

Fusion Compiler™ Design-for-Test User Guide

Version T-2022.03-SP4, September 2022



Copyright and Proprietary Information Notice

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

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <https://www.synopsys.com/company/legal/trademarks-brands.html>. All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

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

Third-Party Links

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

www.synopsys.com

Contents

New in This Release	10
Related Products, Publications, and Trademarks	10
Conventions	11
Customer Support	11

1. DFT Insertion in the Fusion Compiler Tool	13
The DFT Post-Compile Flow	14
Setting DFT Optimization Goals	15

2. Basic DFT Configuration	17
Defining DFT Signals	17
Defining Scan Signals	17
Defining Test Clocks	18
Specifying a Hookup Pin for DFT-Inserted Clock Connections	19
Specifying a Location for DFT Logic Insertion	20

3. Scan Architecture Concepts	21
Requirements for Valid Scan Chain Ordering	21
Lock-Up Latch Insertion Between Clock Domains	22
Automatically Creating Skew Subdomains Within Clock Domains	26
Manually Creating Skew Subdomains at Associated Internal Pins	29
Manually Creating Skew Subdomains With Scan Skew Groups	32
Building Scan Chains From Base Chains	34
Retiming Registers	34
Multivoltage Support	36
Scan and Stub Chains	36

4. DFTMAX Compression	38
The DFTMAX Compression Architecture	38

The DFTMAX Codec	38
Decompressor Operation	40
Compressor Operation	40
Compressor X-Blocking	40
Codec Implementation	41
Configuring DFTMAX Compression	41
Excluding Scan Chains From Scan Compression	44
Scan Compression and OCC Controllers	45
Using Compressed Clock Chains	45
Defining External Clock Chains	47
<hr/>	
5. DFTMAX Ultra Compression	49
DFTMAX Ultra Compression Architecture	49
Input Shift Register and Decompression MUX	50
Control Register	51
Output XOR Compression Tree and Shift Register	51
Test Pattern Scan Procedure	52
Scan-Enable Signal Requirements for Codec Operation	53
Multiple-Input, Multiple-Output Architecture	53
Configuring DFTMAX Ultra Compression	55
Using OCC Controllers With DFTMAX Ultra Compression	56
<hr/>	
6. DFT Partitions	58
Defining DFT Partitions	58
Configuring DFT Partitions	59
Per-Partition Scan Configuration Commands	61
set_scan_configuration	62
set_dft_signal	62
set_scan_path	62
set_wrapper_configuration	63
Known Issues of the DFT Partition Flow	63
<hr/>	
7. Pipelined Scan Data	64
Introduction to Pipelined Scan Data	64
The Pipelined Scan Data Logic Structure	65

Configuring Pipelined Scan Data	66
Pipelined Scan Data Test Protocol Format	66
Pipelined Scan Data Limitations	67
<hr/>	
8. Pipelined Scan-Enable Signals	68
The Pipelined Scan-Enable Architecture	68
Pipelined Scan-Enable Requirements	70
Excluding Elements From Pipelined Scan-Enable Configurations	71
Implementing Pipelined Scan-Enable Signals	73
Reporting the Pipelined Scan-Enable Clusters	74
Implementation Considerations for Pipelined Scan-Enable Signals	74
Pipelined Scan Enable Limitations	76
PSE max Fanout per Clock Specification	77
Limitations	78
<hr/>	
9. Core Wrapping	79
Core Wrapping Concepts	79
Wrapper Cells and Wrapper Chains	80
Wrapper Test Modes	83
Core Wrapper Cells	83
Core Wrapper Chains	85
Controlling Register Reuse	86
Special Cases for Register Reuse	90
Wrapper Shift Signals	92
Dedicated Wrapper Cell Insertion in UPF Designs	92
Low-Power Core Wrapping Features	95
Wrapping a Core	96
Enabling Core Wrapping	97
Enabling Soft Core Wrapping for Non-abutted Flow	97
Defining Wrapper Shift Signals	97
Configuring Wrapper Clock Signals	98
Controlling Wrapper Chain Count	99
Setting a Reuse Threshold	100
Configuring Port-Specific Wrapper Settings	100
Mixing Input and Output Wrapper Cells	101

