insert_dft

NAME

insert_dft

Inserts DFT logic in the current design.

SYNTAX

```
status insert_dft
```

ARGUMENTS

The **insert_dft** command has no arguments.

DESCRIPTION

This command inserts DFT logic in the current design. The following types of DFT logic are supported:

- Standard scan chains
- Compressed scan chains with scan compression codec logic
- Clock-gating cell test pin connections
- AutoFix logic
- Test points
- On-chip clock (OCC) controller blocks
- Core wrappers
- ANSI/IEEE Std 1149.1/1149.6-compliant boundary-scan logic

By default, **insert_dft** performs standard scan chain insertion and routing. To insert compressed scan, boundary scan, or other types of DFT logic, use the applicable DFT configuration commands for that DFT logic type before running **insert_dft**.

The **insert_dft** command is supported with multicorner-multimode (MCMM) technology, which is available with Design Compiler Graphical.

Use the **set_dft_location** command to specify an alternate user-defined location for one or more types of DFT logic. For more information, see the man page.

Scan Synthesis Description

Scan cells and DFT logic cells are chosen from the technology library specified in the **target_library** variable. You must define a target library before issuing the **insert_dft** command.

The **insert_dft** command supports top-down, bottom-up, and middle-out scan insertion methodologies. In the process of adding DFT logic, testable versions of the subdesigns and top-level design are created. For testable subdesigns, new designs are created in the database that are distinct from the original subdesigns. The new subdesigns are named according to the value of the **insert_test_design_naming_style** variable. Note that the name of the current design is not changed during the **insert_dft** execution.

If existing test ports have been identified with the **set_dft_signal**, **insert_dft** will make connections to these ports.

If test ports have not been identified, new ports are added to the design as needed. The new ports are named according to the following variables:

```
test_mode_port_naming_style
test_mode_port_inverted_naming_style
test_clock_port_naming_style
test_scan_clock_a_port_naming_style
test_scan_clock_b_port_naming_style
test_scan_clock_port_naming_style
test_scan_enable_inverted_port_naming_style
test_scan_enable_port_naming_style
test_scan_in_port_naming_style
test_scan_out_port_naming_style
```

These variables are described in the **insert_dft_variables** man page and in the appropriate individual variable man pages.

The DFT insertion process performed by the **insert_dft** command works as described below.

First, existing logic is marked so scan insertion can detect whether a particular piece of logic has been added.

If test DRC results from a previous **dft_drc** command are not available, **insert_dft** invokes test DRC for the specified scan style. DRC generates the clock information that scan insertion retrieves. DRC also marks violating cells that cannot belong to scan chains.

Next, **insert_dft** architects scan chains into the design. The **insert_dft** command applies a scan-equivalence process to nonscan cells that passed DRC. Scan equivalents are initially selected based on library function identifiers. If no scan equivalent for a sequential cell is found using this behavior, **insert_dft** uses constraint-based sequential mapping techniques. These techniques consider the logic function implemented by each flip-flop instance to be scan replaced, along with immediately surrounding logic. This consideration generates a set of possible alternatives, from which the best replacement is selected based on **scan_style**, design rule considerations (for example, **connection_class** and **max_fanout**), and design area.

Use the **preview_dft** command to view the proposed architecture without inserting it. By default, **insert_dft** constructs as many scan chains as there are clocks and edges. By setting the **-clock_mixing** option of **set_scan_configuration**, the number of scan chains created based on clocks can be controlled. Set the **-clock_mixing** option to **no_mix**, **mix_edges**, or **mix_clocks**. If you want **insert_dft** to build more than the minimal scan chain count, set the **-chain_count** option of **set_scan_configuration** command. Multiple scan chains are also constructed to reflect the assignment of scan cells to scan chains by using the **set_scan_path** command.

When **insert_dft** constructs multiple scan chains, scan cells are allocated to scan chains based on the following criteria:

- The **set_scan_path** command
- Clock signals in the design
- Subdesign scan chains
- Design hierarchy

- Names of individual scan cells.

Scan cells are ordered on scan chains based on the following criteria:

- The **set_scan_path** command position
- Scan
- Clock trigger times
- Scan clock domains
- Scan cell names

The **insert_dft** command overrides the **set_scan_path** command positions to ensure functional chains if **set_scan_path** is not inserting lock-up latches.

