SpyGlass® latch
Rules Reference Guide

Version N-2017.12-SP2, June 2018

SYNOPSYS®

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Report an Error

The SpyGlass Technical Publications team welcomes your feedback and suggestions on this publication. Please provide specific feedback and, if possible, attach a snapshot. Send your feedback to *spyglass_support@synopsys.com*.

Contents

Preface

About This Book

The SpyGlass® latch Rules Reference Guide describes the SpyGlass rules that check HDL designs for latch design related guidelines.

Contents of This Book

The SpyGlass latch Rules Reference Guide consists of the following sections:

Chapter	Describes...
Using the Rules in the SpyGlass latch Product	Describes how to use the rules in the SpyGlass latch product
Rules in SpyGlass latch	Describes the rules in the SpyGlass latch 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 rule...

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 the Rules in the SpyGlass latch Product

The SpyGlass® latch product has the following usage feature:

- *SpyGlass latch Rule Parameters*

SpyGlass latch Rule Parameters

This section provides detailed information on the SpyGlass latch product rule parameters.

You can set these parameters in both Atrenta Console and Tcl by using the following syntax:

```
set_parameter <parameter_name> <parameter_value>
```

For more information on setting the parameters, refer to the *SpyGlass Tcl Interface User Guide* and *Atrenta Console User Guide*.

consider_sgdc_clock

Specifies whether the *LatchGatedClock* rule 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
Example	
Console/Tcl-based usage	`set_parameter consider_sgdc_clock yes`
Usage in goal/source files	`-consider_sgdc_clock=yes`

checkunusedlatch

Specifies whether the *LatchGatedClock* rule checks for latches whose output is not used.

By default, the value of the `checkunusedlatch` parameter is set to `no` and the *LatchGatedClock* rule does not consider the latches whose output is

not used.

Used by	*LatchGatedClock*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter checkunusedlatch yes`
Usage in goal/source files	`-checkunusedlatch=yes`

effort_level

Specifies the effort level that SpyGlass should make for checking the *LatchFeedback* rule. For some cases SpyGlass may take excessive time, if `effort_level` is set to a high value. If this parameter is set to a smaller value, then the rule 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.

Used by	*LatchFeedback*
Options	<positive integer value>
Default value	100
Example	
Console/Tcl-based usage	`set_parameter effort_level 50`
Usage in goal/source files	`-effort_level=50`

ignoreCellName

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 rule

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.

Used by	*LatchGatedClock*
Options	PERL regular expressions
Default value	""
Example	
Console/Tcl-based usage	`set_parameter ignoreCellName “CK.*”`
Usage in goal/source files	`-ignoreCellName=“CK.*”`

latch_clock

Specifies a list of primary clock port names for the *ClockEdges* and *W422L* rules.

Use the `latch_clock` rule parameter as follows:

```
set_parameter latch_clock <clk-name-list>
```

Used by	*ClockEdges*, *W422L*
Options	Name of the clock pin for latches in the design
Default value	Empty string
Example	
Console/Tcl-based usage	`set_parameter latch_clock clk1,clk2`
Usage in goal/source files	`-latch_clock=clk1,clk2`

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* rule 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
Example	
Console/Tcl-based usage	`set_parameter latch_chain_max_length 4`
Usage in goal/source files	`-latch_chain_max_length=4`

latch_verbose

(Optional) Specifies *ClockEdges*, *LatchReset*, *W392aL*, and *W392bL* rules to generate detailed messages.

Use the `latch_verbose` rule parameter as follows:

```
set_parameter latch_verbose on
```

or

```
set_parameter latch_verbose off
```

Used by	*ClockEdges*, *LatchReset*, *W392aL*, *W392bL*
Options	on, off, comma separated list of rules
Default value	off
Example	
Console/Tcl-based usage	`set_parameter latch_verbose on`
Usage in goal/source files	`-latch_verbose=on`

latch_clock_file

(Optional) Specifies the location where the detailed messages file that is generated by the *ClockEdges* rule when the *latch_verbose* rule 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` rule parameter to specify a different file.

