SpyGlass® MBIST Tcl Flow 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® DFT MBIST TCL Flow Reference Guide describes the SpyGlass MBIST Tcl flow and the related commands.

Contents of This Book

The SpyGlass DFT MBIST TCL Flow Reference Guide consists of the following chapters:

Section	Description
Using SpyGlass MBIST Tcl Flow	Describes the language support, references, limitations, related parameters, and parameter default values for SpyGlass DFT MBIST Tcl Flow.
SpyGlass DFT MBIST Tcl Commands	This section describes the design-dependent and design-independent Tcl commands.

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 SpyGlass MBIST Tcl Flow

Overview

The SpyGlass DFT MBIST product automates the insertion of Memory Built-In Self-Test (MBIST) IP into a design at the RTL level. The product also works at the gate level. However, working at the RTL level has many advantages that are crucial in leading-edge SoC designs.

Following are the benefits of insertion at the RTL level:

- Allows timing optimization of complete RTL and MBIST area
- Allows early and faster validation, as simulations are run at RTL
- Allows to run DFT rules at RTL, including MBIST logic

This guide describes the SpyGlass Memory Built-In Self Test (MBIST) insertion flow for:

- Replacing the original memories in a design with the bisted memories. This includes inserting the repair solution, if a fuse-wrapper IP is provided.
- Replacing non-bisted blocks with bisted blocks or inserting blocks that are already bisted into an SoC.

The flow is a single-step process, which is invoked using SpyGlass Tcl Shell Interface. You can run the MBIST Insertion tool at both the gate and the

RTL level.

This section describes the following topics:

Language Support

SpyGlass DFT MBIST product supports the RTL constructs in the following languages:

- Verilog 95
- Verilog 2001
- VHDL 93
- VHDL-87
- System Verilog

However, SpyGlass DFT MBIST product does not support RTL constructs in VHDL 2000.

NOTE: *Before using the SpyGlass DFT MBIST product, you are expected to have basic knowledge of SpyGlass operations.*

References

Please refer to the following guides for information related to SpyGlass DFT MBIST Tcl Flow:

- *SpyGlass DFT MBIST Sub Methodology Guide*
- *SpyGlass Tcl Shell Interface User Guide*

Platform Support

SpyGlass DFT MBIST product works on the Linux-64 RHEL 4.0 operating system.

NOTE: *SpyGlass runs much faster on the Linux platform as compared to the Solaris platform.*

Reading a Design

You should use set_option v <file name> in the TCL file when the module description in the Verilog file is encapsulated within "`celldefine + `endcelldefine":

```
set_option v ../rtl/mem/example_lib.v
```

For more information on reading a design, see *Specify the library files in the source design* section in the *SpyGlass Explorer User Guide*.

Limitations

The SpyGlass DFT MBIST product has the following limitations:

- SpyGlass Analyzer and SpyGlass DFT MBIST do not support VHDL 2000. SpyGlass does not compile the VHDL 2000 code. Therefore, you must do the changes manually.
- Local variables or generate variables present in the generic map are not supported inside generate statements.
- You can not modify entities having the same name present in different libraries in VHDL.

Related Parameters

This section explains all the mandatory and optional parameters in the SpyGlass DFT MBIST Tcl flow. The default value for all parameters is given in section SpyGlass DFT MBIST Tcl Flow Parameters Default Values.

mbist_top_level_name

Specifies the name of the top-level design unit. You can specify the top-level module or entity name as an input to this parameter. The parameter performs case-sensitive checks for Verilog and case-insensitive checks for VHDL.

Syntax

```
set_parameter mbist_top_level_name <string>
```

Example

```
set_parameter mbist_top_level_name my_top
```

The above example specifies that the top design unit is `my_top`.

mbist_flow

Allows you to chose whether to run the flow at the gate or rtl level.

Ensure that the value of this parameter is set to gate, when running the SpyGlass DFT MBIST product on a netlist. If this parameter is not provided, run-time at the gate-level is very high.

Syntax

```
set_parameter mbist_flow <rtl|gate>
```

mbist_csv_separator

Governs the separator used in the `.csv` report generated by the tool. The default value of this parameter is `;` (semicolon).

Syntax

```
set_parameter mbist_csv_separator "<separator>"
```

Example

```
set_parameter mbist_csv_separator ","
```

reportdirectory

Specifies the location and the name of the directory where the SpyGlass generated reports are stored.

Syntax

```
set_parameter reportdirectory <string>
```

Example

```
set_parameter reportdirectory GENERATED_REPORTS
```

mbist_generate_single_netlist

Governs the generation of a single netlist (one single .v or .vhd file) at gate level. If the value of the *mbist_flow* parameter is `gate`,then, by default, the tool generates a single file that contains the definitions of the design modules as well as the definitions of the newly inserted blocks, BIST modules. A new file named `atrenta_generated_<design_top_Name>.v|.vhd` is generated. By default, this parameter is switched `on`. This means that the tool creates a single netlist. If this is not desired, it is recommended to turn this parameter `off`.

Syntax

```
set_parameter mbist_generate_single_netlist <on|off>
```

Example

```
set_parameter mbist_generate_single_netlist on
```

mbist_display_message

Enables you to set the type of messages to be displayed on stdout.

Syntax

```
set_parameter mbist_display_message <string>
```

Example

```
set_parameter mbist_display_message "INFO ERROR WARNING"
```

mbist_overload_severity

Enables you to overload severity of the mbist messages.

Syntax

```
set_parameter mbist_overload_severity <msg_code> <severity>
```

Example

```
set_parameter mbist_overload_severity MB_2001 ERROR
```

Parameter Default Values

The following table shows the default values of the parameters described in the Related Parameters sections:

Parameter Name	Default Values
mbist_top_level_name	None
mbist_flow	rtl
mbist_csv_separator	;
reportdirectory	current_working_directory
mbist_generate_single_netlist	on
mbist_display_message	INFO ERROR
mbist_overload_severity	None

SpyGlass DFT MBIST Tcl Commands

Overview

The SpyGlass DFT MBIST flow typically requires information from the following two sources:

- **MBIST Design Configuration**: Enables the designer to identify the memory instances in the design and the various MBIST components to be used. In this case, the designer provides the design dependent information using the *Design Dependent Commands*.
- **MBIST Architecture Definition**: It describes the relation between various SpyGlass DFT MBIST components. In this case, the vendor provides the design independent information using the *Design Independent Commands*.

Design Independent Commands

This section describes the design independent commands for the SpyGlass DFT MBIST Tcl Flow.

mb_define_class

Defines a class for the mbist component.

Syntax

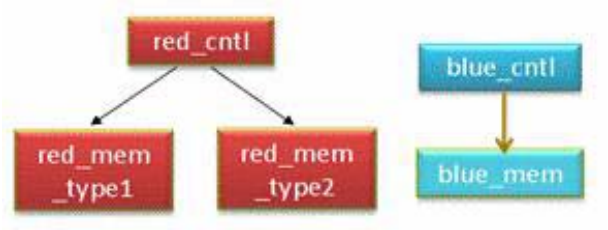
```
mb_define_class
    -class <class_name>
    [-children <child_class>]
    [-abstract]
    [-auto_configure]
```

Arguments

- **-class <class_name>**: Specifies the class name for mbist components.
- **[-children <child_class>]**: Specifies the list of child classes associated to the current class.
- **[-abstract]**: Specifies if the class is an abstraction class. This is useful for bottom-up/top-down flow.
- **[-auto_configure]**: Specifies whether to consider the abstract class auto configuration.

Examples

Consider the following figure:



The following commands describe the above figure:

```
sg_shell> mb_define_class -class red_mem_type1
# Define class red_mem_type1
sg_shell> mb_define_class -class red_mem_type2
#define class red_mem_type2

sg_shell> mb_define_class -class blue_mem
# Define class blue_mem

sg_shell> mb_define_class -class red_cntl
-children {red_mem_type 1 red_mem_type2}
# Define class red_cntl with children red_mem_type1 and
red_mem_type2

sg_shell> mb_define_class -class blue_cntl
-children {blue_mem}
# Define class blue_cntl with children blue_mem

sg_shell> mb_define_class -class lower_blk
-children {red_cntl blue_cntl} -abstract
# Define abstract class with children as red_cntl and
blue_cntl
```

See Also

mb_define_instance

mb_map_ip_to_class

mb_configure_connect_net

The `mb_configure_connect_net` command creates point to point connection between related mbist component class in the current design.