Next, the **insert_dft** command makes any needed clock-gating cell test pin connections. These test pin connections place the clock-gating cells into a bypass mode when the test pin is asserted. For information on the available controls for these test pin connections, see the man pages for the **set_dft_clock_gating_configuration** and **set_dft_clock_gating_pin** commands.

insert_dft ensures that the internal three-state buses, which are on-chip buses not driven by pad cells, are disabled during scan shift. It also ensures that bidirectional ports are properly configured during scan shift. You can force the **insert_dft** command to disregard these steps by using the **-fix_bus disable** or **-fix_bidirectional disable** options of the **set_dft_configuration** command.

The **insert_dft** command avoids adding unnecessary logic by ignoring buses that already have exactly one active driver during scan shift, and bidirectional ports that are properly configured for scan shift. It takes into account any constant signals that have been applied with the **set_dft_signal -view existing_dft -type Constant** command.

Having identified candidate buses, **insert_dft** identifies the disabling logic required. The command iterates through all three-state bus drivers and computes the logic values that must be applied to input pins to disable the bus. It does not succeed in the following situations, when:

- A bus driver belongs to a design that has a **dont_touch** attribute.
- Any of the three-state drivers cannot be disabled.
- Conflicting logic values are required to disable three-state drivers.
- Disabling values conflict with those of previous buses.

The **insert_dft** command then adds generic disabling logic where necessary. It finds and gates all pins that do not hold the values they require during scan shift.

The command processes all bidirectional ports. It discards the following:

- Ports used to apply scan enable signals to the design
- Degenerated bidirectional ports that are configured as inputs or outputs by constant logic values
- Ports that do not have exactly 1 three-state driver that is not in a **dont_touch** design

The **insert_dft** command establishes the direction to which bidirectional ports must be configured during scan shift. Scan outputs are turned outwards, and other scan signals are turned inwards. By default, all other bidirectional ports are turned inwards. To turn bidirectional ports outwards, use the **set_autofix_element** command with the **-type external_bus** and **-method output** options.

The **insert_dft** command computes the logic values that must be applied to the driver input pins to configure each port. It fails if required logic values conflict with previous logic values. The command then adds generic configuration logic where necessary. It finds all pins that do not hold the required values during scan shift, and inserts appropriate test mode logic.

Having disabled three-state buses and configured bidirectional ports, **insert_dft** builds the scan chains if

the **set_scan_configuration** command has not been previously specified with the **-route false** option. For each chain, **insert_dft** determines the scan-in pin. It first looks for **ScanDataIn** signals specified by using the **set_dft_signal** commands. It then looks for design ports with ScanDataIn **signal_type** attributes. If it finds no such ports, it creates a dedicated scan-in port for the chain.

Having identified the scan-in pin, **insert_dft** jumps over the connected pad if it is not in a **dont_touch** design. It first tries finding a true or inverting path through the pad that is sensitized by test hold, scan enable, and bidirectional control signals, and does not need any additional disabling. If not successful, it gets the vector required to enable a path through the pad and the resulting hookup pin. It finds all pins that do not hold the necessary values during scan shift and inserts appropriate test mode logic. Note that **insert_dft** does not have to add three-state disabling logic to ensure that the pad is in input mode because this was performed during a previous step.

The **insert_dft** command then jumps through input buffers. Starting at the load pin, it jumps over single-fanout inverters and buffers to find a better hookup pin. The scan sense is toggled as this process jumps over inverters.

Given the best hookup pin, scan insertion builds the rest of the chain. The preferred routing order, specified by using the **set_scan_path** command, is used to route between serial signals and scan cells. If no preferred order is specified, it uses a default ordering scheme.

If the **set_scan_configuration** command has not been previously executed with the **-add_lockup false** option, the **insert_dft** command inserts lock-up latches at clock-domain boundaries. This compensates for clock skew and ensures a glitch-free scan shift. You can specify to **insert_dft** to add lock-up latches at the end of chain by using the **set_scan_configuration** command with **-insert_terminal_lockup**.

