nets that are driven multiple times by the same constant value. Set this parameter to `yes` to report such cases.
- [*check_bbox_driver*](#): Default value is `no`. Set this parameter to `yes` to consider the black-box as a valid driver.
- [*ignore_greybox_drivers*](#): Default value is `no`. Set the value of the parameter to `yes` to consider greybox as a valid driver, if this parameter is set to 'yes' then tag will not consider greybox as a valid driver.
- [*handle_equivalent_drivers*](#): The default value is `no`. Set the value of the parameter to `yes` to report violations when a net is simultaneously driven by two identical instances of two input gates of type BUFF, NOT, OR, NOR, AND, or, NAND and both instances are driven by same input signals.
- [*show_connected_net*](#): The default value is `no`. Set the value of the parameter to `yes` to report adjacent nets to a fan-in terminal of a multi-driven net and help in preventing hierarchical waiver breaks.

Constraints

None

Messages and Suggested Fix

[WARNING]Signal '<sig-name>' has multiple simultaneous drivers

Potential Issues

The violation is reported when a signal has multiple simultaneous drivers.

Consequences of Not Fixing

A net that is not a tristate net should have a unique driver. If such nets are driven by more than one net, there is a possibility of an electrical conflict. For example, consider a net driven by outputs of two different AND gates. If one output is 0 and other output is 1, the driven net is in a non-digital state. Such cases lead to excessive current flows from one AND gate to another AND gate causing chip electrical failures, causing chip malfunction. The effect of a non-digital net can also propagate to its fan-out points. For such fan-out points, the net behaves as if it was floating resulting in further chip electrical and functional failures.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic shows the multiple driven net and a corresponding load for that net.

Most probably, there is an error in your logic. To fix the violation, ensure that variable/signal does not have multiple drivers, unless you actually intended tristate logic.

Example Code and/or Schematic

Example 1

Consider the following example:

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
port(
in1,in2 : in std_logic;
output  : out std_logic
);
end test;
architecture rtl of test is
begin
output <= in1 ;
output <= in2 ;
end rtl;
```

In the above example, the *Rule_32* tag reports a violation as the signal, output, has multiple simultaneous drivers, in1 and in2.

Default Severity Label

Warning

Tag Group

MultipleDriver

Reports and Related Files

None

Rule_34

Detects undriven and unloaded nets/terminals in the design

Description

The *Rule_34* tag flags nets and terminals that are neither driven nor loaded. See the [Rule_2](#) tag for description of how terminal drivers and terminal loads are inferred.

The *Rule_34* tag does not cross boundary of a library cell in any of its traversal. If a net is connected to an input (output) terminal of a library cell, then the net is considered as loaded (driven).

NOTE: *This tag does not report violation for output nets/terminals.*

Parameters

- [check_input_port](#): Default value is yes. Set the value of the parameter to no to enable violation on output ports.
- [ignoreRtlBuffer](#): To ignore all the RTL buffers within and outside the module boundary, set the parameter to yes.
- [checkInHierarchy](#): By default, the *Rule_34* tag does not traverse down the hierarchy to search for valid driver/load. Set the tag parameter to yes to have the *Rule_34* tag check inside the hierarchy.
- [checkTopDu](#): By default, the *Rule_34* tag checks for all DU's but if option -checkTopDu is set to tag name then it will check only top level nets and terminals and while detecting load, tag dives down the hierarchy till it detects the input of a combination/sequential/black-box. Also, in case -checkTopDu is set to tag name, then to detect a driver it traverses back to find a driver through hierarchy.
- [ignore_unloaded_inst](#): By default, the *Rule_34* tag considers combinational blocks to search for a valid load. Set this parameter and [checkInHierarchy](#) parameter to yes to ignore the combinational blocks so that this tag traverses down the hierarchy to search for a valid load.
- [handle_greybox](#): By default, the *Rule_34* tag does not report violation for output terminals (undriven from inside) of a module instance that are loaded by a grey box. The output terminals are considered as loaded (though undriven) due to its connectivity with complete grey box. However, if you set the value of the parameter to no, the *Rule_34* tag

Netlist Tags

reports violation for output terminals of a module instance that are loaded by a greybox.

Language

Verilog

Messages and Suggested Fix

The following message appears at the location where net *<net-name>* is first used when the net is neither driven nor loaded:

[WARNING] Detected undriven and unloaded(unconnected) net *<net-name>*

The following message appears at the location of instance containing terminal *<term-name>* when the terminal is neither driven nor loaded:

[WARNING] Detected undriven and unloaded(unconnected) terminal *<term-name>*

Example Code and/or Schematic

Consider the following example:

```
module top(in1,in2);  
wire [6:0] out1;  
    input in1,in2;  
    assign out1[1]= in1 & in2;  
endmodule
```

In the above example, the *Rule_34* tag reports a violation as the *out1[6:2, 0]* is neither driven nor loaded (unused). The following message is reported by this example:

Detected undriven and unloaded(unconnected) net top/out1[6:2,0]

Default Severity Label

Warning

Tag Group

None

Report and Related File

None

Rule_35

Detects undriven but loaded net in the design

Description

The *Rule_35* tag flags nets that are not driven but are loaded.

The *Rule_35* tag checks only those nets which are described in a module and have load or driver using RTL construct like assignment statements. For example:

```
out = in1 & in2;
```

In the above example, nets, *in1* and *in2*, have load but no driver.

The *Rule_35* tag is not applicable for those nets that have load/driver through another module instances (connected to instance input terminal). To check for undriven nets connected to modules instances, use the [Rule_2](#) tag.

This tag does not report any violation for undriven port nets.

The *Rule_35* tag does not cross boundary of a library cell in any of its traversal. If a net is connected to an input (output) terminal of a library cell, then the net is considered as loaded (driven).

While traversing the design to find the drivers of top-level nets, the *Rule_35* tag ignores buffers and inverters.

For other nets, the first connected output pin of a combinational gate, a sequential cell instance, or a user-instantiated cell is assumed to be the driver.

Language

Verilog

Parameters

- [checkTopDu](#): By default, the *Rule_35* tag checks for all DU's. However, if the *checkTopDu* parameter is set, then the tag checks only top level nets and while detecting the driver, the tag traverses through the hierarchy till it detects any combination, sequential, or black box acting as a driver. The tag reports violation only for top level undriven nets.
- [ignore_unloaded_inst](#): By default, the *Rule_35* tag considers combinational blocks to search for a valid load. Set this parameter and

[*checkInHierarchy*](#) parameter to `yes` to ignore the combinational blocks so that this tag traverses down the hierarchy to search for a valid load.

- [*handle_greybox*](#): By default, the *Rule_35* tag considers the net (properly loaded) connected (driven by) to grey box terminal as driven. Set the value of parameter to `no` to consider the net connected to the grey box terminal as undriven.
- [*checkInHierarchy*](#): Set the parameter to `yes` to traverse through the hierarchy to search for valid driver/load.
- [*ignoreRtlBuffer*](#): Set the parameter to `yes` to ignore all the RTL buffers within and outside the module boundary.

NOTE: *If the `checkTopDu` parameter is set then the `ignoreRtlBuffer` parameter will not be effective.*

Messages and Suggested Fix

The following message appears at the location where net `<net-name>` is first used when the net is undriven but loaded:

[INFORMATION] Detected undriven net `<net-name>`

Example Code and/or Schematic

Example 1

Consider the following example:

```
module internalNet (op, in1, in2);
  output [3:0] op;
  input [3:0] in1, in2;

  wire [7:0] internalNet;

  assign op[0] = in1[0] & in2[0];
  assign op[1] = in1[1] & in2[1];
  assign op[2] = in1[2] & in2[2];
  assign op[3] = internalNet[2] & internalNet[3];
endmodule
```

The *Rule_35* tag reports the following message for the above example:

Netlist Tags

Detected undriven net `internalNet.internalNet[3:2]`

Example 2

Consider the following example:

```
module test(tap_spare2_out,mode_control_bsr_scanout);  
  output tap_spare2_out;  
  input mode_control_bsr_scanout;  
  assign mode_control_bsr_scanout = tap_spare2_out;  
endmodule
```

In the above example, the *Rule_35* tag does not report violation for `tap_spare2_out` and `mode_control_bsr_scanout`.

Default Severity Label

Information

Tag Group

None

Report and Related File

None

Rule_36

Checks if all outputs of an instance are unconnected

Description

This tag reports violation if all outputs of an instance of a cell are unconnected. The tag also considers an output terminal as unconnected, if the net connected to the terminal is unloaded. Such instances are redundant and should be removed from the design.

The tag does not report a violation for the spare cells. A spare cell is a cell:

- Whose all input pins are tied to power/ground, all output pins are floating, and all inout pins are either tied to power/ground or are floating.
- Whose name is specified using the [spare_cell_name](#) parameter. Default value of this parameter is `spare_.*`.

Language

Verilog

Messages and Suggested Fix

The following message appears if all outputs of an instance of a cell are unconnected:

[WARNING] All outputs of Instance '<instance-name>' are floating

Default Severity Label

Warning

Tag Group

None

Report and Related File

None

Rule_41

Primary port of a block is tied to global signal

When to Use

Use this tag to check if the primary port of a block is tied to a global signal.

Description

The Rule_41 tag checks for primary ports of a current block, that is, the top modules, which are tied to a global signal.

Global signals means that power/ground signal as defined by supply0 and supply1.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[ERROR] Input port <PORT_NAME> of Top Module is tied to Global Signal

Potential Issues

The violation message is displayed when an input port of the top module is tied to a global signal.

Consequences of Not Fixing

When a primary port of a block is tied to global signal, RC extraction takes very large run time.

How to Debug and Fix

Double-clicking the violation highlights the violating path.

Example Code and/or Schematic

Consider the following code snippet:

```
module top(in1,in2,out);
```

```
input in1;  
input in2;  
output out;  
supply0 gnd;  
reg out;  
assign in1 = gnd;
```

In the above example, the input port 'in1' of top module is tied to global signal 'top/gnd'.

Default Severity Label

Error

Tag Group

None

Report and Related File

None

Rule_42

Analog pin is tied to power or ground.

When to Use

Use this tag to check if an analog or pad pin is tied to power or ground.

Description

The Rule_42 tag detects and reports a violation for an input analog or pad pin, which is tied to power or ground.

An analog cell is the cell whose one pin has the following attributes set:

- `sec_class : analog`
- `is_analog: true`

A pad cell is the cell whose one pin has the following attributes set:

- `is_pad:true`
- `pad_cell:true`
- `sec_pin_type=pad`

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Analog pin<name of pin > for instance< instance name> is tied to VDD or GND

Potential Issues

The violation message is displayed when an analog pin for an instance is tied to power or ground.

Consequences of Not Fixing

In the multi-power device, the tied pins should be identified with the exact

voltage level.

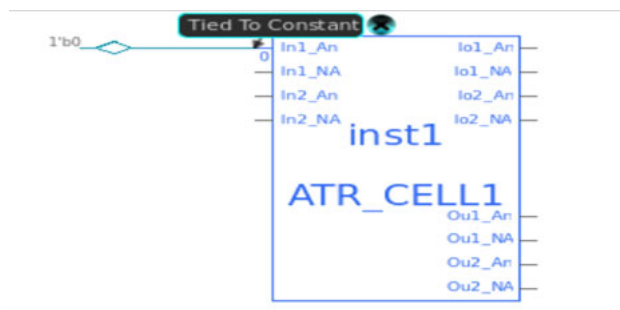
How to Debug and Fix

Double-clicking the violation highlights the violating path.

Example Code and/or Schematic

Example 1

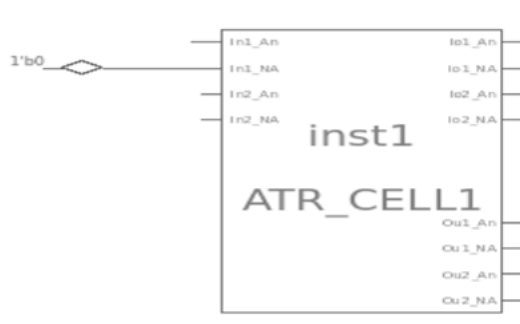
In the following figure, constant 0 is supplied to analog pin "in1_An".



Therefore, the tag reports a violation.

Example 2

In the following figure, constant 0 is attached to non-analog pin of the inst1 of ATR_CELL .



Therefore, the tag does not report a violation.

Example 3

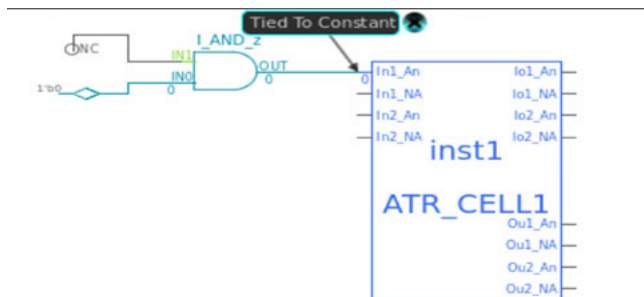
In the following figure, constant is attached to output pin of instance.



Therefore, the tag does not report a violation.

Example 4

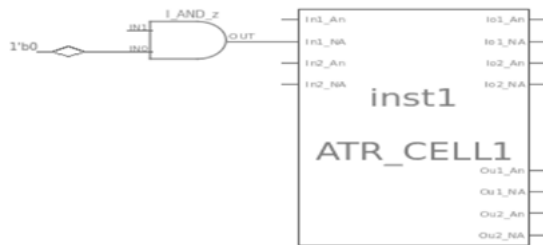
In the following figure, constant input 0 is propagated through AND gate(I_AND_z) and attached to analog pin (In1_An) of libcell (ATR_CELL1).



Therefore, the tag reports a violation.

Example 5

In the following figure, constant is propagated through AND gate(I_AND_z) but attached to non-analog Pin (In1_NA) of inst1.



Therefore, the tag does not report a violation.

Example 6

In the following figure, constant is supplied to Non analog pin (In_NA).



Therefore, the tag does not report a violation.

Default Severity Label

Warning

Tag Group

None

Report and Related File

None

Rule_43

Memory Efuse Pin should be directly connected to PadCell

When to Use

Use this tag to check if the memory Efuse pin is directly connected to PadCell.

Description

The Rule_43 tag reports a violation if memory Efuse Pin is not directly connected to PadCell.

A pad cell is the cell whose one pin has the following attributes set:

- `is_pad:true`
- `sec_class: pad`
- `sec_pin_type: pad`

Parameter(s)

FUSE_PIN: The default value is ' '. Specify a comma-separated list of pin names to specify the names of instances and pins.

Constraint(s)

None

Messages and Suggested Fix

[ERROR] Memory EFUSE pin <Pin Name> of instance < Instance Name> is not connected to PAD cell.

Potential Issues

The violation message is displayed when a memory EFUSE pin of an instance is not connected to the PAD cell.

Consequences of Not Fixing

Not fixing the violation may result in more CPU time.

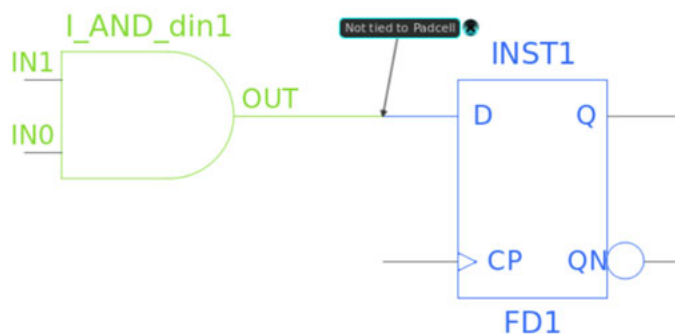
How to Debug and Fix

Double-clicking the violation highlights the pin, which is attached to the constant.

Example Code and/or Schematic

Example 1

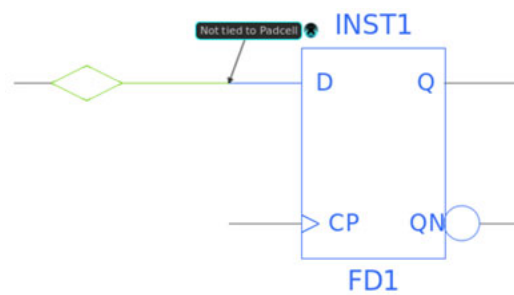
In the following figure, Pin D of INST1 is connected to PadCell through AND gate:



Therefore, the tag will report a violation.

Example 2

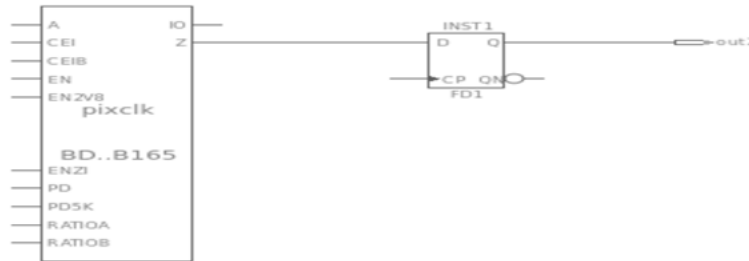
In the following figure, pin D of FD1 INST1 is attached to the buffer.



Therefore, the tag reports a violation.

Example 3

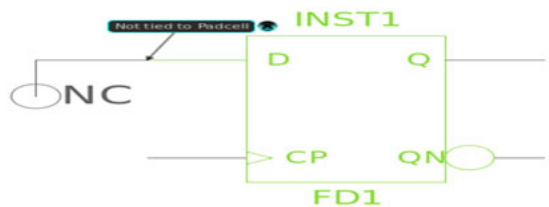
In the following figure, pin D of INST1 is directly connected to pin Z of PadCell.



Therefore, the tag will not report a violation.

Example 4

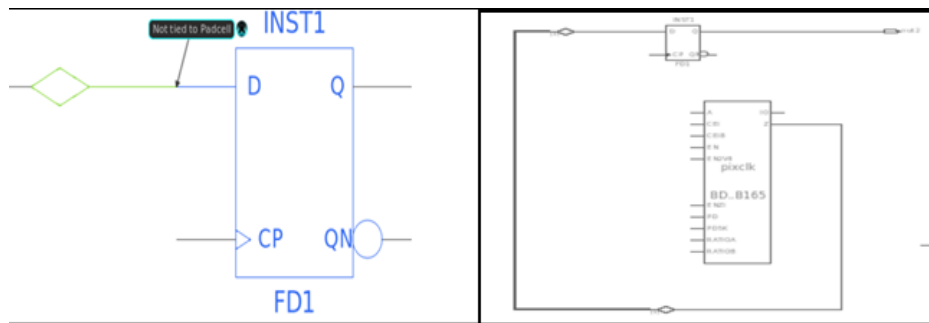
In the following figure, pin D of INST1 FD1 is not connected to PadCell.



Therefore, the tag will report a violation.

Example 5

In the following figure, partial bit is not attached to the Pad Cell.



Therefore, the tag will report a violation.

Default Severity Label

Error

Tag Group

None

Report and Related File

None

Netlist Tags

CellInfo

List of cells used in the design

Description

This tag generates a report called cell_instances.rpt that contains the details about cells and their instances .

Set the following option in the .prj file to generate the report:

```
set_option report cell_instances
```

The following is a sample of this report:

List of cells used in this design					
Cell name	Library	#Instance	Size (gate count)	Size (real count)	
AND2	lsi_10k	4	8.000000	8.000000	
INT_OR2	lsi	4	16.000000	8.000000	
IV	lsi_10k	2	2.000000	2.000000	
Summary Statistics					
Total number of cells	:	10			
Total Gate Count	:	26.00			
Total number of nets	:	0			
Total number of pins	:	28			
Average number of pins per net	:	3.50			
Gate count of sequential cell(s)	:	0.00			
Occurrences of sequential cell(s)	:	0			
Gate count of FF cell(s)	:	0.00			
Occurrences of FF cell(s)	:	0			
Gate count of LATCH cell(s)	:	0.00			
Occurrences of LATCH cell(s)	:	0			
Ratio of sequential cell(s)	:	0.00%			
Gate count of analog cell(s)	:	0.00			
Occurrences of analog cell(s)	:	0			
Gate count of analog pad cell(s)	:	0.00			
Occurrences of analog pad cell(s)	:	0			
Gate count of unknown cell(s)	:	0.00			
Occurrences of unknown cell(s)	:	0			

Language

Verilog, VHDL

Messages and Suggested Fix

The following message appears when this tag generates the cell_instances.rpt report:

```
[INFO] please refer the cell_instances.rpt report
```

Default Severity Label

Info

Tag Group

None

Report and Related File

None

CheckAnalogConnection

Report connection between analog and non-analog pin

Description

The *CheckAnalogConnection* tag reports a violation if an analog pin connects to a non-analog pin. A library pin whose attribute is `Analog_pin: true` or `is_analog: true` is considered as an analog pin.

The tag does not report a violation if an analog pin connects to another analog pin or an input/output port.

Language

Verilog, VHDL

Parameters

None

Messages and Suggested Fix

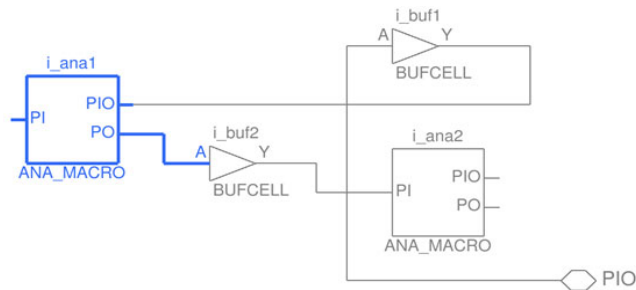
The following message appears when an analog pin is connected to a non-analog pin:

[WARNING] Analog pin <pin name> of instance <instance name> is connected to non-analog pin <pin name> of inst <instance name>

Example Code and/or Schematic

Example 1:

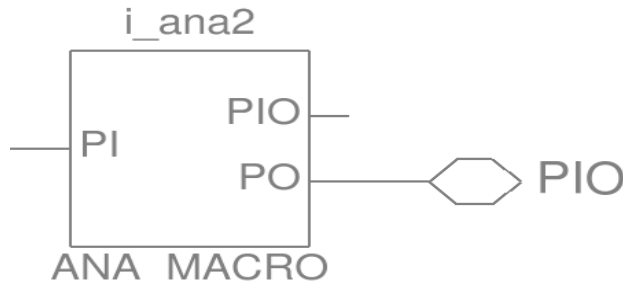
Consider the following schematic:



In the above schematic, the tag reports a violation as the port `PO` of the `ANA_MACRO` cell is an analog pin whereas input `A` of instance `i_buf2` is a non-analog.

Example 2:

Consider the following schematic:



In the above schematic, the tag does not report a violation as the port `PO` of the `ANA_MACRO` cell is an analog pin and `PIO` is an I/O port.

Default Severity Label

Warning

InoutPin

Inout pin of a cell should be directly connected to an inout port or another inout pin

Description

The *InoutPin* tag reports a violation if the inout pin of a cell is not directly connected to a port or an inout pin of an another cell.

Parameters

ignore_cell_list: By default, this parameter is set to "". Set this parameter to a comma-separated list of PERL regular expressions to provide a list of cell names and enable the tag to not report violations for the cell instances.

Messages and Suggested Fix

The following message appears when the inout pin of a cell is not directly connected to a port or an inout pin of an another cell:

[ERROR] Inout pin '<pin_name>/<connected_net_name>' of instance '<instance_hierarchy>' (master : '<cell_name>') is connected to non-inout <pin | port> '<pin_or_port_name>/<connected_net_name>' of instance '<instance_hierarchy>'

Example Code and/or Schematic

Example 1

Consider the following example:

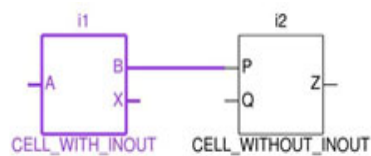
```
cell (CELL_WITH_INOUT) {
  pin(A) {
    direction: input;
  }
  pin(B) {
    direction: inout;
    function: "A";
  }
  pin(X) {
```

```

direction: output;
function: "B";
}
}
cell (CELL_WITHOUT_INOUT) {
pin(P) {
direction: input;
}
pin(Q) {
direction: input;
}
pin(Z) {
direction: output;
function: "P";
}
}
////////////////////////////////////

module test (input in, output out);
wire tmp;
CELL_WITH_INOUT    i1 (.A(), .B(tmp), .X()); // violation -
(inout to input connection)
CELL_WITHOUT_INOUT i2 (.P(tmp), .Q(), .Z());
endmodule

```



Example 2

Consider the following example:

```
cell (CELL_WITH_INOUT)
{
  pad_cell: true;
  pin(A) {
    direction: input;
  }
  pin(B) {
    direction: inout;
    function: "A";
  }
  pin(X) {
    direction: output;
    function: "B";
  }
}

cell (CELL_WITHOUT_INOUT) {
  pin(P) {
    direction: input;
  }
  pin(Q) {
    direction: input;
  }
  pin(Z) {
    direction: output;
    function: "P";
  }
}
```

```

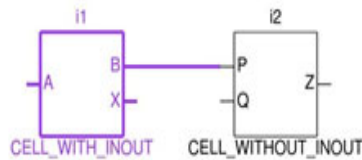
////////////////////////////////////
////////////////////////////////////

```

```

module test (input in, output out);
wire tmp;
CELL_WITH_INOUT i1 (.A(), .B(tmp), .X()); // violation - (inout
to input connection)
CELL_WITHOUT_INOUT i2 (.P(tmp), .Q(), .Z());
endmodule

```



Example 3

Consider the following example:

```

cell (CELL_WITH_INOUT)
{
pin(A)
{
direction: input;
}
pin(B) {
direction: inout;
function: "A";
}
pin(X) {
direction: output;
function: "B";
}
}

```

Netlist Tags

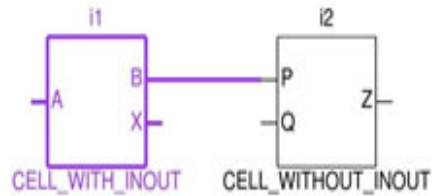
```

    }
  }
  cell (CELL_WITHOUT_INOUT) {
    pad_cell: true;
    pin(P) {
      direction: input;
    }
    pin(Q) {
      direction: input;
    }
    pin(Z) {
      direction: output;
      function: "P";
    }
  }

  //////////////////////////////////////
  //////////////////////////////////////

  module test (input in, output out);
    wire tmp;
    CELL_WITH_INOUT    i1 (.A(), .B(tmp), .X()); // violation -
    (inout to input connection)
    CELL_WITHOUT_INOUT i2 (.P(tmp), .Q(), .Z());
  endmodule

```



Example 4

Consider the following example:

```
cell (CELL_WITH_INOUT) {
  pin(A) {
    direction: input;
  }
  pin(B) {
    direction: inout;
    function: "A";
  }
  pin(X) {
    direction: output;
    function: "B";
  }
}

cell (CELL_WITH_INOUT_2) {
  pin(L) {
    direction: input;
  }
  pin(M) {
    direction: inout;
  }
}
```


Netlist Tags

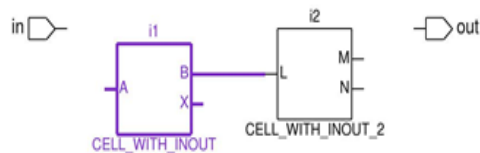
```

function: "L";
}
pin(N) {
direction: output;
function: "M";
}
}

////////////////////////////////////
////////////////////////////////////

module test (input in , output out);
wire tmp;
CELL_WITH_INOUT    i1 (.A(), .B(tmp), .X()); // violation -
(inout to input connection)
CELL_WITH_INOUT_2  i2 (.L(tmp), .M(), .N());
endmodule

```

**Example 5**

Consider the following example:

```

cell (CELL_WITH_INOUT) {
pin(A) {
direction: input;
}
pin(B) {
direction: inout;
}
}

```

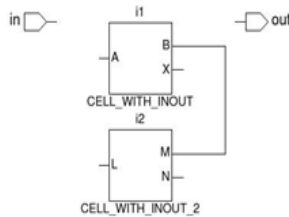
```

function: "A";
}
pin(X) {
direction: output;
function: "B";
}
}
cell (CELL_WITH_INOUT_2) {
pin(L) {
direction: input;
}
pin(M) {
direction: inout;
function: "L";
}
pin(N) {
direction: output;
function: "M";
}
}
////////////////////////////////////
////////////////////////////////////

module test (input in, output out);
wire tmp;
CELL_WITH_INOUT i1 (.A(), .B(tmp), .X()); // no-violation - (inout to
inout connection)
CELL_WITH_INOUT_2 i2 (.L(), .M(tmp), .N());
endmodule

```

Netlist Tags

**Example 6**

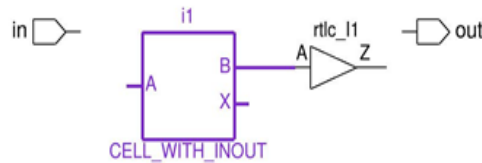
Consider the following example:

```
cell (CELL_WITH_INOUT) {
  pin(A) {
    direction: input;
  }
  pin(B) {
    direction: inout;
    function: "A";
  }
  pin(X) {
    direction: output;
    function: "B";
  }
}

////////////////////////////////////
////////////////////////////////////

module test (input in , output out);
  wire tmp, tmp_2;
  CELL_WITH_INOUT i1(.A(), .B(tmp), .X()); // violation -
  (connects to a gate)
  assign tmp_2 = tmp;
```

```
endmodule
```



Example 7

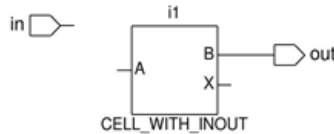
Consider the following example:

```
cell (CELL_WITH_INOUT) {
  pin(A) {
    direction: input;
  }
  pin(B) {
    direction: inout;
    function: "A";
  }
  pin(X) {
    direction: output;
    function: "B";
  }
}

////////////////////////////////////
////////////////////////////////////

module test (input in , output out);
  CELL_WITH_INOUT  i1(.A(), .B(out),.X()); // violation -
  (connects to a non-inout port)
Endmodule
```

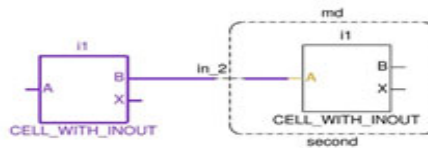
Netlist Tags

**Example 8**

Consider the following example:

```
module test(input in , output out);
wire tmp;
CELL_WITH_INOUT i1(.A(), .B(tmp),.X()); // violation - (inout
to input connection)
second md(.in_2(tmp),.out_2());
endmodule

module second(input in_2, output out_2);
CELL_WITH_INOUT i1(.A(in_2), .B(), .X());
endmodule
```

**Example 9**

Consider the following example:

```
cell (CELL_WITH_INOUT) {
pin(A) {
direction: input;
```

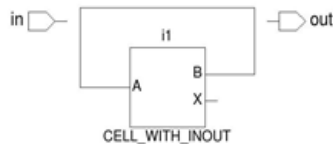
```

}
pin(B) {
direction: inout;
function: "A";
}
pin(X) {
direction: output;
function: "B";
}
}

////////////////////////////////////
////////////////////////////////////

module test (input in, output out);
wire tmp;
CELL_WITH_INOUT i1(.A(tmp), .B(tmp), .X()); // violation -
(inout to input connection)
endmodule

```



Example 10

Consider the following example:

```

cell (CELL_WITH_INOUT) {
pin(A) {
direction: input;
}
}

```

Netlist Tags

```

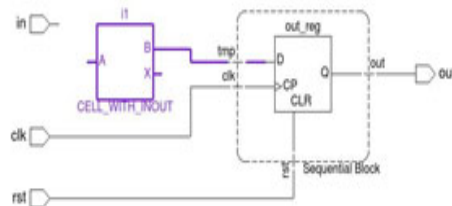
pin(B) {
direction: inout;
function: "A";
}
pin(X) {
direction: output;
function: "B";
}