NOTE: *Use the mb_connect_net command to override the vendor connection defined using the mb_configure_connect_net command.*

Syntax

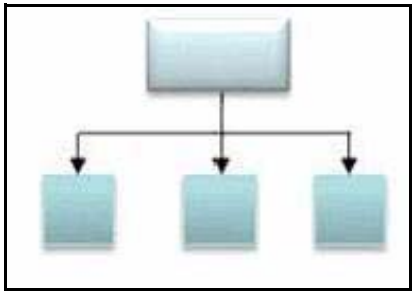
```
mb_configure_connect_net
    [-from_class <class_name>]
    [-from_pin <pin_name>]
    [-to_class <class_name>]
    [-to_pin <pin_name>]
    [-from_top_port <port_name>]
    [-to_top_port <port_name>]
    [-tie_extra_receivers]
    [-ignore_extra_receivers]
    [-ignore_extra_drivers]
    [-broadcast | -parallel | -bitwise_broadcast]
    [-through_class <class_name>]
    [-in_pin <pin_name>] [-out_pin <pin_name>]
    [-multiplex] [-select_enable <pin_name>]
    [-active <polarity>] [-max_children <count>]
    [-single_connection] [-id <id_string>]}
```

Arguments

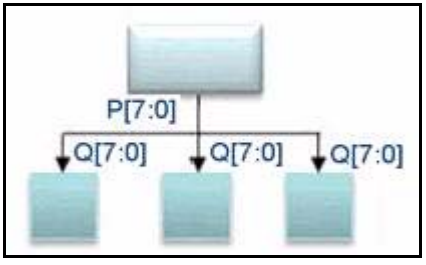
- **[-from_class <class_name>]**: Specifies the list of class names from where connection is established.
- **[-from_pin <pin_name>]**: Specifies the list of pin names on each instance of `-from_class` object.
- **[-to_class <class_name>]**: Specifies the list of class names to which connection is configured.
- **[-to_pin <pin_name>]**: Specifies the list of pin names on each instance of `-to_class` object.
- **[-from_top_port<port name>]**: Specifies input/inout top port name for defining generic connection with class.

- **[-to_top_port <port name>]**: Specifies output port name for defining generic connection with class.
- **[-tie_extra_receivers]**: Specifies 0/1 connection for spare `to_pin` pins.
- **[-ignore_extra_receivers]**: Specifies to ignore extra receivers when there is a mismatch in the number of receiver ports in the design that match this configuration.
- **[-ignore_extra_drivers]**: Specifies to ignore extra drivers when there is a mismatch in the number of driver ports in the design that match this configuration.
- **[-broadcast | -parallel | -bitwise_broadcast]**: If `-broadcast` option is specified, then connection is of broadcast type. If `-parallel` option is specified, then connection is of parallel type. If `-bitwise_broadcast` option is specified, then connection is of bitwise_broadcast type.

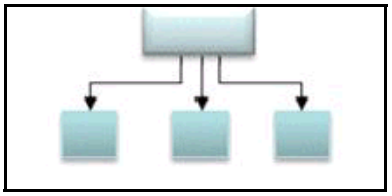
  The following figure illustrates the generated connection, if you specify the `-broadcast` option:



The following figure illustrates the generated connection, if you specify the `-bitwise_broadcast` option:

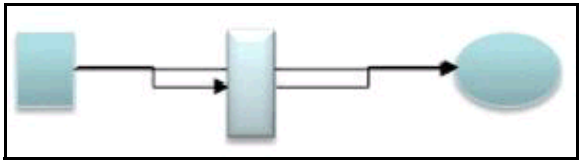


The following figure illustrates the generated connection, if you specify the `-parallel` option:



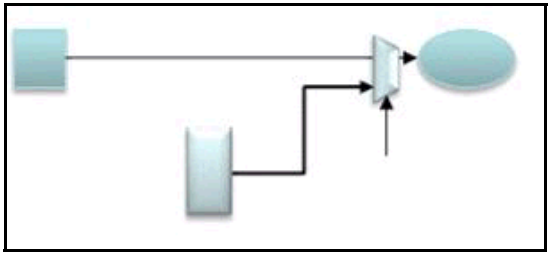
- **[-through_class <class_name>]**: Connect net through the list of class names specified in this option.

  The following figure illustrates the generated connection, if you specify the `-through_class` option:



- **[-in_pin <pin_name>]**: Input pin on each instance of `-through_class` object.
- **[-out_pin <pin_name>]**: Output pin on each instance of `-through_class` object.
- **[-multiplex]**: Specifies to insert a multiplexor with existing connection on the `-to` pin.

  The following figure illustrates the generated connection, if you specify the `-multiplex` option:



- **[-select_enable]**: Select-enable signal for multiplexing.

- **[-active] <HIGH | LOW | 1 | 0>**: Specifies polarity of select-enable signal.
- **[-max_children]**: Specifies maximum number of child instances to be connected.
- **[-single_connection]**: Tap the connection from single, that is, first child_instance only. It is useful for sharing common net connected to child instances.
- **[-id <ID>]**: Specifies configuration rule ID.

For more information on the connection types, refer to the *Specifying Connection* section in the *SpyGlass DFT MBIST Methodology guide.*

Examples

```
sg_shell> mb_configure_connect_net -from_class BIST
-from_pin A -to_class MEM0 -to_pin B -broadcast
# Creates a net connection between the BIST and MEM0 class.

sg_shell> mb_configure_connect_net -from_class MEM1
-from_pin {Q} -through_class BIST -in_pin {dgr_1 dgr_2 dgr_3}
-out_pin {dgrs_1 dgrs_2 dgrs_3} -parallel
# Creates a point to point connection.

sg_shell> mb_configure_connect_net -from_class BIST
-from_pin WEN -to_class MEM1 -to_pin WEN -multiplex
-select_enable SEN -active HIGH
# Creates a point-to-point connection with a multiplexer to
select the pre-bist and post-bist connection.

sg_shell> mb_configure_connect_net -from_class BIST
-from_pin WEN -to_top_port TOP_WEN
# Creates a connection from pin WEN for all instances of
class BIST to top port TOP_WEN

sg_shell> mb_configure_connect_net -from_top_port TOP_A
-to_class BIST -to_pin A
# Creates a connection from input port TOP_A to pin A for all
instances of class BIST
```

See Also

mb_configure_connect_chain

mb_configure_connect_gate

mb_configure_connect_gate

Creates logic tree connection between related mbist component class in the current design. Permissible combination gate types are AND, OR, and XOR only.

For more information on the command, refer to *Combining Signals through Gates* section in the *SpyGlass DFT MBIST Methodology Guide*.

NOTE: *Use the mb_connect_gate command to override the vendor connection defined using mb_configure_connect_gate command.*

Syntax

```
mb_configure_connect_gate
    -type <type>
    -from_class <from_class> -from_pin <from_pin>
    -to_class <to_class> -to_pin <to_pin>
    [-cell <cell>]
    [-use_as_buffer]
    [-max_children <max_children>]
    [-id <id>]
```

Arguments

- **-type <string>**: Specifies the gate type, that is, whether the gate is OR, AND, or XOR gate.
- **-from_class <string>**: Specifies the list of class names from where gate connection is established.
- **-from_pin <string>**: Specifies the list of pin names on each instance of `-from_class` object.
- **-to_class <string>**: Specifies the list of class names to which gate connection is configured.
- **-to_pin <string>**: Specifies the list of pin names on each instance of `-to_class` object.
- **[-cell <string>]**: Specifies cell to be used for logic.
- **[-use_as_buffer]**: Specifies to use the cell as a buffer, if necessary.
- **[-max_children <integer>]**: Specifies maximum number of child instances to be connected.

- **[-id <string>]**: Specifies configuration rule ID.

Examples

Example 1

Consider the following command:

```
sg_shell> mb_configure_connect_gate -type AND
       -from_class {MEM0 MEM1 MEM1 MEM1}
        -from_pin   { bbad bbad_1 bbad_2 bbad_3 }
         -to_class BIST -to_pin G_bbad
```

In the above example, the `bbad*` ports of the memories are connected through AND gate and bring the status signal to the bist level port, `G_bbad`.

Example 2

Consider the following command:

```
sg_shell> mb_configure_connect_gate -type AND -cell AND02_4X
       -from_class { MEM0 MEM1 MEM1 MEM1 }
       -from_pin   { bbad bbad_1 bbad_2 bbad_3 }
       -to_class BIST -to_pin G_bbad
```

In the above example, the `bbad*` ports of the memories are connected through AND gate and bring the status signal to the bist level port, `G_bbad`. Also, `AND02_4X` is specified as a technology cell.

See Also

mb_configure_connect_net

mb_configure_connect_chain

mb_configure_connect_chain

Creates serial chain connection between related mbist component class in the current design.

For more information on the command, refer to the *Specifying Chain Connections* section in the *SpyGlass DFT MBIST Methodology Guide*.

NOTE: *Use the mb_connect_chain command to override the vendor connection defined using the mb_configure_connect_chain command.*

Syntax

```
mb_configure_connect_chain
    -from_class <class_name>
    [-start_pin <pin_name>]
    [-end_pin <pin_name>]
    -through_class <class_name>  -in_pin <pin_name>
    -out_pin <pin_name> [-max_chidren <count>] [-id <id>]
```

Arguments

- **-from_class <class_name>**: Specifies the list of class names from where chain connection is established.
- **[-start_pin <pin_name>]**: Specifies the start pin of the chain.
- **[-end_pin <pin_name>**]: Specifies the end pin of the chain.
- **-through_class <class_name>**: Specifies the list of class names through which chain connection is established.
- **-in_pin <pin_name>**: Specifies the list of input pin present on each instance of `-through_class`.
- **-out_pin <pin_name>**: Specifies the list of input pin present on each instance of `-through_class`.
- **[-max_chidren <count>]**: Specifies maximum number of child instances to be connected.
- **[-id <id>]**: Specifies configuration rule ID.