The **insert_dft** command can reuse functionally-connected **ScanDataIn** ports or pins for scan chain routing. It does not recognize a pin as functionally connected if the following conditions are true:

- The pin is not connected to a net.
- The pin has been created by **insert_dft**.
- The pin is a driver and does not have a load, or the pin is connected to a subdesign output port that **insert_dft** created.
- The pin is a load and does not have a driver, or the pin is connected to a subdesign input port that **insert_dft** created.

The **insert_dft** command does not touch optional methodology signal pins that are functionally connected. It reconnects mandatory methodology signal pins that are functionally connected. A warning message appears in this case.

The **insert_dft** command selects the scan-out driver for each cell by using the following criteria:

- It identifies the scan-out signal by looking at **signal_type** attributes. If pins with both ScanDataIn and ScanDataIn with inverted attributes are present, it chooses the noninverted pin, unless the other has a routing position.
- It considers all of the equal and opposite output pins (such as Q and QN) on the final scan cell in the chain, and chooses the output pin with the most slack. If equal, it chooses the driver with greatest drive strength.

You can set the **test_disable_find_best_scan_out** environment variable to **true** to disable this behavior.

To finish construction, **insert_dft** locates the scan-out port for the chain. If a scan-out port has not been previously specified with the **set_dft_signal** command, **insert_dft** creates a new one. If an existing scan-out port is found, **insert_dft** analyzes it to avoid inserting redundant multiplexing logic. It first establishes whether there is already a signal path between the last scan-out pin and the scan-out port when scan enable, test hold, and bidirectional control conditions are asserted. If the scan out port is connected to a three-state driver or a pad, it uses the reverse of the method described earlier for input ports to jump back through three-state drivers, pads, buffers, or inverters to a hookup pin. If the this pin is already functionally connected, **insert_dft** adds multiplexing logic and connects the selected output pin of the chain's last scan cell to the multiplexer input. If the scan-out net connects to some other existing port,

insert_dft isolates the scan-out port connection by making the connection with a buffer. The reason for this port isolation is because some downstream tools cannot handle nets that fanout to multiple ports.

Finally, **insert_dft** routes global signals (including any scan enable and test clock signals) and performs a default technology mapping for all new generic logic (including disabling logic and multiplexers). It introduces dedicated test clock signals for the **clocked_scan**, **lssd**, and **scan_enabled_lssd** scan styles.

When **set_dft_configuration -scan disable -clock_gating enable** has been previously specified, **insert_dft** performs only clock-gating cell routing. In this case, the **create_test_protocol** command should be run after **insert_dft** to update the CTL model with the correct ScanEnable or TestMode signal assertions.

Typically, at this point, the **insert_dft** command has violated compile design rules and constraints. If the **set_dft_insertion_configuration -synthesis_optimization none** command has been previously issued, or if Design Compiler topographical mode is used, optimization is skipped and the initial mapping is retained. Otherwise, **insert_dft** optimizes the design according to the setting of the **-map_effort** option of the **set_dft_insertion_configuration** command. For more information, see the **set_dft_insertion_configuration** man page. Note that **insert_dft** does not optimize the ScanEnable net to meet timing constraints.

In Design Compiler topographical mode, you should follow DFT insertion with an incremental topographical compile.

At this point, some cleanup is necessary. The **insert_dft** command resolves connection class violations, replaces and merges logic constants, then uniquifies the design. This is because you might want to write out subdesigns that **insert_dft** has modified as instances of new designs. In some cases, where test modifies instances of the same design differently, **insert_dft** produces more than one new design. It uses a technique called selective uniquification to produce new designs only when necessary. This technique generates a canonical netlist ID that does not depend on net-names or cell-names. The canonical netlist ID works only on gate-type, net connections, and port-names.

In the case of a non-uniquified hierarchical design, forced uniquification is performed to record the changes to the design. This is because you might want to write out subdesigns that **insert_dft** has modified as instances of new designs. In some cases, where test modifies instances of the same design differently, **insert_dft** produces more than one new design. You might want **insert_dft** to do this only when necessary.

The **insert_dft** command only considers **dont_touch** attributes for basic initial scan replacement of nonscan cells, such as those that would occur if a test-ready compile was not used. They are ignored for scan stitching, test signal routing, logic insertion, and identified shift register splitting (which scan-replaces the new head flip-flop). To control the scope of DFT insertion, use test-specific directives such as **set_scan_element false** or **set_scan_configuration -exclude_elements**.