////////////////////////////////////
////////////////////////////////////

module test (input in,clk,rst, output reg out);
wire tmp;
CELL_WITH_INOUT i1(.A(), .B(tmp), .X()); // violation -
(connects to a sequential block)

always @ (posedge clk or posedge rst)
if(rst)
out <= 0;
else
out <= tmp;
endmodule

```



Example 11

Consider the following example:

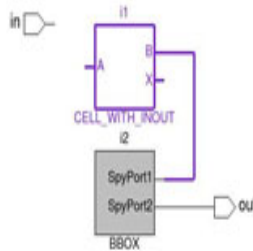
```
cell (CELL_WITH_INOUT) {
pin(A) {
direction: input;
}
pin(B) {
direction: inout;
function: "A";
}
pin(X) {
direction: output;
function: "B";
}
}

////////////////////////////////////
////////////////////////////////////

module test (input in, output out);
wire tmp;
CELL_WITH_INOUT i1 (.A(), .B(tmp), .X()); // violation -
(connects to a blackbox)
BBOX i2 (tmp, out);
```


Netlist Tags

```
endmodule
```

**Example 12**

Consider the following example:

```
cell (CELL_WITH_INOUT) {
  pin(A) {
    direction: input;
  }
  pin(B) {
    direction: inout;
    function: "A";
  }
  pin(X) {
    direction: output;
    function: "B";
  }
}

cell (CELL_WITHOUT_INOUT) {
  pin(P) {
    direction: input;
  }
  pin(Q) {
    direction: input;
  }
}
```

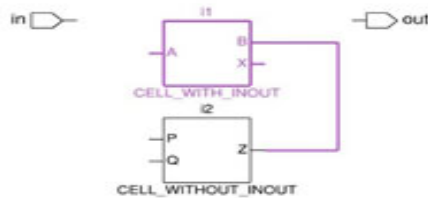
```

}
pin(Z) {
direction: output;
function: "P";
}
}

////////////////////////////////////
////////////////////////////////////

module test (input in, output out);
wire tmp;
CELL_WITH_INOUT i1 (.A(), .B(tmp), .X()); // violation - (inout
to output connection)
CELL_WITHOUT_INOUT i2 (.P(), .Q(), .Z(tmp));
endmodule

```



Example 13

Consider the following example:

```

cell (CELL_WITH_INOUT) {
pin(A) {
direction: input;
}
pin(B) {

```

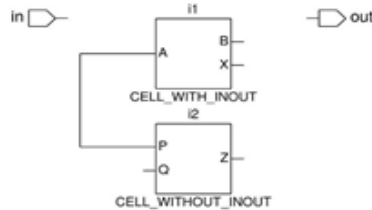
Netlist Tags

```

direction: inout;
function: "A";
}
pin(X) {
direction: output;
function: "B";
}
}
cell (CELL_WITHOUT_INOUT) {
pin(P) {
direction: input;
}
pin(Q) {
direction: input;
}
pin(Z) {
direction: output;
function: "P";
}
}
////////////////////////////////////
////////////////////////////////////

module test (input in, output out);
wire tmp;
CELL_WITH_INOUT i1 (.A(tmp), .B(), .X()); // no-violation - [input (non-
input) to input (non-inout) connection]
CELL_WITHOUT_INOUT i2 (.P(tmp), .Q(), .Z());
endmodule

```



Example 14

Consider the following example:

```

cell (CELL_WITH_INOUT) {
    pin(A) {
        direction: input;
    }
    pin(B) {
        direction: inout;
        function: "A";
    }
    pin(X) {
        direction: output;
        function: "B";
    }
}

////////////////////////////////////
////////////////////////////////////

module test (input in, output out);
wire tmp, tmp_2;
CELL_WITH_INOUT  i1 (.A(), .B(tmp), .X()); // violation -

```

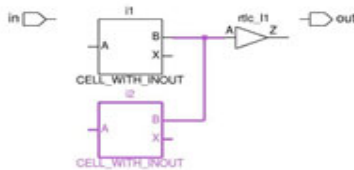
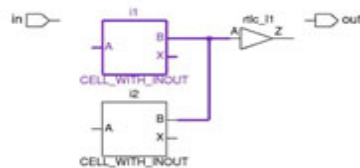
Netlist Tags

(connects to a gate)

```
CELL_WITH_INOUT i2 (.A(), .B(tmp), .X()); // violation -
connects to a gate)
```

```
assign tmp_2 = tmp;
```

```
endmodule
```



Default Severity Label

Error

ModulePinExistence

Module has no input or output ports

When to Use

Use this tag to check for modules that do not have any port interface.

Description

The *ModulePinExistence* tag reports modules that do not have any port interface (no input or output ports).

Parameter(s)

None

Constraint(s)

None

Message Details

Verilog

The following message appears at the location where a module `<module-name>` is defined without any inputs or outputs:

```
Module <module-name> has no ports
```

VHDL

The following message appears at the location where a design unit is defined without any inputs or outputs:

```
Module has no input/output ports
```

Severity

Warning

Suggested Fix

If this is a top-level testbench, there is nothing to fix. Otherwise you can only read to and write from this design unit through global variables, which is unsynthesizable. It may sometimes be OK for simulation monitors

(though would still be considered by many to be poor coding style).

UnmixableCellCheck

Detecting unmixable standard cells in a module

When to Use

Use this tag to detect module that contains different types of cells, which are not defined as valid mixable cells.

Description

This tag ensures that standard cells in a specification module can be mixed. The tag reports violation if a specified module contains the different types of cells, which are not defined as valid mixable cells.

To specify the module to be checked, use the `define_module_list` parameter, as shown below:

```
configure_lint_tag_parameter -tag "UnmixableCellCheck" -
parameter define_module_list -value "<space-separated list
of modules>"
```

If a module is not specified, top module is checked.

The cell group, which can be mixable, is specified using the arguments. The first argument is given the definition of the small group of a cell.

A small group name and the naming tag of a cell are defined.

Example:

Small Group Name	Naming Tag
7GRID_L	CELLA_*
9GRID_L	CELLB_*

The second argument is given the definition of the cell group, which can be mixable. The valid mixable cell group is defined by using small cell group name, as shown in the following example:

Valid mixable group list
7GRID_L 7GRID_M 7GRID_F
9GRID_L 9GRID_M 9GRID_F

Parameter(s)

- `cell_group_list_file`: The default value is . Specify the name of the file containing the small group name and the naming tag of a cell as an input to the parameter.
- `mixable_groups_file`: The default value is . Specify the name of the file containing the space-separated list of mixable cell group names.

Constraint(s)

None

Messages and Suggested Fix

Message 1

Error: Module In 'module-name' <small-group-name> (<cell-naming-rule>) are Unmixable with list of <small-group-name>(<cell-naming-rule>)

Potential Issues

The error message is reported when there are unmixable standard cells in a module.

Consequences of Not Fixing

Currently Unavailable

How to Debug and Fix

Click the violation message to cross-probe to the location of violated cell instance in the given netlist design.

In addition, view the Incremental Schematic to analyze the violating cell instance.

To fix the violation, ensure that the standard cells in a specification module can be mixed.

Message 2

Info: Setup Error in file <mixable_groups_file_name>, mixable groups name shall be mutually exclusive.

Potential Issues

This is an informational message. There are no potential issues related to the violation.

Consequences of Not Fixing

This is an informational message. There are no consequences of not fixing the violation.

How to Debug and Fix

This is an informational message.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Error

ExternalPortAndIOPinCheck

Pad pin of I/O should be directly connected with external port

Description

The *ExternalPortAndIOPinCheck* tag ensures that the pad pin of I/O is connected with an external port (input/inout/output) directly.

The tag ignores SpyGlass-generated buffer and inverter in fanin and fanout cone of the pad pin of I/O cell.

The tag reports a violation when a pad pin of I/O is not connected anywhere.

Language

Verilog

Parameters

None

Messages and Suggested Fix

Message 1

The following message appears when pad pin of I/O is not directly connected with external port :

[ERROR] '<pad-pin>' of '<IO-instance>'(<cell-name>) is not connected with external port directly. Current connection: '<instance-name>' '<pin-name>' pin

Message 2

The following message appears when pad pin of I/O is not connected anywhere :

[ERROR] '<pad-pin>' of '<IO-instance>'(<Cell-Name>) is not connected anywhere

Default Severity Label

Error

ExternalPortBranchCheck

Branching of the net which is directly connected to an external port should be avoided

Description

The *ExternalPortBranchCheck* tag ensures that a single net connection exists between an external input/inout/output port of a top module and an initial cell. This tag reports an error message if this net has multiple branch connections.

The [excluded_cell_list](#) parameter specifies the list of comma-separated excluded cells.

The `excluded_check_port` parameter specifies the names of ports for which tag checking should not be done. It can be specified as below

```
configure_lint_tag_parameter -tag "ExternalPortBranchCheck"  
-parameter EXCLUDED_CHECK_PORT -value { top.out }
```

The parameter `clk_rst_net` specifies the names of clock or reset nets. The tag will not report violation for external port if its branch reaches to clock/reset net. It can be specified as below

```
configure_lint_tag_paramceter -tag "ExternalPortBranchCheck"  
-parameter CLK_RST_NET -value { top.clk }
```

Language

Verilog

Parameters

- [excluded_cell_list](#): By default, this parameter is not set. Set this parameter to a comma-separated list of excluded cell names. When a net is connected to these cells, it is not considered as valid.
- [excluded_check_port](#): By default, this parameter is not set. Set this parameter to a space-separated list of ports names for which tag checking should not be done.
- [clk_rst_net](#): By default, this parameter is not set. Set this parameter to a to specify a space-separated list of clock/reset nets. The tag will not report a violation for the external port if its branch reaches to clock/reset net.

Messages and Suggested Fix

The following message appears when a single net connection does not exist between an external input/inout/output port of a top module and an initial cell :

```
[ERROR] Multiple branches existing on net '<net-name>'
connecting <input | input | output> port '<port-name>'
<'<instance-hierarchical-name/port-name>(<cell-master-name/
External Port>)'>
```

Default Severity Label

Error

InoutIOOutput

Output pin of inout cell has been connected to clock/reset/set/enable pin of a flop/latch

Description

The *InoutIOOutput* tag reports a violation when the clock/reset/set/enable signal of the flop/latch is connected to an output pin of an inout cell. In addition to the direct assignment, the rule checks only through buffers and inverters.

Inout cell is a pad cell `pad_cell:true`, which has at least one pin with `direction:inout` and `is_pad:true`.

Language

Verilog, VHDL

Parameters

None

Messages and Suggested Fix

The following message appears when the clock/reset/set/enable signal of the flop/latch is connected to an output pin of an inout cell:

```
[ERROR] Inout cell's out pin '<<cell_instance_hierarchy>/<pin_name>>' connected to <clock | reset | set | enable> signal '<<signal_name>/<pin_name>>' of <flop | latch> '<instance_hierarchical_name>'
```

Example Code and/or Schematic

Example 1

Consider the following example:

```
// library cell //  
library (tmp)  
{  
  cell (IOBUF)  
  {
```

Netlist Tags