Use the `latch_clock_file` rule parameter as follows:

```
set_parameter latch_clock_file <file-name>
```

Used by	*ClockEdges*
Options	Name of a file in which the detailed messages are saved
Default value	latch_clock_edges
Example	
Console/Tcl-based usage	`set_parameter latch_clock_file myFile`
Usage in goal/source files	`-latch_clock_file=myFile`

latch_reset

Specifies a list of primary reset port names for the *GatedReset*, *LatchReset*, *W392aL*, and *W392bL* rules.

Use the `latch_reset` rule parameter as follows:

```
set_parameter latch_reset <rst-name-list>
```

Used by	*GatedReset*, *LatchReset*, *W392aL*, *W392bL*
Options	Name of the reset pin for latches in the design
Default value	Empty string
Example	
Console/Tcl-based usage	`set_parameter latch_reset rst1,rst2`
Usage in goal/source files	`-latch_reset=rst1,rst2`

latch_reset_file

(Optional) Specifies the location where the detailed messages file that is generated by the *LatchReset* rule and the *W392aL* rule when the *latch_verbose* rule 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` rule parameter to specify a different file.

Use the `latch_reset_file` rule parameter as follows:

```
set_parameter latch_reset_file <file-name>
```

Used by	*LatchReset*, *W392aL*
Options	Name of a file in which the detailed messaged are saved
Default value	latch_reset
Example	
Console/Tcl-based usage	`set_parameter latch_reset_file rstFile`
Usage in goal/source files	`-latch_reset_file=rstFile`

overlappingLatchLoops

Enables detection of overlapping loops.

By default, the value of this parameter is set to `yes` and the *LatchFeedback* rule 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
Example	
Console/Tcl-based usage	`set_parameter overlappingLatchLoops no`
Usage in goal/source files	`-overlappingLatchLoops=no`

strict

Specifies whether the *LatchGatedClock* performs a strict check.

By default, the `strict` parameter is set to `no` and strict checking is not done.

You can set the value of the parameter to either `yes` or provide a comma or space separated list of rules.

Used by	*LatchGatedClock*
Options	yes, no, comma/space separated list of rules
Default value	no
Example	

Console/Tcl-based usage	`set_parameter strict yes`
Usage in goal/source files	`-strict=yes`

Rules in SpyGlass latch

Overview

The SpyGlass latch product has the following rules for checking latch descriptions:

Rule	Reports
ClockEdges	Design units where latches are triggered by different edges of a named primary port
GatedReset	Latch descriptions that use the specified primary port as latch reset after combinational transformation
LatchFeedback	Latch descriptions where there is a feedback path from the latch output pin to latch data pin or latch enable pin
LatchGatedClock	Latch descriptions that have internally generated or gated clock/enable signals
LatchReset	Design units where specified primary port is used to reset latches both synchronously and asynchronously
W122L	(Verilog only) `always` constructs modeling latches where a signal read in the construct, is missing from the sensitivity list

Rule	Reports
W336L	(Verilog only) Blocking assignments used to assign latch outputs
W392aL	Design units where both edges of a named reset signal are used to reset latches asynchronously
W392bL	Design units where both edges of a named reset signal are used to reset latches synchronously
W422L	(Verilog only) Multiple clocks used in event control list of latch descriptions
W428L	(Verilog only) Task calls in sequential always constructs
W429L	(Verilog only) Task calls in combinational always constructs
W442aL	(Verilog only) always constructs modeling latches where the first statement is not an if statement
W442bL	(Verilog only) Complex asynchronous reset conditions
W449L	(Verilog only) Expressions used as latch enable conditions
W450L	(Verilog only) Multi-bit expressions or nets used as latch enable conditions

ClockEdges

Reports design units that use both labels of the clock to drive latches

When to Use

Use this rule to report design units in which latches are triggered by different levels of clocks.

Description

The *ClockEdges* rule 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* rule 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

Tcl Attributes

- `is_latch_clock_driven_on_both_edges`: This Tcl attribute returns those clock nets whose different levels trigger the laches in the design.

  For example:

  ```
  sg_shell> set_pref dq_design_view_type flat
  sg_shell> set net_iter [get_nets * -filter
  "is_latch_clock_driven_on_both_edges == true"]
  ```

  For more details, refer to the `is_latch_clock_driven_on_both_edges` attribute in the *Base Attributes* section of the *SpyGlass Tcl Shell Interface User Guide*.