Examples

```
sg_shell> mb_configure_connect_chain -from_class BIST
-start_pin serial_out_ctrl -end_pin serial_in_ctrl
-through_class MEM0 -in_pin serial_in -out_pin serial_out

sg_shell> mb_configure_connect_chain -from_class BIST
-start_pin MB_ERRINFO -end_pin MSI_SHIFT_IN  -through_class
MEM0 -in_pin DQ_SI -out_pin DQ_SO -through_class MEM1 -in_pin
DQ_SI -out_pin DQ_SO
```

See Also

mb_configure_connect_net

mb_configure_connect_gate

mb_map_ip_to_class

Creates a mapping between mbist component class defined using the *mb_define_class* command and equivalent design module.

Syntax

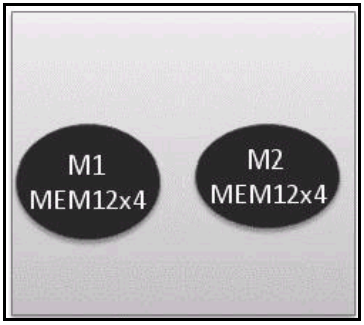
```
mb_map_ip_to_class
    -class <class_name>
    -ip <ip_name>
```

Arguments

- **-class <class_name>**: Specifies an mbist component class.
- **-ip**: Specifies the pattern for design module. For this argument, pattern matching is allowed. SpyGlass automatically picks the definition of `ip` from the path or you can also explicitly load file using the `read_file` command.

Examples

Consider the following figure:



Also, consider the following commands:

```
sg_shell> mb_define_class -class MEMORY
sg_shell> mb_map_ip_to_class -ip MEM12x4 -class MEMORY
```

In the above example, the ip, `MEM12x4`, is mapped to the defined class, `MEMORY`.

See Also

mb_define_class

mb_auto_configure_abstract_classes

Extends the connection configuration commands to defined abstract classes.

For more information on the command, refer to the *Abstract Class with auto_configure* section in the *SpyGlass DFT MBIST Methodology Guide*.

Syntax

```
mb_auto_configure_abstract_classes [-verbose] [-debug]
```

Arguments

- **[-verbose]**: Specifies to print out additional information.
- **[-debug]**: Runs in debug mode.

Examples

```
sg_shell> mb_auto_configure_abstract_classes -verbose -debug
```

See Also

mb_define_class

mb_configure_connect_net

mb_configure_connect_chain

mb_configure_connect_gate

Design Dependent Commands

This section describes the design dependent commands for the SpyGlass DFT MBIST Tcl Flow.

mb_reset

This command resets all mbist-related configuration data in the current design. Therefore, all the vendor and design related information is deleted and all the settings are restored to a default value.

The mb_reset command returns 0 after successful execution of the command.

Syntax

```
mb_reset
```

Examples

```
sg_shell> mb_reset
```

See Also

mb_insert

mb_set_prefix_n_suffix

Enables you to set prefix and suffix parameters to control naming schemes in modified design.

The prefixes and suffixes applied using the *mb_set_prefix_n_suffix* command are applied to all replaced and inserted instances, except the instance-specific prefixes and suffixes specified using the *mb_insert_instance* or *mb_replace_instance* commands.

Exceptions

- If the name of the module is different from the original file name, for example, module A is in file foo.v, then file name of foo.v is not changed. However, only suffix is added.
- If there are more than one modules in the .v file, then the name of the file is not changed. However, only suffix is added.
- If a global prefix or suffix is specified using the mb_set_prefix_n_suffix command, the instance-specific prefix and suffix for instance/module names would override the global one.

Syntax

```
mb_set_prefix_n_suffix
     [-module_prefix <module_prefix>]
     [-module_suffix <name>]
     [-instance_prefix <prefix>]
     [-instance_suffix <name>]
     [-file_prefix <file_prefix>]
     [-file_suffix <file_suffix>]
     [-wire_prefix <wire_prefix>]
     [-wire_naming_style <prefix_and_pin | prefix_only>]
```

Arguments

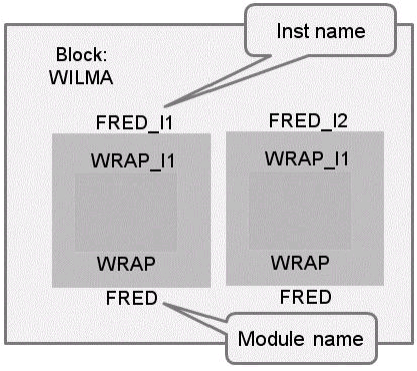
- **[-module_prefix <module_prefix>]**: To prepend in the name of modified or generated module. Default value is NULL. You can also specify the value of this argument as NO_PREFIX to not have any prefix on the modified module name.

- **[-module_suffix <name>]**: To append in the name of modified or generated module.
- **[-instance_prefix <prefix>]**: To prepend in the name of modified or inserted instance.
- **[-instance_suffix <name>]**: To append in the name of modified or inserted instance.
- **[-file_prefix <file_prefix>]**: To prepend in the name of modified or generated HDL file. Default value is `atrenta_modified_` for modified files and `atrenta_generated_` for the generated files. You can also specify the value of this argument as NO_PREFIX to not have any prefix on the modified filename.
- **[-file_suffix <file_suffix>]**: To append in the name of the modified or generated HDL file. Default value is `NULL`. You can also specify the value of this argument as NO_SUFFIX to not have any suffix on the modified filename.
- **[-wire_prefix <wire_prefix>]**: To prepend in the name of wire. Default value is `atrenta_wire`.
- **[-wire_naming_style <wire_naming_style>]**: To specify the naming style as either prefix_and_pin or prefix_only of newly inserted wires.

Examples

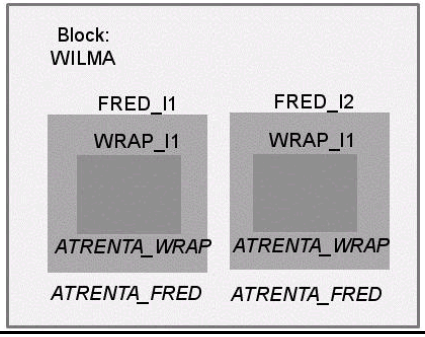
Example 1

The following illustrates a sample design:



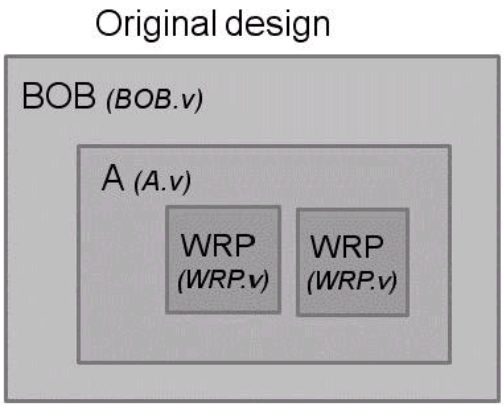
If you specify a prefix on the above design, using the `mb_set_prefix_n_suffix` command, the design is modified as shown in the following figure:

```
sg_shell> mb_set_prefix_n_suffix -module_prefix ATRENTA_
```



Example 2

Consider the following design example:



You can specify the mb_set_prefix_n_suffix command to set prefix and suffix parameters to control naming schemes in modified design as shown in the following figure:

```
mb_set_prefix_n_suffix \
                -module_prefix atrenta_BOB \
                -field_prefix Atrenta_BOB \
                -file_suffix sg
```

The design is modified as shown in the following figure, if you specify the mb_set_prefix_n_suffix command:

Modified module name	Modified file name
BOB	BOB.v_sg
atrenta_BOB_A	atrenta_BOB_A.v_sg
atrenta_BOB_WRAP	atrenta_BOB_WRP.v_sg

Example 3

Consider the following command:

```
sg_shell> mb_set_prefix_n_suffix -wire_prefix mbist_wire_
```

In the above example, specifying the `mb_set_prefix_n_suffix` command generates newly inserted wire names as `mbist_wire_0`, `mbist_wire_1`.. in the modified design.

Example 4

Consider the following command:

```
sg_shell>  mb_set_prefix_n_suffix -wire_prefix mbist_wire_
-wire_naming_style prefix_and_pin
```

In the above example, specifying the `mb_set_prefix_n_suffix` command generates newly inserted wire names, such as, `mbist_wire_0_<driver_pin>`, `mbist_wire_1_<driver_pin>`, and so on in the modified design.

See Also

mb_set_hierarchy_separator

mb_set_hierarchy_separator

Sets the hierarchy separator used for specifying hierarchical instances in the design.