```

pad_cell : true; # 'pad_cell' specified
pin(A)
{
direction : input;
}
pin(B)
{
direction : inout; # 'inout' pin
function : "A";
is_pad : true; # 'is_pad' specified
}
pin(X)
{
direction : output;
function : "B";
}
}
}
//////////

module test(clk,rst,d,out);
input clk,rst,d;
output out;
reg tmp,b,c;
IOBUF io_clk (.A(clk),.B(b),.X(c));
always@(posedge c or posedge rst)
begin
if(rst) tmp <= 0;
else tmp <= d;

```

```
end
assign out = tmp;
endmodule
```

In the above example, the tag reports a violation as `is_pad` is specified with an inout pin.

Example 2

Consider the following example:

```
// library cell //
library (tmp)
{
  cell(IOBUF)
  {
    pad_cell : true; # 'pad_cell' specified
    pin(A)
    {
      direction : input;
    }
    pin(B)
    {
      direction : inout; # 'inout' pin
      function : "A";
      'is_pad' not specified with an 'inout' pin
    }
    pin(X)
    {
      direction : output;
      function : "B";
      is_pad : true; # 'is_pad' specified
    }
  }
}
```


Netlist Tags

```

}
}
//////////

module test(clk,rst,d,out);
input  clk,rst,d;
output out;
reg tmp,b,c;
IOBUF io_clk (.A(clk),.B(b),.X(c));
always@(posedge c or posedge rst)
begin
if(rst) tmp <= 0;
else tmp <= d;
end
assign out = tmp;
endmodule

```

In the above example, the tag does not report a violation as `is_pad` is not specified with an inout pin.

Example 3

Consider the following example:

```

// library cell //
library (tmp)
{
cell(IOBUF)
{
pad_cell : true; # 'pad_cell' specified
pin(A)
{

```

```
direction : input;  
}
```

no 'inout' pin with 'is_pad' attribute

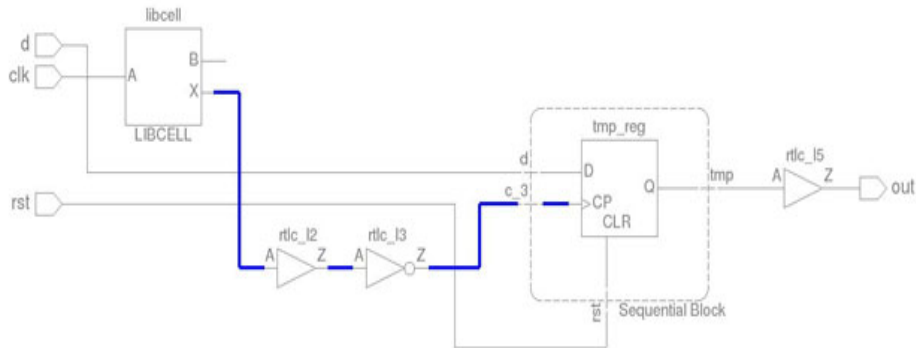
```
pin(X)  
{  
direction : output;  
function : "B";  
is_pad : true; # 'is_pad' specified  
}  
}  
}  
//////////
```

```
module test(clk,rst,d,out);  
input clk,rst,d;  
output out;  
reg tmp,b,c;  
IOBUF io_clk (.A(clk),.X(c));  
always@(posedge c or posedge rst)  
begin  
if(rst) tmp <= 0;  
else tmp <= d;  
end  
assign out = tmp;  
endmodule
```

In the above example, the tag does not report a violation as `is_pad` is not specified with an inout pin.

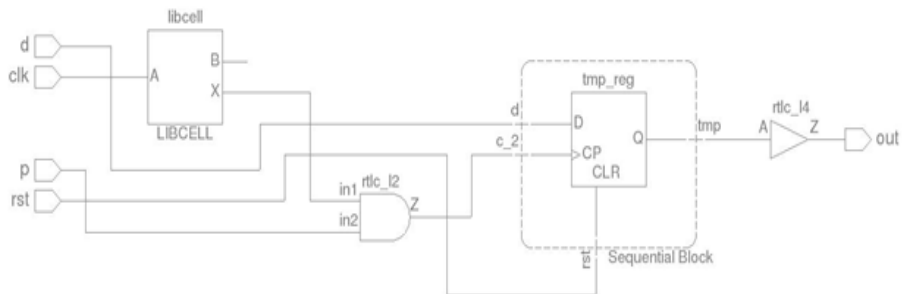
Example 4: Traverse through buffers/ inverters (violation)

Consider the following schematic:



Example 5: (no violation)

Consider the following schematic:



Default Severity Label

Error

InitialConnectionCheck

Detects the initial cell which does not connect directly to an external port

Description

The *InitialConnectionCheck* tag ensures that port of a module connects directly to a valid initial cell.

The valid initial cell is specified to this tag by the [valid_cell_list](#) parameter.

The invalid initial cell for warning is specified to this tag using the [invalid_cell_list_warning](#) parameter.

The tag checking is done for external ports only.

Language

Verilog

Parameters

- [valid_cell_list](#): By default, this parameter is not set. Set this parameter to a comma separated list of strings to define the valid cells that can be either cell type or PERL regular expression for cell names.
- [invalid_cell_list_warning](#): By default, this parameter is not set. Set this parameter to a comma separated list of strings to define the invalid cells that can be either cell type or PERL regular expression for cell names.

Messages and Suggested Fix

Message 1

The following message appears when port of a module does not directly connect to a valid initial cell :

[ERROR] External port <port-name> of module <module-name> is connected to cell(cell-name)

Message 2

The following message appears when port of a module is connected to an invalid cell:

[warning] External port <port-name> of module <module-name> is connected to cell(cell-name)

Netlist Tags

Default Severity Label

Error

HierarchicalPathLengthCheck

Too long hierarchical path name is not allowed

Description

The *HierarchicalPathLengthCheck* tag reports a violation when hierarchical path name length exceeds the specified limit of characters.

This tag ensures that hierarchical path name length does not exceed the specified limit.

You can specify the limit number using the [hier_path_length_limit](#) parameter.

NOTE: *Hierarchical delimiter is treated as a part of hierarchical path name. Dot (.) is considered as hierarchical delimiter in VC SpyGlass.*

Language

Verilog

Parameters

[hier_path_length_limit](#): By default, this parameter is set to 1024. Set this parameter to a positive integer value to define the maximum limit of the length of a hierarchical path name.

Messages and Suggested Fix

The following message appears when a hierarchical path name exceeds the specified character limit:

```
[ERROR] hierarchical path name ('<hier-path>') exceeded the  
length limit('<number>')
```

Default Severity Label

Error

SameNetConnectionCheck

Detects the circuit which affects delay calculation

Description

If the branches of a net, driven either by a technology cell or primary input/inout port, are reaching as input pins to same technology cell instance then these pins are called as equivalent pins.

The circuit in which equivalent pins are connected to the same net, is defined as same-net connection. The input pin of this cell changes simultaneously. This causes the mistaken cell delay calculation, because the cell delay by this simultaneous change cannot be defined in a cell library.

The *SameNetConnectionCheck* tag reports a violation when a same-net connection is found in the Verilog netlist. However, no violation is reported if the same-net is undriven or connected to supply.

When you specify the net or terminal or instance, only the path (fanout cone) from the specification point is checked. If nothing is specified, the tag checks all the nets of the design.

During the traversal, VC SpyGlass-generated buffers are ignored.

Language

Verilog

Parameters

- ***DEFINE_SAME_NET_START_POINT***: The default value of this parameter is empty string. When you specify the net name connected to the input port or output pin name of a cell or cell name using this parameter, only the path (fanout cone) from the specification point is checked.

If this parameter is empty then the tag will check for all the nets of the design. If this parameter contains an invalid name then the tag reports the following Info Message.

Setup Issue: '<NODE_NAME>' is not a valid name

- ***effort_level***: Default value is 100. Set the value of the parameter to a positive integer value to specify the effort that VC SpyGlass should make while finding the offending issue. Using the parameter enables

you to restrict the traversal through the specified net while performing a check.

Messages and Suggested Fix

The following message appears when a circuit that affects delay calculation is detected :

[ERROR] Equivalent pins of a gate instance connected to same signal. The circuit is constituted by the following.

Driving pin : <instance-name> <pin-name> pin

Loaded pin :<instance-name> <pin-name> pin and <pin-name> pin

Default Severity Label

Error

GlitchGenCircuitCheck

Detects incorrectly used glitch generation circuits

Description

The *GlitchGenCircuitCheck* tag reports a violation if incorrectly used glitch generation circuits are detected.

If the branches of a net, driven either by a technology cell (any type) or primary input/inout port (source design object (SDO)) are reaching as inputs to same technology cell instance (destination design object (DDO)) then the pins of the DDO are called as equivalent pins.

The following is the criterion for a design structure to be considered as a valid glitch generation circuit:

- The net branches reaching to equivalent input pins of the destination cell.
- The converging net branches can only contain buffers and/or inverters. The tag checks for both library and SpyGlass-generated buffers and inverters. The tag checks if the destination pin (DDO) is inverting or not, which is also treated as inverter.
- The converging net branches should carry signal with opposite phases. Which means, one path has even and the other path has odd number of inverters, if destination cell is any of AND, OR, NAND, and NOR.

If the destination cell is of XOR and XNOR type, it still is considered as a valid glitch generation circuit irrespective of phase.

NOTE: *An SDO should have a valid driver and should not be undriven.*

The tag performs the forward traversal in the fanout cone of the destination cell of the glitch generation circuit until it hits a black box or sequential cell or IO cell or open net or primary output port and generates appropriate violation message.

The tag reports an **Error** message when the rule traversal stops at:

- Clock pin of the sequential cell (FLOP, LATCH, RAM, ROM, PLL, Gated Clock Buffer, etc.)
- Asynchronous pin, like clear/pre-set pin of the flip-flop/latch

- Input pin of an IO cell
- Open/unconnected net (having no load)
- Primary output port
- Black box

NOTE: *The tag will report a Error message even when the traversal stops at any point except the ones mentioned above.*

Language

Verilog

Parameters

- [*DEFINE_SAME_NET_START_POINT*](#): Default value is " ". Set the value of the parameter to specify the net name connected to the input port or output pin name of a cell or cell name. When specified, the tag checks only the path (fanout cone) from the specification point.
- [*report_all_objects*](#): Default value is no. Set the value of the parameter to yes to report violation for all the objects in DDO fanout.
- [*effort_level*](#): Default value is 100. Set the value of the parameter to a positive integer value to specify the effort that VC SpyGlass should make while finding the offending issue. Using the parameter enables you to restrict the traversal through the specified net while performing a check.

Messages and Suggested Fix

The following message appears when an incorrectly used glitch generation circuits is detected :

```
[ERROR] The circuit is constituted by the following. Driving
pin : <SDO-instance-name>(<master-of-SDO>, [output pin of SDO])
Loaded pin: <DDO-instance-name> (<masterof-DDO>, [<pin1>,
<pin2>]) Connected to <instance-name> <pintype> <pin-name>
```

The details of the violation message are reported in the [*glitch_circuit*](#) report.

Also, if the [*DEFINE_SAME_NET_START_POINT*](#) parameter contains invalid name, then the tag will report the following message:

Setup Issue: '<NODE_NAME>' is not a valid name

Netlist Tags

Default Severity Label

Error

Functional Lint Tags

VC SpyGlass Functional Lint allows you to perform lint checks in Formal flow while reducing the noise, thereby improving the accuracy of the results using formal techniques.

To enable VC SpyGlass Functional Lint, use the following app var:

```
set_app_var lint_functional_mode true
```

For more information on VC SpyGlass Functional Lint, see VC SpyGlass Lint User Guide.

To enable a tag (formal), provide the `-formal` switch with the rule.