BSD Synthesis Description

The **insert_dft** command synthesizes ANSI/IEEE Std 1149.1/1149.6-compliantboundary-scan logic using DesignWare macro cells when you specify the **-bsd enable** option of the **set_dft_configuration** command.

Synthesis is done at the gate level. The newly-synthesized BSD logic is mapped to the target library. The BSR cells added during BSD insertion are not uniquified. You must run the **uniquify** command after BSD insertion if you need a uniquified design. The **insert_dft** command can use a design with or without core logic, but pad cells must be present on the top-level design ports.

All JTAG test access ports (TAPs) must be specified. If you omit the JTRST signal using the **set_bsd_configuration -asynchronous_reset false** command, you must specify the remaining four mandatory ports. otherwise, you must specify all five ports.

Depending on the instructions you selected to implement with the **set_bsd_instruction** command, by default the instruction register (IR) is synthesized with the minimum number of bits to accommodate the set of instructions. However, you can enforce one-hot encoding of the instruction register by using **set_bsd_configuration -instruction_encoding one_hot**. The **insert_dft** command synthesizes BSR cells using DesignWare BC_1 on all the design output ports except on bidirectional ports, where a BC_7 is

synthesized. On all input ports and for all control points, BC_2 BSR cells are synthesized. If you set system clocks, a BC_4 is synthesized on these design ports. If you used some design ports other than the actual controlling ports to control the pads' controlling logic (for example, an extra controlling port to disable all three-state outputs together), logic is synthesized to allow the actual controlling signal to do the controlling during the boundary-scan testing. This removes the need for subsequently setting the extra controlling ports as compliance-enable ports.

It is not possible to synthesize boundary scan logic without pads. If the design does not have pads, the command terminates.

Error messages can result from missing TAP ports, missing pad cells or missing pad cell functionality in the library. A message is printed indicating an inability to proceed, and the command exits.

By default, synthesis of boundary scan logic is done in compliance with the IEEE1149.1-2001 standard. You can switch back to the IEEE1149.1-1993 standard by setting the **-std {ieee1149.1_1993}** option of the **set_bsd_configuration** command. To implement the IEEE1149.6-2003 standard, specify the **-std {ieee1149.6_2003}** option of the **set_bsd_configuration** command.

By default, synthesis of boundary scan logic is done using the new DesignWare tap, DW_tap_uc. To use the old DesignWare tap, DW_tap, instead, set the **bsd_use_old_tap** variable to true.

To use user-defined BSR/TAP designs in BSD synthesis, use the **define_dft_design** command to specify the user-defined designs.

For more details on the IEEE Std 1149.1/1149.6 rules, see the *IEEE Std Test Access Port and Boundary-Scan Architecture*.

EXAMPLES

The following example shows the usage syntax for the **insert_dft** command on the design **WC66** without fixing constraint violations and compile design rules:

```
prompt> current_design WC66

prompt> insert_dft
```

The following command performs synthesis of boundary-scan logic in compliance with ANSI/IEEE Std 1149.1 on the current design, and illustrates the types of messages that are output:

```
prompt> current_design M1
prompt> set_dft_configuration -scan disable -bsd enable
prompt> set_dft_signal -type  tdi  -port my_tdi
prompt> set_dft_signal -type  tck  -port my_tck
prompt> set_dft_signal -type  tms  -port my_tms
prompt> set_dft_signal -type  trst -port my_trst
prompt> set_dft_signal -type  tdo  -port my_tdo
prompt> create_clock -p 10 my_system_clock
prompt> insert_dft

  Loading design 'M1'
  ...
  Generating the TAP
  ...
  Generating the BSR cells
  ...
  Generating the control logic
  Writing implemented instructions information to the design
  Creating Hierarchy for Boundary Scan Logic ...
  ...
1
```

SEE ALSO

create_test_protocol(2)
current_design(2)
dft_drc(2)
preview_dft(2)

```
set_dft_configuration(2)
set_dft_insertion_configuration(2)
set_dft_location(2)
set_scan_configuration(2)
```