Syntax

```
mb_set_hierarchy_separator <hierarchy_separator>
```

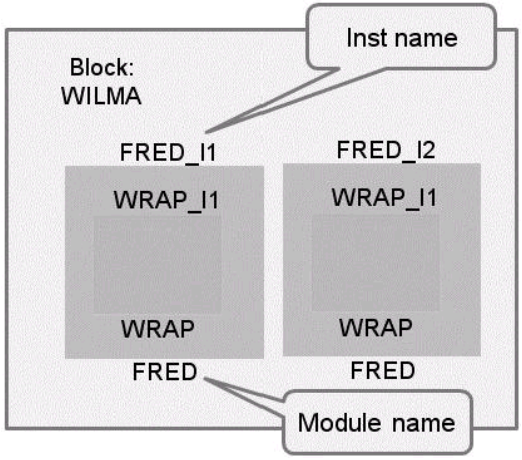
Arguments

<hierarchy_separator>: The default value is '.' (dot). Acceptable set of values are '.' (dot) and '/' (slash) only.

Examples

Example 1

Consider the following design example:



Specify the following command to set hierarchical separator is set as '/':

```
sg_shell> mb_set_hierarchy_separator /
```

Consider a hierarchical design with instance `FRED_I1` instantiated in top module and instance `WRAP_I1` inside FRED module. So, hierarchical instance `WRAP_I1` is specified as `top/FRED_I1/WRAP_I1`.

Specify the following command to set the hierarchical separator as '.':

```
sg_shell> mb_set_hierarchy_separator .
```

Hierarchical instance `WRAP_I1` is specified as `top.FRED_I1.WRAP_I1`.

See Also

mb_get_hierarchy_separator

mb_report_instances

Generates the soc_06.rpt file under the spyglass_reports, which contains all instances of the specified cell names or module patterns.

For more information on the `mb_report_instances` command, refer to the *Collecting Design Data and Preparing for MBIST Run* section in the *SpyGlass DFT MBIST Methodology Guide*.

Syntax

```
mb_report_instances
    -cell_name<cell_name>
    [-group_by_hierarchy]
    [-report_file]
    [-dive_in]
```

Arguments

- **-cell_name <cell>**: Specifies the list of patterns for the cell name. In this case, pattern matching is allowed.
- **[-group_by_hierarchy]**: Specifies to group the instances by hierarchy. In this case, depth first search order is applied.
- **[-report_file]**: Specifies path of the report file. The specified file is used as a soft link to the default file location.
- **[-dive_in>]**: Reports all matching instances. By default, tool does not dive inside the hierarchy of a matched cell.

Examples

```
sg_shell> mb_report_instances * -group_by_hierarchy
sg_shell> mb_report_instances *V*  *P*C* -dive_in
```

See Also

mb_define_instance

mb_insert_instance

mb_define_instance

Defines the instances in the current design and map them to the defined classes.

Syntax

```
mb_define_instance
    -top | -instance <instance>
    -class <class>
    [-children <children>]
    [-connected]
    [-add_port]
    [-view <view>]
    [-sub_class <sub_class>]
    [-skip_connection <skip_connection>]
    [-debug]
```

Arguments

- **-top**: Specifies to apply the definition to top module.
- **-instance <instance>**: Specifies the path of the hierarchical instance in the design. This instance should either exist in the design or you must insert the instance using mb_insert_instance command.
- **-class <class>**: Specifies the name of mbist component class. A mapping between the mbist class and specified instance is created.
- **[-children <children>]**: Specifies the hierarchical instance name of related child instances. The child instances must be previously defined using mb_define_instance command.
- **[-connected]**: Specifies whether an instance is already connected to its children and grand-children. This option is useful for incremental bist insertion. This option has a limited scope and usability in the current implementation.
- **[-add_port]**: Specifies to add ports to the module to make MBIST connections only applicable for abstract class.
- **[-view <view>]**: Specifies view name when defining for top module. Defines a name to the abstract boundary of your block. Helpful in making connections in bottom-up or top-down flows.

- **[-sub_class <sub_class>]**: Specifies list of sub_classes included under the abstract instance. This option is particularly useful for making connections in the top-down flow. You can specify mbist component classes below the abstract block, which is not yet ready for bist insertion.
- **[-skip_connection <skip_connection>]**: Specifies list of IDs of connection-configurations that are to be skipped for this instance and its children. The specified ID must be associated with at least one of the connection configurations.
- **[-debug]**: Signifies a debug flag.

Examples

Example 1

The following example shows the usage of the `mb_define_instance` command:

```
sg_shell> mb_define_instance -instance top.inst.w11_1
-class MEM0
# defines instance "top.inst.w11_1" of class "MEM0"

sg_shell> mb_define_instance -instance top.sub_block.w11_2
-class MEM0
# defines instance "top.sub_block.w11_2" of class "MEM0"

sg_shell> mb_define_instance -instance top.w12_1 -class MEM0
# defines instance "top.w12_1" of class "MEM0"

sg_shell> mb_define_instance -instance CONTROLLER_1
-class BIST -children {top.inst.w11_1 top.sub_block.w11_2
top.w12_1}
# defines the bist class instance, CONTROLLER_1 to control
MEM0 instances

sg_shell> mb_define_instance -top -class ABS_BIST
-children top.inst.w11_1 -view v1
mb_define_instance -top -class ABS_BIST -children
{top.sub_block.w11_2 top.w12_1} -view v2
```

```
#defines an instance of abstract class along with views v1
and v2.
```

Example 2

Consider the following command:

```
sg_shell> mb_define_instance -instance {top.\MEM0.inst }
-class MEM0
```

In the above command, instances are defined using brackets to represent the memories that are instantiated inside the generate statements. Also note the white space within the brace.

Example 3

Consider the following command:

```
sg_shell> mb_define_instance -instance top.H1
-class abs_CNTRL -view V1 -sub_class {{BIST 1} {MEM 1}}
```

The above command illustrates the usage of `sub_class` option for top-down flow. The abstract block, `top.H1`, has one instance of class MEM and BIST below it. You must specify a list of all mbist component class recursively below the current block.

See Also

mb_define_class

mb_insert_instance

mb_insert_instance

Inserts new instances of specified cell or module definition in the current design.

Syntax

```
mb_insert_instance
    -cell <cell>
    -instance <instance>
    [-parameter_map <parameter_map_list> |
    [-parameter_map_ordered <parameter_map_order_list>]
    [-instance_prefix <prefix>]
    [-module_prefix <prefix>]
    [-instance_suffix <name>]
    [-module_suffix <suffix>]
```

Arguments

- **-cell <cell>**: Specifies the name of a cell, module, or entity to be inserted in the design.
- **-instance <instance>**: Specifies the hierarchical path of the new instance in the design.
- **[-parameter_map <parameter_map_list>]**: Specifies the name-value pair for parameterized instantiation. New instantiation in the design is created with the specified parameters. For missing values, the default parameter value is used.
- **[-parameter_map_ordered <parameter [-instance_prefix <prefix>]**: Specifies the ordered list of parameter values for parameterized instantiation. This is useful for positional parameter mapping.
- **[-instance_prefix <prefix>]**: Specifies the prefix value to prepend in the name of inserted instance.
- **[-module_prefix <prefix>]**: Specifies the prefix value to prepend in the name of module of inserted instance.
- **[-instance_suffix <name>]**: Specifies the suffix value to append in the name of inserted instance.
- **[-module_suffix <suffix>]**: Specifies the suffix value to append in the name of module of inserted instance.

Examples

```
sg_shell> mb_insert_instance -instance CONTROLLER_1
-cell BIST -parameter_map {NUM_BAYS 23 MEM_ADDR 10}
# Inserts bist class instance, CONTROLLER_1, to control the
MEM0 instance.

sg_shell> mb_define_instance -instance I7 -class MEM0
sg_shell> mb_define_instance -instance I6 -class MEM1
# Defines the instances of memory class MEM0 and MEM1

sg_shell> mb_insert_instance -instance CONTROLLER_1
-cell BIST1
sg_shell> mb_define_instance -instance CONTROLLER_1
-class BIST1 -children {I7 I6}
# Inserts an instance of bist class, CONTROLLER_1, is
inserted to control instances, I7 and I6.

sh_shell> mb_insert_instance -instance CONTROLLER_1
-cell BIST1 -parameter_map {NUM_BAYS 23 MEM_ADDR 10}
# Uses the parameter_map option is used to specify the
name-value pair for parameterized instantiation.
```

See Also

mb_define_class

mb_define_instance

mb_remove_instance

Removes specified instances in the current design.

Syntax

```
mb_remove_instance
    [-cell <cell>]
    [-instance <instance>]
    [-all_hierarchies]
```

Arguments

- **[-cell <cell>]**: Specifies list of cell names. For this argument, pattern match is allowed.
- **[-instance <instance>]**: Specifies list of instance names. For this argument, pattern match is allowed.
- **[-all_hierarchies]**: Specifies to search the instance across all hierarchies.

Examples

```
sg_shell> mb_remove_instance -instance w* -all_hierarchies
# Removes all instances with the name starting with w from
the design.

sg_shell> mb_remove_instance -instance {{*MEM*}}
# Removes the instances having MEM in their name at top-
level hierarchy from the design.

sg_shell> mb_remove_instance -cell *
# Removes all cells from the design.

sg_shell> mb_remove_instance -cell {BIST}
# Removes the BIST cell from the design.
```

See Also

mb_define_instance

mb_insert_instance

mb_report_instances

mb_replace_instance

mb_replace_instance

Replaces an existing instance in the design with a block or a wrapper. The replacement is done in such a way that every pin of the replaced block is mapped to a pin of the replacing block.