```
configure_lint_tag -enable -tag <tag-name> -goal <goal_name>
-formal
```

The following tags are available in VC SpyGlass Functional Lint.

Rule Name	Category
W164a	Width Mismatch
W362	Width Mismatch
W263	Width Mismatch
W116	Width Mismatch
W110	Width Mismatch

Rule Name	Category
STARC05-2.1.3.1	Width Mismatch
STARC05-2.10.3.2a	Width Mismatch
STARC05-2.2.3.3	Multi Assignment
sim_race02	Multi Assignment
W415a	Multi Assignment
W71	Case Item Coverage
DuplicateCaseLabel-ML	Case Item Coverage
SignedUnsignedExpr-ML	Signed vs Unsigned Usage
W467	Width Mismatch
W551	Case Item Coverage
ImproperRangeIndex-ML	Width Mismatch
MissingFsmStateTransition	Case Item Coverage
MissingFsmStateTransition	FSM rule check
NotReachableFsmState	FSM rule check
NoExitFsmState	FSM rule check
RegisterStuckInResetState-ML	Register state coverage
DeadCode-ML	Conditional Statement Coverage
MissingCaseItems-ML	Reports violations for reachable case items that are not specified

W164a

For detailed information on W164a tag, see [W164a](#).

Consider the following example:

```
module top (clk, reset_b, wait_a);  
  input clk;  
  input reset_b;  
  output reg[1:0]wait_a;  
  
  always @(posedge clk or negedge reset_b)  
  begin  
    if (~reset_b)  
      wait_a <= 2'b00;  
    else  
      begin  
        if (!wait_a[1])  
          wait_a <= wait_a + 1'b1;  
        else  
          wait_a <=2'b00;  
        end  
      end  
    end  
  endmodule
```

In the above example, the LHS width is less than the RHS width for the following expression:

```
wait_a <= wait_a + 1'b1;
```

Therefore, the *W164a* rule reports the above assignment.

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: W164a | file: test.v | line: 14 | MODULE: top |  
HDL_LHS_SIZE: 2 | HDL_RHS_SIZE: 3 | HIERARCHY::top |  
HDL_LHS_EXPR: wait_a | HDL_RHS_EXPR: (wait_a + 1'b1)
```

```
STATUS: NOT_REPORTED
```

```
Reason: Violation is not reported because of the  
following "Related FLPs" status.
```

```
Related FLPs: top.expr_mismatch_0: PROVEN
```

The possible values for `wait_a` are `2'b00`, `2'b01` and `2'b10`.

```
if (~reset_b)
```

```
wait_a <= 2'b00;
```

Therefore, width of the `wait_a + 1'b1` expression cannot be greater than three. The width mismatch violation for the above mentioned assignment is a noise and violation is not reported because the FLP related to the violation is `PROVEN`.

W362

For detailed information on W362 tag, see [W362](#).

Consider the following example:

```
module top(out,in1,in2,clk,in);
input in, clk;
input [1:0] in1;
input [3:0] in2;
output reg [2:0] out;

always @(posedge clk)
begin
    if((4'b0010*in1)==in2)
        begin
            out=4'b0010*in1;
        end
end
endmodule
```

In the above example, the W362 rule reports a violation for unequal length or width mismatch in the following expression:

```
((4'b0010*in1)==in2)
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: W362 | file: test10.v | line: 12 | MODULE: top |
OP_TYPE: == | HDL_LHS_EXPR: (4'b0010 * in1) | HDL_LHS_SIZE: 6
| HDL_RHS_EXPR: in2 | HDL_RHS_SIZE: 4 | HIERARCHY::top
STATUS: NOT_REPORTED
Reason: Violation is not reported because of the
```

following "Related FLPs" status.

Related FLPs: top.expr_mismatch_1: PROVEN

In above example `in1` is two bit and `in2` is four bits. The left hand side expression is `4'b0010*in1` and the maximum value of that expression is six (max value of `in1` is three). Therefore, the left hand side width does not exceed the right hand side width. The violation is also not reported because the FLP related to the violation is `PROVEN`.

W263

For detailed information on W263 tag, see [W263](#).

Consider the following example:

```
module top(in,out,sel,in1,in2);
  input in;
  output reg out;
  input [2:0] sel;
  input [3:0] in1,in2;

  parameter [2:0] C=3;
  parameter [5:0] D=4;

  always @(*)
  begin
    if(in1<4'b1000)
    begin
      case(sel)
        C:out=in;
        D:out=in;
      endcase
    end
  end
endmodule
```

In the above example, the W263 rule reports a violation for the following expression as a case expression width does not match a case select expression width:

```
D:out=in;
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: W263 | file: test8.v | line: 15 | MODULE: top |  
LINT_EXPRESSION: D | EXP_SIZE_RTL: 6 | NODE_TYPE: sel |  
HDL_EXPR_SIZE: 3 | HIERARCHY::top |  
MULTI_LINE_STMT_LIST::test8.v::25:-:test8.v::21:
```

```
STATUS: NOT_REPORTED
```

Reason: Violation is not reported because of the following "Related FLPs" status.

Related FLPs:

```
top.expr_mismatch_0: PROVEN
```

In above example, the width of parameter is defined as six bits. But, the value of the parameter is four. The actual width of the case expression D is three. The width of the `sel` (case selector) is three. Therefore, case expression width does not exceed the case select expression width. The rule does not report violation because the FLP related to the violation is PROVEN.

W116

For detailed information on W116 tag, see [W116](#).

Consider the following example:

```
module top(in,out,a,b,d);
input [1:0] in,b;
input d,a;
output [1:0] out;
supply0 c;
    assign out=(in==c?{c,d}:c);
endmodule
```

In the above example, the *W116* rule reports a violation for the following assignment:

```
assign out=(in==c?{c,d}:c);
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: W116 | file: test8.v | line: 6 | MODULE: top |
SIGNAL?: | HDL_LHS_EXPR: {c, d} | HDL_LHS_SIZE: 2 |
HDL_RHS_EXPR: c | HDL_RHS_SIZE : 1 | HIERARCHY::top
```

```
STATUS: NOT_REPORTED
```

```
Reason: Violation is not reported because of the
following "Related FLPs" status.
```

```
Related FLPs:
```

```
top.expr_mismatch_1: PROVEN
```

In above example, the value of `c` is `1'b0`. Hence the width of `{c, d}` is equal to one. The width of `c` and `{c,d}` is equal. The tag does not report violation, because the FLP related to the violation is PROVEN.

W110

For detailed information on W110 tag, see [W110](#).

Consider the following example:

```
module top(in1,in2,out);
input [2:0] in1,in2;
output reg [4:0] out;
supply1 a;
wire [3:0] g;
    assign g={1'b1,in1};
    test x1(~g,(g ^ {1'b1,in1}),out,a);
endmodule

module test(in1,in2,out,in);
input [2:0] in1,in2;
output reg [4:0] out;
input in;
    always @(*)
        begin
            if(in)
                out=((in1*in2)/2'b10);
            else
                out=in1*in2;
        end
endmodule
```

In the above example, the *W110* rule reports a violation for the following assignment:

```
test x1(~g,(g ^ {1'b1,in1}),out,a);
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: W110 | file:./test.v | line: 8 | MODULE: top |
MOD_PORT_NAME_RTL: in1 | EXP_SIZE_RTL: 3 | HDL_INST_NAME: x1
| NODE_TYPE_1: (~g) | HDL_EXPR_SIZE: 4 | HIERARCHY::top |
MODULE_NAME_RTL: test | NODE_TYPE: module
```

STATUS: PROVEN

Reason: Waiving violation as all the below FLPs are proven.

Related FLPs:

```
top.expr_mismatch_3: PROVEN
```

In above example, the width of the signal `g` is four and the width of the signal `in1` is three. But, the first bit of the signal `g` is one as follows:

```
assign g={1'b1,in1};
```

In the following assignment, the first bit of `~g` is zero and therefore the width of `~g` is equal to three.

```
test x1(~g, (g ^ {1'b1,in1}),out,a);
```

The width of `in1` and `~g` is not equal, but the rule does not report a violation because the FLP related to the violation is `PROVEN`.

STARC05-2.1.3.1

For detailed information on STARC05-2.1.3.1 tag, see [STARC05-2.1.3.1](#).

Consider the following example:

```
module top(in1,in2,out);
  input [3:0] in1;
  input [4:0] in2;
  output reg [5:0] out;

  function [5:0] and1;
    input [3:0] arg1;
    input [4:0] arg2;
    begin
      and1=arg1+arg2;
    end
  endfunction

  always@(*)
    begin
      out=and1((in2/2),(in1*2));
    end
endmodule
```

In the above example, the *STARC05-2.1.3.1* rule reports a violation for the following assignment:

```
out=and1((in2/2),(in1*2));
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:


```
Rule: STARC05-2.1.3.1 | file: test.v | line: 18 | MODULE: top  
| NODE_TYPE: (in2 / 2) | EXP_SIZE_RTL: 5 | NODE_TYPE_1: arg1  
| HDL_EXPR_SIZE: 4 | HIERARCHY::top |  
MULTI_LINE_STMT_LIST::test.v::18:-:test.v::8:
```

```
    STATUS: PROVEN
```

```
    Reason: Waiving violation as all the below FLPs are  
proven.
```

```
    Related FLPs:
```

```
        top.expr_mismatch_0: PROVEN
```

In above example, the width of the `arg1` is four bit and the width of the `in2` is five. The width of the maximum value of `in2/2` is four. Therefore, the bit-width mismatch violation for the above mentioned assignment is a noise and violation is not reported because the FLP related to the violation is PROVEN.

STARC05-2.10.3.2a

For detailed information on STARC05-2.10.3.2a tag, see [STARC05-2.10.3.2a](#).

Consider the following example:

```
module top(a,b,c,out,in);
input [2:0] a;
input [3:0] b;
input in;
output reg out;
input [3:0] c;
wire [1:0] d=2'b10;

always @(*)
begin
    if(c>3'b101)
        begin
            if(d || ((c/2'b11)-2'b10))
                out=in;
        end
    else
        begin
            if(d || ((c/2'b11)-2'b10))
                out=in;
        end
    end
end
endmodule
```

In the above example, the *STARC05-2.10.3.2a* rule reports a violation for the following expression:

```
if(d || ((c/2'b11)-2'b10))
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: STARC05-2.10.3.2a | file: test13.v | line: 14 | MODULE:
top | OP_NAME: || | LINT_EXPRESSION: d | EXP_SIZE_RTL: 2 |
HDL_EXPRESSION_RTL : ((c / 2'b11) - 2'b10) | HDL_EXPR_SIZE: 5
```

```
| HIERARCHY::top
```

```
    STATUS: NOT_REPORTED
```

```
    Reason: Violation is not reported because of the  
following "Related FLPs" status.
```

```
    Related FLPs:
```

```
        top.expr_mismatch_1: PROVEN
```

In above example, the width of `c` is four and the maximum value is 15. The maximum value of the expression `((c/2'b11)-2'b10)` is three and the width of the maximum value of that expression is two. Therefore, the bit-width mismatch violation for the above mentioned assignment is a noise. The violation is not reported because the FLP related to the violation is PROVEN.

STARC05-2.2.3.3

For detailed information on STARC05-2.2.3.3 tag, see [STARC05-2.2.3.3](#).

Consider the following example:

```
module top(in1,in2,out,clk);
input in1,in2;
output reg out;
input clk;
supply0 b;
always @(posedge clk) begin
    if(b) begin
        out<=in1;
    end
    out<=in2;
end
endmodule
```

In the above example, the *STARC05-2.2.3.3* rule reports a violation for the following assignment:

```
out<=in1;
out<=in2;
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: STARC05-2.2.3.3 | file: test.v | line: 13 | MODULE: top
| CLK_REG_RTL: out | C_LINE_NUMBER: 11
```

```
STATUS: NOT_REPORTED
```

```
Reason: Violation is not reported because of the
following "Related FLPs" status.
```

```
Related FLPs:
```

```
top.multi_assign_0: PROVEN
```

In the above example, the value of *b* is zero and therefore the first

assignment never happened. The violation for the above mentioned assignment is a noise and is not reported because the FLP related to the violation is `PROVEN`.

sim_race02

For detailed information on sim_race02 tag, see [sim_race02](#).

Consider the following example:

```
module top(in,out,a,clk);
  input [3:0] in;
  input clk,a;
  wire w1,w2;
  output reg out;

  assign w1=&in;
  assign w2=~&in;

  always @(posedge clk)
  begin
    if(w2)
      begin
        out<=in;
      end
  end

  always @(posedge clk)
  begin
    if(w2==1'b0)
      begin
        out<=in;
      end
  end
end
```

```

        end
    end
endmodule

```

In the above example, the *sim_race02* rule reports a violation for the following assignment:

```
out<=in;
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```

Rule: sim_race02 | file: test30.v | line: 32 | MODULE: top |
SIGNAL: out | LINE_NUM_RTL: 32 | C_LINE_NUMBER: 21 |
MULTI_LINE_STMT_LIST::test30.v::32::-:test30.v::21:

```

```
STATUS: NOT_REPORTED
```

Reason: Violation is not reported because of the following "Related FLPs" status.

Related FLPs:

```
top.multi_assign_3: PROVEN
```

In the above example, the first if condition execute with $w2 = 1$ and the second expression execute with $w2 = 0$. Therefore, the multi assignment violation for the above case is a noise and the violation is not reported because the FLP related to the violation is `PROVEN`.

W415a

For detailed information on W415a tag, see [W415a](#).

Consider the following example:

```
module top(in1,in2,out,clk,in3,in);  
  input in1,in2,in3,in,clk;  
  output reg out;  
  wire a, b;
```

```
  assign a=(in?1'b0:1'b1);  
  assign b=(in?1'b1:1'b0);
```

```
  always @(posedge clk)  
  begin  
    out<=in1;  
    if(a==b)  
      out<=in2;  
  end  
endmodule
```

In the above example, the *W415a* rule reports a violation for the following assignment:

```
out<=in1;  
out<=in2;
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: W415a | file: test.v | line: 13 | MODULE: top | SIGNAL:  
out | C_LINE_NUMBER: 11 | HIERARCHY::top |  
MULTI_LINE_STMT_LIST::test.v::13:-:test.v::11:
```


STATUS: PROVEN

Reason: Waiving violation as all the below FLPs are proven.

Related FLPs:

top.multi_assign_1: PROVEN

In the above example, the signal `a` and `b` are complement of each other. Therefore `(a==b)` condition can not happen. Also, multi assignment condition can not happen. Therefore, the violation for the above case is a noise and the violation is not reported because the FLP related to the violation is PROVEN.

W71

For detailed information on W71 tag, see [W71](#).

Consider the following example:

```
module top(in,out,sel);  
  input in;  
  output reg out;  
  input [1:0] sel;
```

```
  always @(*)  
    begin  
      case(sel)  
        2'b00:out=in;  
        2'b01:out=in;  
        2'b10:out=in;  
        2'b11:out=in;  
      endcase  
    end  
endmodule
```

In the above example, the *W71* rule reports a violation for the following case block:

```
case(sel)
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: W71 | file: test.v | line: 9 | MODULE: top | NODE_TYPE:  
in combinational block | HIERARCHY::top
```

```
STATUS: PROVEN
```

```
Reason: Waiving violation as all the below FLPs are
```

proven.

Related FLPs:

`top.default_case_0: PROVEN`

In the above example, all possible case items are defined for case selector. Therefore, the *W71* rule violation is a noise and the violation is not reported because the FLP related to the violation is `PROVEN`.

Therefore, the associated violation for the above FLP is `PROVEN`.

DuplicateCaseLabel-ML

For detailed information on DuplicateCaseLabel-ML tag, see [DuplicateCaseLabel-ML](#).

Consider the following example:

```
module top(in,out);
  input in;
  output reg out;
  wire [2:0] sel;
  wire a;
  wire b;
  buf x1(a,in);
  not x2(b,in);

  always @(*)
    begin
      casez({a,b}) //synopsys parallel_case
        2'b00:out=in;
        2'b0?:out=in;
        2'b10:out=in;
        2'b1?:out=in;
      endcase
    end
endmodule
```

In the above example, the *DuplicateCaseLabel-ML* rule reports a violation for the following case items:

```
2'b00:out=in;
2'b0?:out=in;
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: DuplicateCaseLabel-ML | file: test.v | line: 14 |  
MODULE: top | NODE_TYPE : 2'b0? | NODE_TYPE_1: 2'b00 |  
MULTI_LINE_STMT_LIST::test.v::14:-:test.v::13:
```

```
STATUS: PROVEN
```

```
Reason: Waiving violation as all the below FLPs are  
proven.
```

```
Related FLPs:
```

```
top.parallel_case_0: PROVEN
```

In the above example, x1 is a buffer, x2 is a NOR gate; a and b are invert signals to each other. Therefore, 2'b00 case never happens and multi assignment violation is a noise violation. The violation is not reported because the FLP related to the violation is `PROVEN`.

SignedUnsignedExpr-ML

For detailed information on *SignedUnsignedExpr-ML* tag, see [SignedUnsignedExpr-ML](#).

Consider the following example:

```
module top (d, q);
    input d;
    output q;
    reg r, clk;
    parameter p1 = 1'b0;
    parameter p2 = 1'sb0;
    parameter p3 = 2;
    integer i, j;
    wire wireVar;
    mid INST1(.in(d + i),.out(q));
    assign r = myFunc(p1 + p2);
        function myFunc;
            input i;
            assign myFunc = 1;
        endfunction
endmodule

module mid(in, out);
    input in;
    output out;
    reg out;
    assign out = in;
endmodule
```

In the above example, the *SignedUnsignedExpr-ML* rule reports a violation for the following assignment:

```
assign r = myFunc(p1 + p2);
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: SignedUnsignedExpr-ML | file: test.v | line: 11 |  
MODULE: top | LINT_EXPRESSION: p1 | HDL_EXPRESSION_RTL: p2
```

```
STATUS: PROVEN
```

```
Reason: Waiving violation as all the below FLPs are  
proven.
```

```
Related FLPs:
```

```
top.expr_mismatch_3: PROVEN
```

In the above example, *p1* is unsigned and *p2* is signed and the value of the *p2* is zero. Therefore, the *SignedUnsignedExpr-ML* violation for the above example is a noise violation. The violation is not reported because the FLP related to the violation is *PROVEN*.

W467

For detailed information on W467 tag, see [W467](#).

Consider the following example:

```
module top (a, c, clk, out);
  input c, clk;
  input [1:0]a;
  wire [1:0]b;
  output reg [3:0] out;
  reg [3:0] d;

  assign b = a | 2'b11;

  always @(c or b)
  begin
    case (b)
      2'b00: out = a || d[0];
      2'b01: out = a && d[1];
      2'b11: out = a && d[3];
      default: out= 3'b11?;
    endcase
  end

endmodule
```

In the above example, the *W467* rule reports a violation for the following assignment:

```
default: out= 3'b11?;
```


Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: W467 | file: top.v | line: 16 | MODULE: top |  
VALUE_RTL: 3'b11?
```

```
STATUS: NOT_REPORTED
```

```
Reason: Violation is not reported because of the  
following "Related FLPs" status.
```

```
Related FLPs:
```

```
top.dont_care_0: PROVEN
```

In the above example, `b` is defined as `assign b = a | 2'b11;` and therefore the value of `b` can never be `2'b10`. Therefore, the W467 violation for the above example is a noise violation. The violation is not reported because the FLP related to the violation is `PROVEN`.

W551

For detailed information on W551 tag, see [W551](#).

Consider the following example:

```
module top(inp, outp, sel);
    input [3:0] inp;
    input [1:0] sel;
    output outp;
    reg outp;

    always @(sel or inp)
        casex(sel) //synopsys full_case
            2'bx0: outp = inp[0];
            2'b?1: outp = inp[1];
            default: outp = inp[3];    //Violation proven
        endcase
endmodule
```

In the above example, the *W551* rule reports a violation for the following assignment:

```
casex(sel) //synopsys full_case;
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: W551 | file: top.v | line: 29 | MODULE: top |
NODE_TYPE: full_case
STATUS: NOT_REPORTED
Reason: Violation is not reported because of the
following "Related FLPs" status.
Related FLPs:
```

```
top.default_case_2: PROVEN
```

In the above example, all cases are defined for input sel; henceforth, the default case is not mandatory here. Therefore, the W551 violation for the above example is a noise violation. The violation is not reported because the FLP related to the violation is `PROVEN`.

ImproperRangeIndex-ML

For detailed information on ImproperRangeIndex-ML tag, see [ImproperRangeIndex-ML](#).