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.
Rule 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, rule 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`:

```
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;

    out1 <= q1;
    out2 <= q2;
```

```
end rtl;
```

Default Severity Label

Warning

Rule Group

latch

Reports and Related Files

None

GatedReset

Do not use gated or internally generated resets to reset the latches

Language

Verilog and VHDL

Description

The *GatedReset* rule flags latch descriptions that use the specified primary port as latch reset after combinational transformation.

The *GatedReset* rule 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.

Rule Arguments

The *GatedReset* rule requires a user-specified list of primary port names using the *latch_reset* rule parameter and checks for these ports only.

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

Rule 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 rule-checked with the primary port `rst` specified using the `latch_reset` rule parameter, SpyGlass 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`:

```
entity top is
  port (enable, rst, data, gatingSig : in bit;
        output : out bit);
end top;
```

```
architecture rtl of top is
  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 rule-checked with the primary port `rst` specified by using the `latch_reset` rule parameter, SpyGlass generates the following message:

```
Reset pin is gated for the latch with data top.output
```

LatchFeedback

Reports a latch in which a combinational feedback path exists from output pin to data or enable pin

When to Use

Use this rule 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 rule improves functionality, testability, and reliability of a design.

Description

The *LatchFeedback* rule 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 rule 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.

Rule Exceptions

The *LatchFeedback* rule 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 SpyGlass should make for checking the *LatchFeedback* rule. 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 rule-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. SpyGlass 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* rule description in the *SpyGlass OpenMore Rules 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`:

```
entity top is
  port (output : out bit;
        enable, rst, data, gatingSig : in bit);
end top;

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

Rule Group

latch

Reports and Related Files

None

LatchGatedClock

Reports latches driven by gated or internally generated clock

When to Use

Use this rule to report latches driven by gated or internally generated clock.

Description

The *LatchGatedClock* rule reports latches that contain internally generated or gated clock or enable signals. This rule supports block path analysis, that is, constant value propagation. Therefore, this rule does not traverse on the blocked path.

Turbo Mode Support

The Turbo mode support is available for this rule. For more information, see the *SpyGlass Lint Turbo Structural User Guide*.

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* rule 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* rule 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 rule 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.

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* rule 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* rule 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*"
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
```

```
assign out = q;
endmodule
```

VHDL Example

Consider the following example where latch `q` is being triggered by an internally generated clock `enable`.

```
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;
end rtl;
```

Default Severity Label

Warning

Rule 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 rule to report design units in which a primary port is used to reset latches both synchronously and asynchronously. This rule improves functionality (bug escape) and testability of a design.

Description

The *LatchReset* rule 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 rule supports block path analysis, that is, constant value propagation. Therefore, this rule does not traverse on the blocked path.

Prerequisites

The *LatchReset* rule requires a list of primary port names that can be specified by using the *latch_reset* parameter.

Languages

Verilog and VHDL

Parameters

- *latch_reset*: 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* rule 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* rule 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* rule ignores latch reset pin driven by a constant value that can get propagated through assignment or through this constraint.

Tcl Attributes

- `is_async_sync_reset:` The `is_async_sync_reset` Tcl attribute returns the reset nets that are used to reset latch, both synchronously and asynchronously, in the design.

  For example:

  ```
  sg_shell> set_pref dq_design_view_type flat
  sg_shell> set net_iter [get_nets * -filter
  "is_async_sync_reset == true"]
  ```

For more details, refer to the `is_async_sync_reset` attribute in the *Base Attributes* section of the *SpyGlass Tcl Shell Interface User Guide*.

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. Rule 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, rule 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

  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:

```
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 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

Rule Group

latch

Reports and Related Files

None

W122L

Variable read inside a latch always block should be inside sensitivity list

Language

Verilog

Description

The *W122L* rule 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`

Rule 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, SpyGlass generates the following message:

```
Variable 'data' read within always construct is missing from
the sensitivity list
```

W336L

Blocking assignment to latch output should be avoided

Language

Verilog

Description

The *W336L* rule 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.

Turbo Mode Support

The Turbo mode support is available for this rule. For more information, see the *SpyGlass Lint Turbo Structural User Guide*.

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`

Rule Severity

Warning

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;

  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* rule flags design units where both edges of a named reset signal are used to reset latches asynchronously.

Rule Arguments

The *W392aL* rule requires a user-specified list of primary port names using the *latch_reset* rule parameter and flags design units where both edges of one of these ports are used to reset latches asynchronously.