NOTE: *If you have specified global prefix or suffix using the mb_set_prefix_n_suffix command, the instance-specific prefix and suffix for instance or module names would override the global ones.*

Syntax

```
mb_replace_instance
    -cell <cell>
    -instance <instance>
    [-pin_map <pin_map>]
    [-ignore_pin <ignore_pin>]
    [-instance_prefix <prefix>]
    [-module_prefix <prefix>]
    [-instance_suffix <name>]
    [-module_suffix <suffix>]
```

Arguments

- **-cell <cell>**: Specifies the name of replacing cell.
- **-instance <instance>**: Specifies the name of the hierarchical instance.
- **[-pin_map <pin_map>]**: Specifies the pin mapping between new and old pins.
- **[-ignore_pin <ignore_pin>]**: Specifies the pin to be ignored on the old block. This is useful, if no mapping exists for pin on the old block.
- **[-instance_prefix <prefix>]**: Specifies the prefix value to prepend in the name of replaced instance.
- **[-module_prefix <prefix>]**: Specifies the prefix value to prepend in the name of module of replaced instance.
- **[-instance_suffix <name>]**: Specifies the suffix value to append in the name of replaced instance.
- **[-module_suffix <suffix>]**: Specifies the suffix value to append in the name of module of replaced instance.

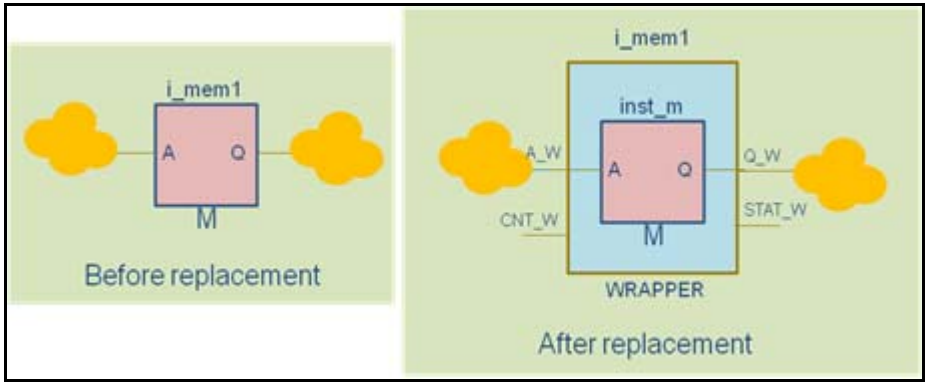
Examples

Consider the following commands to replace an existing instance in the design with a block or a wrapper:

```
sg_shell> mb_replace_instance -instance i_mem1 -cell WRAPPER
-pin_map {A_W A Q_W Q}
sg_shell> mb_define_instance -instance i_mem1 -class wrapper
# Replaces existing instance i_mem1 with instance of class
wrapper.
```

```
sg_shell> mb_replace_instance -instance i_mem1 -cell WRAPPER
pin_map {A_W A Q_W Q} -ignore_pin { Y }
# If there are extra pins on the memory that do not have a
corresponding pin map on the wrapper, then such pins can be
ignored during replacement.
```

Here, `i_mem1` is replaced with an instance of class wrapper during BIST insertion as shown in the following figure:



Please note that the replacing module should belong to a pre-defined class but there is no need to define a class for the replaced object.

See Also

mb_define_instance

mb_insert_instance

mb_connect_net

Connects nets in the current design.

For more information on the command, refer to *Establishing Custom Connections* section in the *SpyGlass DFT MBIST Methodology Guide*.

Syntax

```
mb_connect_net
    [-from <list_of_pin_names>]
    [-to <list_of_pin_names>]
    [-tie_extra_receivers <1|0>]
    [-broadcast | -parallel | -bitwise_broadcast]
    [-through_in_pin <pin_name>]
    [-through_out_pin <pin_name>]
    [-multiplex]
    [-select_enable <pin_name>]
    [-active <high|low|1|0>]
    [-cell <cell>]
    [-add_port]
    [-disconnect]
    [-id <id_string>]
```

Arguments

- **[-from <list_of_pin_names>]**: Specifies the list of pin names from where net is connected. It can either be a driver, such as, input port, output instance.pin or can provide a driver, such as, net connected to an input instance.pin.
- **[-to <list_of_pin_names>]**: Specifies the list of pin names to which net is connected. It should be a sink, such as, output port, input instance.pin.
- **[-tie_extra_receivers]**: Specifies 0/1 connection for spare `to_pin` pins.
- **[-broadcast | -parallel | -bitwise_broadcast]**: If `-broadcast` option is specified, then net connection is of broadcast type. If `-parallel` option is specified, then net connection is of parallel type. If `-bit-wise_broadcastoption` is specified, then net connection is of bitwise_broadcast type.

- **[-through_in_pin <pin_name>]**: Input pin on each instance of `-through` object.
- **[-through_out_pin <pin_name>]**: Output pin on each instance of `-through` object.
- **[-multiplex]**: Specifies to insert a multiplexor with existing connection on the `-to` pin.
- **[-select_enable]**: Selects enable signal for multiplexing.
- **[-active] <HIGH | LOW | 1 | 0>**: Specifies polarity of select-enable signal.
- **[-cell <cell>]**: Specifies cell for multiplexing (not supported).
- **[-add_port]**: Specifies add port when connecting to the top module port-boundary.
- **[-disconnect]**: Specifies to disconnect an existing connection.

NOTE: *Design connection to -to pin is broken, by default, and new connection as specified by the mb_connect_net command is made. Design connection from -from is preserved and new connection is made, in addition to the existing connection. To disconnect this existing connection also from -from, specify the -disconnect option.*

- **[-id <ID>]**: Specifies configuration rule ID.

Examples

Example 1

```
sg_shell> mb_connect_net -broadcast  \-from CONTROLLER_1.A \-
to w11_1.A

sg_shell>mb_connect_net -parallel \-from w21_1.Q \
-through_in_pin "CONTROLLER_1.dgr_1" "CONTROLLER_1.dgr_2" \
"CONTROLLER_1.dgr_3"\ -through_in_pin "CONTROLLER_1.dgrs_1 \
"CONTROLLER_1.dgrs_2" "CONTROLLER_1.dgrs_3"
```

Example 2

Consider the following vendor rule:

```
sg_shell> mb_configure_connect_net -from_class abs_TOP \
-from_pin 1 -to_class BST -to_pin BIST_MODE \
-broadcast -id CCN_BIST_MODE_TO_1
```

The above command connects the BIST_MODE pin of all the instances to 1. However, you can define the following custom connection in user data file to override the above default vendor connection:

```
sg_shell> mb_connect_net -from sys_logic_inst/TM_OUT \
    -to BIST_inst1/BIST_MODE -parallel mb_insert
```

The above command overrides the default connection of the BIST_inst1 instance with a custom connection from sys_logic_inst.

See Also

mb_configure_connect_net

mb_connect_gate

Enables you to configures a custom logic tree connection in the current design.

Syntax

```
mb_connect_gate
  -type <AND|OR|XOR>
  -from <list_of_pin_names>
  -to <pin_name>
  [-add_port] [-cell <cell>] [-use_as_buffer]
```

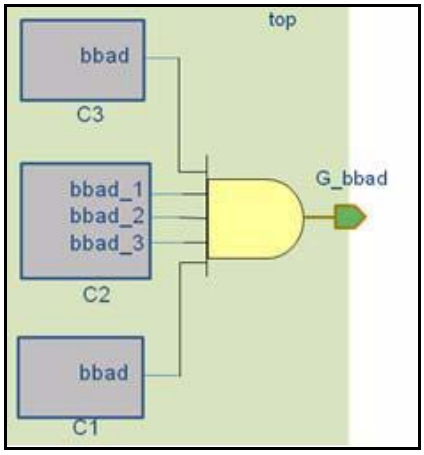
Arguments

- **-type <string>**: Specifies the gate type (OR | AND | XOR)
- **-from <list_of_pin_names>**: Specifies the list of hierarchical pin names, which serves as input to the AND, OR, or EXOR gates.
- **-to <pin_name>**: Specifies the hierarchical pin name, which is the output of AND, OR, or EXOR gates.
- **[-add_port]**: Specifies an additional port when you need to connect the output of the gate, which is connected to `-to` argument, to the top module port-boundary.
- **[-cell <cell>]**: Specifiies cell to be used for logic and indicates the name of the user-defined module to be instantiated. Cell should have only two scalar inputs and one scalar output.
- **[-use_as_buffer]**: Forces the tool to use the cell as buffer, if necessary, by typing the extra input to 0 (OR, EXOR) and 1 (AND).

Examples

Example 1

Following diagram represents three instances, C1, C2, and C3 , each having `bbad` output status signal:



Use the following command to bring the `AND` of these signals to top level port, G_bbad:

```
sg_shell> mb_connect_gate -type AND \
 -from [list C1.bbad C2.bbad_1 C2.bbad_2 C2.bbad_3 C3.bbad] \
 -to G_bbad
```

Use the `-cell <cell type>` option to specify the technology cell, if you want to use a specific library cell to combine signals.