Consider the following example:

```
module top(
    input in1,in2,
        input [3:0] a,
        output reg [20:0] out
    );

    wire [4:0] b;

    assign w1 = in1 & in2;
    assign w2 = w1 & ~in2;

    assign b = {w2,a}; // Setting b[4] as 'w2' which will
                       // always be 0 for every possible input combination

    always@(*) begin
        out[b] = in1; // No Violation should be flag here as
                       // max value of 'b' can be 15 and out can index upto '20'
    end
endmodule
```

In the above example, the *ImproperRangeIndex-ML* rule reports a violation for the following assignment:

```
out[b] = in1;
```

Functional Lint Violation Summary output

Following is the functional Lint violation summary output for this rule:

```
Rule: ImproperRangeIndex-ML | file: top.v | line: 15 |  
MODULE: top | LINT_EXPRESSION: b | HDL_EXPR_SIZE_NEW: 5 |  
VALUE_1: 31 | MAX_VALUE: 20 | NODE_NAME: out | HIERARCHY::top  
STATUS: NOT_REPORTED
```

Reason: Violation is not reported because of the following "Related FLPs" status.

Related FLPs: top.improper_array_access_0: PROVEN

In above example, the value of `b[4]` is always 0 for every possible input combination. Therefore max value of `b` is 15 and `out` can index upto 20. Therefore, the *ImproperRangeIndex-ML* rule violation is a noise and the violation is not reported because the FLP related to the violation is `PROVEN`.

MissingFsmStateTransition

Reports violation for a missing state transitions, which are not possible in FSM

Description

The MissingFsmStateTransition rule reports a violation for state transitions, which are not possible in FSM.

To enable the tag (formal), provide the `-formal` switch with the rule.

```
configure_lint_tag -enable -tag "MissingFsmStateTransition"  
-goal <goal_name> -formal
```

Violation Messages

The FSM [FsmName] may have a missing transition pair from [FromState] to [ToState].

Examples

Consider the following example:

```
module top(clock ,reset, gnt_0, gnt_1);  
input clock,reset;  
output gnt_0,gnt_1;  
wire clock,reset;  
reg gnt_0,gnt_1;  
parameter SIZE = 3;  
parameter IDLE = 3'b001,GNT0 = 3'b010,GNT1 =  
3'b100,GNT2=3'b111;  
reg [SIZE-1:0] state;// Seq part of the FSM  
reg [SIZE-1:0] next_state;// combo part of FSM  
  
always @ (posedge clock or posedge reset) begin: FSM  
    if (reset == 1'b1) begin  
        state <= IDLE;
```

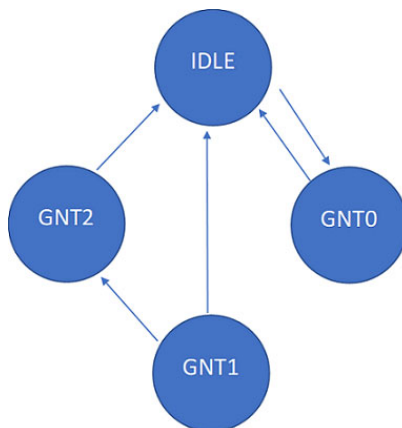
```

end else
begin
    case(state)
    IDLE: state <= GNT0;
    GNT0: state <= IDLE;
    GNT1: state <= GNT2;
    GNT2: state <= IDLE;
    endcase
end
end

endmodule

```

The following figure illustrates the State Transition diagram for the above example:



Here, the MissingFsmStateTransition rule reports a violation for GNT1 > GNT2 transition because state transition for GNT1 and GNT2 is not possible in FSM. This transition is not possible, because GNT1 state is not reachable in FSM. (NotReachableFsmState rule reports violation for GNT1, as it is not

reachable).

Possible State transition: IDLE->GNT0 / GNT0->IDLE / GNT2->IDLE /
GNT1->IDLE

NotReachableFsmState

Reports violation for not reachable state in FSM

Language

Verilog

Description

The *NotReachableFsmState* rule reports a violation for not reachable state in FSM.

To enable the tag (formal), provide the `-formal` switch with the rule.

```
configure_lint_tag -enable -tag "NotReachableFsmState" -goal  
<goal_name> -formal
```

Violation Messages

FSM state [FsmState] may not be reachable in the FSM [FsmName] .

Examples

Consider the following example:

```
module top(clock, reset, gnt_0, gnt_1);  
  input clock, reset;  
  output gnt_0, gnt_1;  
  wire clock, reset;  
  reg gnt_0, gnt_1;  
  parameter SIZE = 3;  
  parameter IDLE = 3'b001, GNT0 = 3'b010, GNT1 =  
    3'b100, GNT2=3'b111;  
  reg [SIZE-1:0] state;// Seq part of the FSM  
  reg [SIZE-1:0] next_state;// combo part of FSM  
  
  always @ (posedge clock or posedge reset)  
  begin: FSM
```

```
if (reset == 1'b1) begin
    state <= IDLE;
end else
    begin
        case(state)
            IDLE: state <= GNT0;
            GNT0: state <= IDLE;
            GNT1: state <= GNT2;
            GNT2: state <= IDLE;
        endcase
    end
end
endmodule
```

Here, the *NotReachableFsmState* rule reports a violation for state GNT1 and state GNT2, because GNT1, GNT2 state never happened.

Possible State transition: IDLE->GNT0 / GNT0->IDLE / IDLE->GNT0

NoExitFsmState

Reports violation if a state is reachable, and no other transition is possible in any of the clock cycle

Language

Verilog

Description

The NoExitFsmState rule reports a violation if a state is reachable, and no other transition is possible in any of the clock cycle.

To enable the tag (formal), provide the `-formal` switch with the rule.

```
configure_lint_tag -enable -tag "NoExitFsmState" -goal  
<goal_name> -formal
```

Violation Message

The FSM [FsmName] may remain stuck in the FSM state [FsmState].

Example Code and/or Schematic

Consider the following example:

```
module top (clock, reset, gnt_0, gnt_1);  
  input clock, reset;  
  output gnt_0, gnt_1;  
  wire clock, reset;  
  reg; gnt_0, gnt_1;  
  parameter SIZE = 3;  
  parameter IDLE = 3'b001, GNT0 = 3'b010, GNT1 =  
    3'b100, GNT2=3'b111;  
  reg [SIZE-1:0] state;// Seq part of the FSM  
  reg [SIZE-1:0] next_state;// combo part of FSM  
  
  always @ (posedge clock or posedge reset)
```

```
begin: FSM
    if (reset == 1'b1) begin
        state <= IDLE;
    end else
        begin
            case(state)
                IDLE: state <= GNT0;
                GNT1: state <= GNT2;
                GNT2: state <= IDLE;
            endcase
        end
    end
endmodule // End of Module arbiter
```

Possible State transition: IDLE->GNT0.

Here, the *NoExitFsmState* rule reports a violation for GNT0 to IDLE state change as the FSM is stuck in that state change.

RegisterStuckInResetState-ML

Reports violation for the registers that are stuck in a constant initial state due to bad coding style

Language

Verilog

Description

The *RegisterStuckInResetState-ML* rule reports a violation for the registers that are stuck in a constant initial state due to bad coding style.

The rule identifies the potential registers and pass only those registers to the formal engine that can correctly identify the stuck at constant situations. Only those registers that are a part of a loop are passed to formal engine to check accuracy. The register outputs that have a self-loop or are in the sequential-loop with any other register are the potential candidate for formal evaluation.

To enable the tag (formal), provide the `-formal` switch with the rule.

```
configure_lint_tag -enable -tag RegisterStuckInResetState-ML  
-goal <goal_name> -formal
```

Violation Messages

The FSM [*FsmName*] may remain stuck in the FSM state [*FsmState*].

Examples

Consider the following example:

```
module foo (clk, reset, in, out);  
  input clk;  
  input reset;  
  input in;  
  output reg out;  
  
  always @(posedge clk or posedge reset) begin  
    if (reset) out <= 0;
```

```
else out <= out & in; // q is output port
end
endmodule
```

The *RegisterStuckInResetState-ML* rule reports a violation for the register `out` as shown in the following assignments:

```
if (reset) out <= 0;
else out <= out & in; // q is output port
```

Here, in the above RTL, the initial value of the `out` register is defined as zero and the `out` register has a defined self loop. Therefore, the `out` register is stuck at zero and the *RegisterStuckInResetState-ML* rule reports a violation.

The violation related to the above FLP is `PROVEN` and the *RegisterStuckInResetState-ML* rule reports a violation

Violation Messages

The register `[RegName]` does not exit from its reset state `[RegValue]` .

DeadCode-ML

Reports redundant logic in the design.

When to Use

Use this tag to report redundant logic in the design.

Description

This tag reports logic redundancy in the design with respect to the current design setup.

To enable the rule (formal), provide the -formal switch with the rule.

Language

Verilog

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

Dead code exists as the block in file '[FileName]' at line(s) '[LineRange]' is not reachable.

Potential Issues

A violation is reported when the dead code exists because a block in a file is not reachable.

Consequences of Not Fixing

Not fixing the violation may lead to additional silicon area.

How to Debug and Fix

To fix the violation, ensure that the dead code does not exist.

Example Code and/or Schematic

Example 1

```
module top(in1 ,in2 , out);

input logic in1 , in2 ;

output logic out ;
logic [1:0] sel;
assign sel = 2'b11;

always_comb begin

if( sel == 2'b00 )

out = in1 & in2 ;

else

out = in1 | in2 ;

end
endmodule
```

In the above example, if condition is always false. Therefore, the DeadCode-ML reports a violation for the if statement.

Default Severity Label

Warning

Reports and Related Files

None

MissingCaseItems-ML

Reports violations for reachable case items that are not specified.

Language

Verilog

Description

The MissingCaseItems-ML rule reports violations for the reachable case items that are missing in a case statement where the full_case pragma or priority modifier is attached.

This rule does not report violations for the case statements that have a default branch. In addition, it does not report a violation if the missing (uncovered) case item can never be executed. Such items are known as unreachable case items.

Violation Message

Case statement over '[Signal]' is not a full-case

Default Severity Label

Warning

Examples

Example 1

In this example, the 2'b11 case item is not defined and it is a reachable case item. Therefore, the rule will report violations.

```
module top(clk, in1, in2, out);  
  input clk, in1, in2;  
  output reg out;  
  wire [0:1] bs;  
  assign bs = {in1,in2};  
  always @(posedge clk) begin
```

```
case (bs) // synopsys full_case
2'b00: out <= 0;
2'b01: out <= 0;
2'b10: out <= 1;
endcase
end
endmodule
```

Example 2

In this example, the 2'b10 case item is not defined, but it is an unreachable case item. Therefore, the rule will not report any violations.

```
module top(clk, in1, in2, out);
input clk, in1, in2;
output reg out;
wire [0:1] bs;
assign bs = {in1,in1};
always @(posedge clk) begin
case (bs) // synopsys full_case
2'b00: out <= 0;
2'b01: out <= 0;
2'b11: out <= 1;
endcase
end
endmodule
```