You can also generate detailed messages using the *latch_verbose* rule parameter. By default, these detailed messages are saved to a file named latch_reset in the current directory. Use the *latch_reset_file* rule parameter to specify a different file.

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
```

Rule 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 rule-checked with the primary port `rst` specified using the `latch_reset` rule parameter, SpyGlass 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:

```
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 rule-checked with the primary port `rst` specified using the `latch_reset` rule parameter, SpyGlass 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* rule flags design units where both edges of a named reset signal are used to reset latches synchronously.

Rule Arguments

The *W392bL* rule requires a user-specified list of primary port names using the *latch_reset* rule parameter and flags design units where both edges of one of these ports are used to reset latches synchronously.

You can also generate detailed messages using the *latch_verbose* rule parameter. By default, these detailed messages are saved to a file named latch_reset in the current directory. Use the *latch_reset_file* rule parameter to specify a different file.

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
```

Rule 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 rule-checked with the primary port `rst` specified using the `latch_reset` rule parameter, SpyGlass 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:

```
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 rule-checked with the primary port `rst` specified using the `latch_reset` rule parameter, SpyGlass generates the following message:

Both edges of synchronous reset signal rst are used to reset latches

W422L

Multiple clocks in the event control list of latch is not allowed

Language

Verilog

Description

The *W422L* rule flags multiple clocks used in event control list of latch descriptions. For constructs, `always @ {*}` and `always_latch`, event control list is auto inferred.

Rule Arguments

The *W422L* requires a user-specified list of clock names using the *latch_clock* rule parameter and reports latch descriptions where more than one of these clocks is used.

Message Details

Message 1

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`

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. Rule checking skipped`

Rule Severity

Warning, Info

Verilog Examples

In the following example, multiple clocks are used in the event control of the always construct modeling a latch:

```
module top (clk1, clk2, rst, data, out);
  input  clk1, clk2, rst, data;
  output out;

  reg q;

  always @(clk1 or clk2 or rst or data)
    begin
      if (rst)
        q <= 1'b0;
      else if (clk1 == 1 || clk2 == 0)
        q <= data;
    end

  assign out = q;
endmodule
```

When you run the above example with `clk1` and `clk2` set as latch clocks using the `latch_clock` rule parameter, SpyGlass generates the W422L rule message.

W428L

Task should not be called inside a sequential always block

Language

Verilog

Description

The *W428L* rule 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`

Rule 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
```

```
  task t1;
    $display(" A Task is being invoked");
  endtask
endmodule
```

For this example, SpyGlass generates the W428L rule message.

W429L

Task should not be called inside a combinational always block

Language

Verilog

Description

The *W429L* rule 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`

Rule 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, SpyGlass generates the W429L rule message.

W442aL

First statement in an asynchronous always block should be an `if` statement

Language

Verilog

Description

The *W442aL* rule 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`

Rule 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 rule message.

W442bL

Asynchronous reset condition can only use simple condition

Language

Verilog

Description

The *W442bL* rule 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
```

Rule Severity

Warning

Verilog Examples

Consider the following example that shows a latch description with complex reset condition:

```
module top (enable, rst1, rst2,  data, out);
  input enable, rst1, rst2, data;
  output out;

  reg q;

  always @(enable or rst1 or rst2 or data)
    begin
```

```
      if (rst1 == 1 && rst2 == 0)
        q <= 1'b0;
      else if (enable)
        q <= data;
    end

  assign out = q;
endmodule
```

For this example, SpyGlass generates the W442bL rule message.

W449L

Expression used as latch enable may not be synthesizable

Language

Verilog

Description

The *W449L* rule 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`

Rule 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
```

W450L

Reports multi-bit expression used as latch enable condition

When to Use

Use this rule to report multi-bit expressions or nets that are used as latch enable conditions.

Description

The *W450L* rule 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.

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

Rule Group

latch

Reports and Related Files

None

Appendix: SGDC Constraints

SpyGlass Design Constraints (SGDC) provides additional design information that is not apparent in an RTL.

In addition, you can restrict SpyGlass analysis to certain objects in a design by specifying these objects by using SGDC commands.

The following table lists the SGDC commands used by the SpyGlass latch product:

SpyGlass latch	
assume_path	*set_case_analysis*

List of Topics