Ensure that the description of the cell is supplied to SpyGlass DFT MBIST product as part of design read. The specified cell can only be of type: AND, OR, or XOR. That is, the gate should have two inputs only.

Specify the following modified command to indicate the connection to an AND cell, `AND02_4X`, from library:

```
sg_shell> mb_connect_gate -type AND -cell AND02_4X \
 -from [list C1.bbad C2.bbad_1 C2.bbad_2 C2.bbad_3 C3.bbad] \
 -to G_bbad
```

The above command results in insertion of a tree of the `AND02_4X` cells.

Example 2

Consider the following vendor rule for connection setup:

```
sg_shell> mb_configure_connect_gate -type XOR \
    -from_class { Class1 Class2 Class2 Class2 Class3} \
    -from_pin { bbad bbad_1 bbad_2 bbad_3 bbad } \
    -to_class top_class -to_pin G_bbad
```

By default, `XOR` of all the `bbad` status signals is brought to top level port, `G_bbad`. However, you can define the following custom connection in user data file to override the above vendor default connection:

```
sg_shell> mb_connect_gate -type AND \
 -from [list C1.bbad C2.bbad_1 C2.bbad_2 C2.bbad_3 C3.bbad] \
 -to G_bbad
```

See Also

mb_configure_connect_gate

mb_connect_chain

Enables you to establish custom serial chain connection in the current design.

Syntax

```
mb_connect_chain
    [-start <pin_name>]
    [-end <pin_name>]
    -instance <list_of_instances>
    -in <list_of_pin_names>
    -out <list_of_pin_names>
    [-add_port]
```

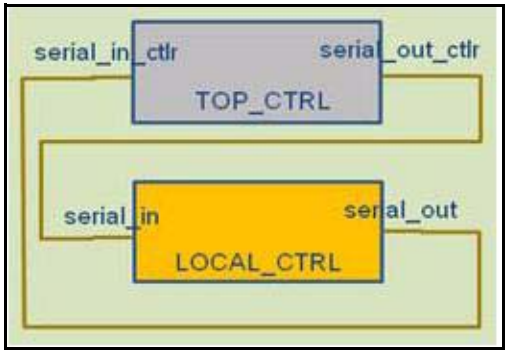
Arguments

- **[-start <pin_name>]**: Specifies the start pin of the chain.
- **[-end <pin_name>]**: Specifies the end pin of the chain.
- **-instance <list_of_instances>**: Specifies the instances, which form the chain connection.
- **-in <list_of_pin_names>**: Specifies the list of input pins present on each instance of `-instance`.
- **-out <list_of_pin_names>**: Specifies the list of output pins present for each instance of `-instance`.
- **[-add_port]**: Specifies an additional port when connecting to the top module port-boundary. This is an optional argument.

Examples

Example 1

Following diagram and the following command represents chain connection where `serial_out_ctlr` signifies start of the chain and `serial_in_ctlr` is the end point:




```
sg_shell> mb_connect_chain -start top_inst.serial_out_ctlr \
    -end top_inst.serial_in_ctlr \
    -instance [list local_inst] \
    -in serial_in -out serial_out
```

Example 2

Consider the following vendor rule for chain connection setup:

```
sg_shell> mb_configure_connect_chain -from_class TOP_CTRL\
     -start_pin serial_out_ctlr -end_pin serial_in_ctlr \
    -through_class BIST_CTRL -in_pin serial_in \
    -out_pin serial_out
```

This requires a chain connection through two class members, `TOP_CTRL` and `BIST_CTRL`.

However, you can define the following custom connection in user data file to override the above default vendor connection:

```
sg_shell> mb_connect_chain -start top_inst.serial_out_ctlr \
    -end top_inst.serial_in_ctlr \
    -instance [list local_inst] \
    -in serial_in -out serial_out
```

This requires a chain connection through two class members, TOP_CTRL and LOCAL_CTRL.

See Also

mb_configure_connect_chain

mb_insert

Initiates application of vendor commands to user-identified instances and generates the intermediate commands. Spyglass DFT MBIST is invoked for making necessary design changes. Modified HDL is produced as an output. After design modification, please, re-run SpyGlass with the following options specified to validate the modified design.

```
$SPYGLASS_HOME/bin/spyglass -designread -project
spyglass.prj
```

A project file (.prj) contains the following data about a particular Atrenta Console session:

- Input HDL files and language settings
- Run options
- State of project (design read, goal setup, goal run, or results analysis)
- Constraint files and parameter settings for goals
- Status of goal setup and analysis

Objective of running SpyGlass or any other tool is to check if the modified HDL is syntactically correct. You must run an equivalence checker between the pre-bisted design and the post-bisted design to ensure that the functionality of the design was not altered in the process of bist insertion. Finally, you must run a testbench and simulate the bist to verify that the status and control signals of the bist are correctly managed by SpyGlass.

Syntax

```
mb_insert
   [-top <top>]
   [-preview]
   [-input_control_file <input_control_file>]
   [-output_control_file <output_control_file>]
   [-force]
   [-report <report_name>]
   [-bottom_up | -top_down]
   [-quiet]
   [-debug]
```

Arguments

- **[-top <top>]**: Specifies the top design name.
- **[-preview]**: Runs the command in preview mode to check for any errors or warnings.
- **[-input_control_file <input_control_file>]**: Specifies the input control data file.
- **[-output_control_file <output_control_file>]**: Specifies the output control data file.
- **[-force]**: Forces mbist insertion despite errors.
- **[-report <report_name>]**: Specifies summary report for mbist configuration.
- **[-bottom_up]**: Specifies whether the mbist bottom-up flow is on. This argument is switched on, by default.
- **[-top_down]**: Specifies that the mbist top-down flow is on.
- **[-quiet]**: Specifies that the output is quiet output
- **[-debug]**: Specifies the debug option

It is recommended that you run the tool in preview mode to ensure that the expanded commands are properly generated. This enables you to review the errors and fix them before undergoing the final step of inserting the BIST logic.

Examples

```
sg_shell> mb_insert -top top -verbose

sg_shell> mb_insert -output_control_file $design.sgdc
-preview
# Sgdc file is specified through the output_control_file.

sg_shell> mb_insert -out $design.sgdc -top_down -force
# errors are ignored using the force insertion option, which
is not recommended.
```

See Also

mb_reset

mb_integrate

Integrates mbist inserted block specified by `-source_file` option and creates consolidated sources.f file at the top level block.

Syntax

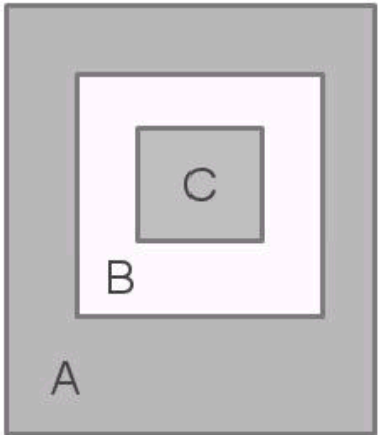
```
mb_integrate
    -source_file <source_file>
    [-debug]
```

Arguments

- **-source_file <source_file>**: Specify list of source files for each block.
- **[-debug]**: Runs the command in debug mode.

Examples

The following figure illustrates a top-down flow, `A->B->C`:



For the above flow, mbist insertion is first called for top block A, then for module B and module C, respectively. For all modified blocks, A, B and C, tool generates a sources file, `atrenta_generated_abstract_block.f`. The `sources.f` file directly or indirectly captures the relevant information, such as, list of files, list of views created, and list of ports added.

You can create the consolidated `sources.f` file that has all the design files after integration(A U B U C) using one of the following methods:

- **Method 1**

```
sg_shell> mb_integrate -source_file {PROJECT_A/mbist_dft/
spyglass_reports/rme/mbist-dft/atrenta_modified_rtl_files/
atrenta_generated_abstract_block.f PROJECT_B/mbist_dft/
spyglass_reports/rme/mbist-dft/atrenta_modified_rtl_files/
atrenta_generated_abstract_block.f  PROJECT_C/mbist_dft/
spyglass_reports/rme/mbist-dft/atrenta_modified_rtl_files/
atrenta_generated_abstract_block.f} -debug
```

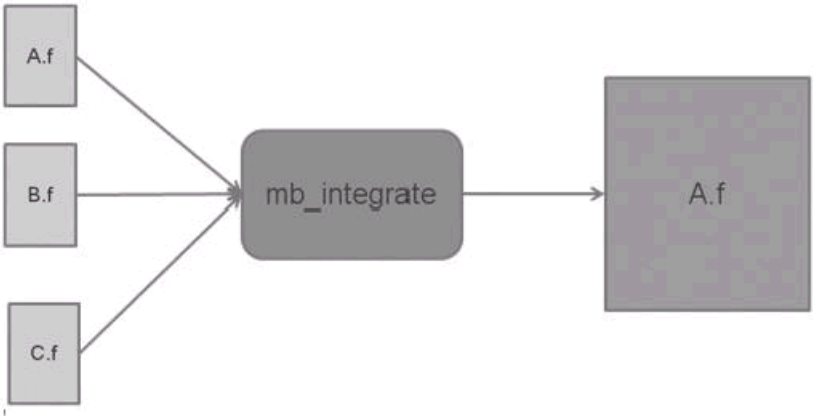
- **Method 2**

```
sg_shell> sg_shell> mb_integrate \
-source_file ./$env(OPTION)/PROJECT_A/mbist_dft/\
spyglass_reports/rme/mbistdft/atrenta_modified_rtl_files/\
atrenta_generated_abstract_block.f \
-source_file ./$env(OPTION)/PROJECT_B/mbist_dft/\
spyglass_reports/rme/mbistdft/atrenta_modified_rtl_files/\
atrenta_generated_abstract_block.f \
-source_file ./$env(OPTION)/PROJECT_C/mbist_dft/\
spyglass_reports/rme/mbist-dft/
atrenta_modified_rtl_files /\
atrenta_generated_abstract_block.f
```

- **Method 3**

```
sg_shell> mb_integrate \
-source_file [list \
./$env(OPTION)/PROJECT_A/mbist_dft/spyglass_reports/rme/\
mbist-dft/atrenta_modified_rtl_files/\
atrenta_generated_abstract_block.f \
./$env(OPTION)/PROJECT_B/mbist_dft/spyglass_reports/rme/\
mbist-dft/atrenta_modified_rtl_files/\
atrenta_generated_abstract_block.f \
./$env(OPTION)/PROJECT_C/mbist_dft/spyglass_reports/rme/\
mbist-dft/atrenta_modified_rtl_files/\
atrenta_generated_abstract_block.f]
```

Also, the following figure illustrates the above command declaration:



See Also

mb_insert

mb_reset_assertions

Resets all the assertion or verification commands specified before the execution of this command. No check is performed on the *mb_assert_function*, *mb_assert_path* and *mb_assert_value* commands, which are specified before the mb_reset_assertions command.

Syntax

```
mb_reset_assertions
```

Examples

```
sg_shell> mb_reset_assertions
```

See Also

mb_check_assertions

mb_assert_function

mb_assert_path

mb_assert_value

mb_assert_function

This command defines the logic function check between nodes in the design. It checks the sanity of created logic tree generated as a result of configure_connect_gate command, that is, whether all paths originating from `-from` field to `-to` field have the same structure as specified in `-function_type` field.

NOTE: *It is recommended to run the mb_check_assertions command to perform the actual checks.*

Syntax

```
mb_assert_function
    -from <list_of_start_nodes>
    -to <end_node>
    -function_type <AND|OR|XOR>
    [-check_type <structural|simulation>]
    [-cycle <integer>]
```

Arguments

- **-from <list_of_start_nodes>**: List of path origination node(s).
- **-to <end_node>**: Name of path termination node.
- **-function_type <AND|OR|XOR>**: Type of function (and, or, xor) to check for.
- **-check_type <structural |simulation>**: Type of check, whether structural or simulation. The simulation based check is default for AND/OR logic. Whereas, structure based check is default for XOR logic.
- **-cycle <integer>**: Specify number of simulation cycles, only for simulation based XOR gate check.

The `mb_assert_function` command uses the `Soc_07` rule to check the sanity of created logic tree. For more information on the `Soc_07` rule, refer to the *SpyGlass DFT Rules Reference Guide*.

Examples

Consider the following declaration for verifying that two source nodes, `in1`, amd `in2`, are connected to the destination node, `out1`, through a XOR tree:

```
sg_shell> mb_assert_function -from {in1 in2} -to out1 -
check_type structural -function_type XOR
```

At the end of the check, the following result appears on stdout:

```
RESULT: mb_assert checks PASSED

Look for Info messages from following rules in
prj_assert_fn_test/test/custom_dft_block_check/
spyglass_reports/moresimple.rpt:
      'Soc_07_Info'
```

A warning message is displayed in more_simple.rpt, if mb_assert checks are failed. For more information on the violation message, refer to the SoC_07 rule section in the *SpyGlass DFT Rules Reference Guide*.

See Also

mb_reset_assertions

mb_check_assertions

mb_assert_path

mb_assert_value

mb_check_node_existence

mb_assert_path

This command defines the path check between nodes in the design. It defines a connectivity check between two points in the design under a specific simulation condition and ensure that the path between user-specified nodes exist.

NOTE: *It is recommended to run the mb_check_assertions command to perform the actual checks.*

Syntax

```
mb_assert_path
    -from <list_of_start_nodes>
    -to <list_of_end_nodes>
    [-path_type <type>]
    [-invert | -no_invert]
    [-parallel]
    [-undirected]
    [-tag | -use_shift| -use_capture_at_speed |-use_capture]
    [-waveform]
```

Arguments

- **-from <list_of_start_nodes>:** List of path origination node(s)
- **-to <list_of_end_nodes>**:Name of path termination node(s).
- **[-path_type <type>]**: Type of path check. Accepts only the following predefined list of values:
  - ❒ buffered
  - ❒ sensitized
  - ❒ sensitizable

  The default value of this qualifier is sensitizable.
- **[-invert | no_invert]**: Checks for a positive polarity or negative polarity path between the two nodes.
- **[-parallel]**: Performs a one-to-one check between `from` and `to` nodes. By default a `one-to-many` type check is done.
- **[-undirected]**: Performs a bidirectional check.
- **[-tag <tag]**: Name of simulation condition.

- **[-use_shift]**: Simulates under shift mode constraint.
- **[-use_capture_at_speed]**: Simulates under capture_AtSpeed constraint.
- **[-use_capture]**: Simulates under capture constraint.
- **[-waveform]**: Enables waveform viewing for ease of debugging in GUI mode.

The `mb_assert_path` command uses the `Soc_02` rule to check the sanity of the created logic tree. For more information on the `Soc_02` rule, refer to the *SpyGlass DFT Rules Reference Guide*.

Examples

Consider the following declaration for verifying the connectivity of critical paths between start node and destination node:

```
sg_shell> mb_assert_path -from a\[2:0\] -to b\[2:0\] \
-path_type sensitizable -parallel -no_invert -waveform
```

An error message is displayed in more_simple.rpt, if the `mb_assert_path` command checks are failed. For more information on the violation messages, refer to the `SoC_02` Rule section in the *SpyGlass DFT Rules Reference Guide*.

See Also

mb_reset_assertions

mb_check_assertions

mb_assert_function

mb_assert_value

mb_check_node_existence

mb_assert_value

Defines value check at nodes in the design that requires a logic value to be established. This command is used to check specific value on user-specified design node under specific simulation condition.

NOTE: *It is recommended to run the mb_check_assertions command to perform the actual checks.*

The *mb_assert_value* command returns violation messages of severity, Info and Error, in the following cases:

- The following violation message of severity, Error, is reported when a node does not have the required value:

  ```
  mb_assert_value returns Node '<node_name>' has value
  <current_sim_value> (Required: <required_sim_value>)
  ```

- The following violation message of severity, Info, is reported when a node has the required value

  ```
  mb_assert_value returns "Node '<node_name>' has required
  value (<required_sim_value>)
  ```

For more information on these violation messages, refer to the *SoC_01* and *Soc_01_Info* rule section in the *SpyGlass DFT Rules Reference Guide*.

Syntax

```
mb_assert_value
    -name <node_list>
    -value <value_list>
    [-match_n_bits <size>]
    [-allow_inversion]
    [-use_shift | -use_capture_at_speed | -use_capture]
    [-waveform]
```

Arguments

- **-name <node_list>**: Assumes one or more of the following values:
  - ❒ Top-module port name
  - ❒ Internal net name
  - ❒ Terminal name

- **-value <value_list>**: Specifies a logic value string of 0, 1, X or Z. A single-bit value signifies a check at the end of complete simulation. If the value of the argument is X, it signifies a do not compare value.
- **[-match_n_bits <size>]**: This is an optional argument. It specifies the number of least significant bits to be considered. If <size> is greater than <value> (specified with -value argument), the latter is padded with X to match the former's width.
- **[-allow_inversion]**: This is an optional argument. It indicates that the inverted simulated pattern is also considered a valid pattern.
- **[-use_shift]**: This is an optional argument. Simulates under shift mode constraint.
- **[-use_capture]**: This is an optional argument. Simulates under capture constraint.
- **[-use_capture_at_speed]**: This is an optional argument. It simulates under the capture_AtSpeed constraint.

NOTE: *:If more than one of the -tag, -use_shift, -use_capture, or -use_capture_at_speed arguments is specified, an error condition occurs. You should specify exactly one of these modifiers with the mb_assert_value command.*

- **[-waveform]**: This is an optional argument. It generates waveform for better debugging in the GUI mode.

Examples

Consider the following command:

```
sg_shell> mb_assert_value -name top.a.b -value 1010 -
allow_inversion
-match_n_bits 2
```

In the above example, the mb_assert_value command stores the `top.a.b` node in memory for check when the *mb_check_assertions* command is issued.

See Also

mb_reset_assertions

mb_check_assertions

mb_assert_function

mb_assert_path

mb_check_node_existence

mb_check_assertions

Invokes SpyGlass DFT-MBIST to verify specified assertion checks in the design. You can specify the assertions using the mb_assert_value, *mb_assert_path*, and *mb_assert_function* commands. Also, you can view the result of individual assertions checks from the moresimple.rpt file generated in the spyglass_reports directory.

Syntax

```
mb_check_assertions
    [-top <design_name>]
    [-quiet]
    [-output_control_file <file_name>]
    [-project <prj_name>]
```

Arguments

- **[-top <design_name>]**: Specifies name of the design top.
- **[-quiet]**: If this option is specified, then command runs in quiet mode.
- **[-output_control_file <file_name>]**: Specifies the name of the output control data file.
- **[-project <prj_name>]**: Specifies the name of the current project.

Examples

```
sg_shell> mb_check_assertions -top top -quiet
```

See Also

mb_reset_assertions

mb_assert_function

mb_assert_path

mb_assert_value

mb_check_node_existence

Specifies whether to perform the existence check for nodes in the current design.

By default, fatal error is thrown, in case, design objects specified using `mb_assert_value`, `mb_assert_path`, or `mb_assert_function` commands are not found in the design.

Syntax

```
mb_check_node_existence {<on|off>}
```

Examples

```
sg_shell> mb_check_node_existence off
sg_shell> mb_assert_path -from I1.pin -to I2.clk
sg_shell> mb_check_node_existence on
```

mb_get_hierarchy_separator

Returns the hierarchy separator used to specify hierarchical instances in the current design.

Syntax

```
mb_get_hierarchy_separator
```

Examples

```
sg_shell> mb_get_hierarchy_separator
```

See Also

mb_set_hierarchy_separator

mb_set_attribute

Enables you to set attributes on existing classes or instances in the current design.

If you specify an attribute on a class, the attribute is not applied automatically on the respective instances of the class.

Also, you cannot use the mb_set_attribute command to set reserved attributes. The reserved attributes, that is, class_name, children and parent, are automatically set by tool when a class or an instance is defined.

Syntax

```
mb_set_attribute
  -class <object_type>
  <object_name>
  <attr_name>
  [<attr_value>]
```

Arguments

- **-class <object_type>**: Specifies the type of object. Allowed values are mb_class or mb_instance. You cannot specify a class and an instance simultaneously.
- **<object_name>**: Specifies the class or instance object name. Regular expression is allowed.
- **<attr_name>**: Specifies the attribute name. You cannot specify a reserved attribute name, such as, children, parent, and class_name.
- **[<attr_value>]**: Specify the value of the attribute to set on the class or instance. This can be a value or a name value pair.

Examples

```
sg_shell> mb_set_attribute -class mb_class COLLAR64 FORMAL
{{pin1 0} {pin2 1}}

sg_shell> mb_set_attribute -class mb_class COLLAR32 FORMAL1
{pin1 pin2}

sg_shell> mb_set_attribute -class mb_class COLLAR* LOW_POWER
```

```
# sets attribute LOW_POWER on all matching instances of
COLLAR*
```

See Also

mb_get_attribute

mb_get_attribute

Enables you to get attributes on existing classes or instances in the current design.

The mb_get_attribute command returns following values based on the attribute value specified:

- If attribute name is specified, the list of specified attribute values for the class or instance is returned. If no attribute value is specified, true is returned.
- If attribute name is not specified, list of all attributes, including available reserved attributes, for the specified class or instance are returned.

Syntax

```
mb_get_attribute
  -class <object_type>
  <object_name>
  [<attr_name>]
```

Arguments

- **-class <object_type>**: Specifies the type of object. Allowed values are mb_class or mb_instance.
- **<object_name>**: Specifies the class or instance object name. Regular expression is allowed but it must expand to a single class or an instance name.
- **[<attr_name>]**: Specifies the attribute name. This is an optional argument.

Examples

```
sg_shell> mb_set_attribute -class mb_class COLLAR64 \
FORMAL {{pin1 0} {pin2 1}}

sg_shell> mb_get_attribute -class mb_class COLLAR64 FORMAL
# returns a tcl list of form {{pin1 0} {pin2 1}}

sg_shell> mb_set_attribute -class mb_class COLLAR* LOW_POWER
```

```
sg_shell> mb_get_attribute -class mb_class COLLAR64 \
LOW_POWER
# returns true

sg_shell> mb_get_attribute -class mb_class {*} LOW_POWER
# returns error as multiple objects are found matching to
regular expression

sg_shell> mb_get_attribute -class mb_instance collar_inst
# returns list of all attributes on the specified instance

sg_shell> mb_get_attribute -class mb_class COLLAR64 children
```

See Also

mb_set_attribute

mb_get_objects

Enables you to get list of matching objects

Syntax

```
mb_get_objects
  -class <object_type>
  <object_name>
```

Arguments

- **-class <object_type>**: Specifies the type of object. Allowed values are mb_class or mb_instance.
- **<object_name>**: Specifies the class or instance name. You can also specify a list of regular expressions.

Examples

```
sg_shell> set instanceList [mb_get_objects \
                            -class mb_instance *]

sg_shell> foreach instance $instanceList { puts $instance}
#returns list of all instances present in the current design

sg_shell> mb_get_objects -class mb_class {COLLAR* BIST*}
#returns list of all matching classes
```

See Also

mb_define_class, *mb_define_instance*

mb_get_connected_net

Enables you to get connected net on hierarchical instance.pin or port. You can use this command post-insertion only and for new connections, which are created after bist insertion.

This command returns a Tcl list of connected points. In case of multi-fanout, all points are returned. In case specified input is not valid or connected net is not found, an error message is displayed.

Syntax

```
mb_get_connected_net <hierarchical instance.pin/ports>
```

Arguments

<hierarchical instance.pin/ports>: Specifies the hierarchical pin or port name for which connected net is to be found. The hierarchical instance should exist in the design or created through the *mb_insert_instance* command.

Examples

```
sg_shell> mb_get_connected_net "top.I1.inst2.insertout1"
# returns the hierarchical path of connected net, that is,
top.new_out_port

sg_shell> mb_get_connected_net "top.inst1.bbad"
# returns nothing because specified input pin is hanging
```

SEE ALSO

mb_connect_net, *mb_configure_connect_net*

mb_collect_modified_files

Creates links to modified files during bist insertion.

Syntax

```
mb_collect_modified_files
    <output_directory>
    [-project <project_name>]
```

Arguments

- **<output_directory>**: Specifies the name of the output directory.
- **[-project <project_name>]**: Specifies the name of the project. Default value is the name of the current project.

Examples

```
sg_shell> mb_collect_modified_files rtl_dir
# Creates a new directory, rtl_dir, containing links for all
modified files during the bist insertion.
```

mb_define_tag

Defines a named condition in the design to perform assertion checks.

Syntax

```
mb_define_tag
    -tag <tag>
    [-name <list_of_nodes>]
    [-value <list_of_value_at_nodes>]
```

Arguments

- **-tag <tag>**: Specifies the name of tag.
- **[-name <list_of_nodes>]**: Specifies list of nodes on which tag is applied.
- **[-value <list_of_value_at_nodes>]**: Specifies list of values to be applied on the node.

Examples

```
sg_shell> mb_define_tag -tag test_condition -name top.clk
-value {0 1 0 1}
```

mb_report_fv_constraints

Reports functional verification constraint already assigned on the object of the classes in the current design.

Syntax

```
mb_report_fv_constraints
    [-file <file_name>]
    [-value <value>]
```

Arguments

- **-file <file_name>**: Specifies file path in which constraints for formal verification are dumped.
- **-value <value>**: Specifies the type of constraint. The available options are:
  - all
  - dont_verify
  - 0
  - 1

Examples

```
sg_shell> mb_report_fv_constraints -file $(PROJECT)/
constraints_report.rpt

sg_shell> mb_report_fv_constraints -file $(PROJECT)/
constraints_report.rpt -value 1
```

See Also

mb_assign_fv_constraint

mb_assign_fv_constraint

Assigns the functional verification constraint on the object of the classes in the current design.

Syntax

```
mb_assign_fv_constraint
    -class <class>
    -pin <pin>
    -value <value>
    [-id <id>]
```

Arguments

- **-class <class_name>**: Specifies the name of the class on whose object constraint is to be applied.
- **-pin <pin>**: Specifies pin name of the object.
- **-value <value>**: Specifies type of constraint. The available options are:
  - dont_verify
  - 0
  - 1
- **[-id <ID>]**: Specifies the optional id string.

Examples

```
sg_shell> mb_assign_fv_constraint -class MEM0 -pin A -value 0
```

See Also

mb_report_fv_constraints

mb_skip_connection_configuration

Skips the identified configuration connections while mbist insertion in the current design. The connection configuration is skipped for all the instances defined using *mb_define_instance* command. In case of selective skip, use `-skip_connection` option of *mb_define_instance* command.

Syntax

```
mb_skip_connection_configuration -id <id>
```

Arguments

-id <id>: Specifies configuration rule id.

Examples

```
sg_shell> mb_skip_connection_configuration -id
```

See Also

mb_define_instance