Previewing the Wrapper Cells	101
Creating an EXTEST-Only Core Netlist	103
Defining Core Wrapping Test Modes	104
Top-Down Flat Testing With Transparent Wrapped Cores	105
Introduction to Transparent Test Modes	105
Defining Core-Level Transparent Test Modes	107
Defining Top-Level Flat Test Modes	108
Limitations	109
SCANDEF Generation for Wrapper Chains	109
Limitations of Core Wrapping	110
<hr/>	
10. Clock Gating	111
Clock Gating Concepts	111
Clock-Gating Test Control Points	112
Clock-Gating Test Control Signals	115
Configuring Clock-Gating for DFT	116
Specifying Clock-Gating Control Signals	116
Excluding Clock-Gating Cells From Test-Pin Connection	118
Clock Gating for DFT Wrapper Cells	119
Clock Gating	119
Limitations	120
<hr/>	
11. The Internal Pins Flow	121
Introduction to the Internal Pins Flow	121
Defining Signals on Internal Pins	122
Writing Out the Test Protocol	123
Limitations	123
<hr/>	
12. Generating SCANDEF for Scan Reordering in Layout	124
Introduction to SCANDEF	124
SCANDEF Constructs	125
Writing Out the SCANDEF Information	127
Limitations of SCANDEF Generation	127

13. On-Chip Clocking (OCC)	129
Background	130
Clock Type Definitions	131
OCC Controller Structure and Operation	132
DFT-Inserted and User-Defined OCC Controllers	133
OCC-Controlled Clock Relationships	134
OCC Controller Signal Operation	135
Clock Chain Operation	136
Logic Representation of an OCC Controller and Clock Chain	137
Scan-Enable Signal Requirements for OCC Controller Operation	137
Enabling On-Chip Clocking Support	138
Specifying OCC Controllers	138
Specifying DFT-Inserted OCC Controllers	138
Defining Clocks	139
Defining Global Signals	140
Configuring the OCC Controller	141
Configuring the Clock Chain	142
Configuring the Clock-Gating Logic	144
Performing Timing Analysis	145
Script Example	145
Specifying Existing User-Defined OCC Controllers	147
Defining Clocks	147
Defining Global Signals	149
Specifying Clock Chains	150
Script Example	151
Specifying OCC Controllers for External Clock Sources	153
Reporting Clock Controller Information	154
DFT-Inserted OCC Controller Flow	154
Existing User-Defined OCC Controller Flow	154
Controlling the OCC Controller Bypass Path	155
DFT-Inserted OCC Controller Configurations	156
Single OCC Controller Configurations	156
Example 1	156
Example 2	157
Example 3	157
Multiple DFT-Inserted OCC Controller Configurations	158
Example 1	158
Example 2	159

Waveform and Capture Cycle Example	160
Limitations	161
<hr/>	
14. Inserting Test Points	162
Test Point Types	162
Force Test Points	162
Control Test Points	163
Observe Test Points	165
Multicycle Test Points	166
Test Point Structures	166
Test Point Components	167
Test Point Register Clocks	167
Sharing Test Point Registers	169
Automatically Inserted Test Points	171
Enabling Automatic Test Point Insertion	173
Configuring Global Test Point Insertion Settings	173
Configuring the Random-Resistant Test Point Target	174
Configuring the Untestable Logic Test Point Target	175
Configuring the X-Blocking Test Point Target	177
Configuring the Multicycle Path Test Point Target	178
Configuring the Shadow Wrapper Test Point Target	179
Configuring the Core Wrapper Test Point Target	180
Configuring the User-Defined Test Point Target	181
Enabling Multiple Targets in a Single Command	182
Implementing Test Points From an External File	183
Customizing the Test Point Analysis	185
Running Test Point Analysis	186
Automatic Test Point Insertion Example Script	186
Limitations	187
Inserting the Test Point Logic	188
Inserting Power-Aware Test Points	189
<hr/>	
15. Previewing, Inserting, and Checking DFT Logic	190
Previewing the DFT IP	190
Inserting the DFT IP	192
Pre-DFT DRC	192

Contents

Inserting Scan Structures	193
Scan Structure Insertion in UPF Designs	194
Post-DFT DRC	195
Customizing the DFT DRC Analysis	196

About This User Guide

The Synopsys Fusion Compiler tool provides a complete Design-for-Test (DFT) flow.

This guide describes the Fusion Compiler Design-for-Test flow. For more information about the Fusion Compiler tool, see the following companion volumes:

- *Fusion Compiler User Guide*
- *Fusion Compiler Graphical User Interface User Guide*

This user guide is for design engineers who use the Fusion Compiler tool to insert scan logic into a design.

This preface includes the following sections:

- [New in This Release](#)
- [Related Products, Publications, and Trademarks](#)
- [Conventions](#)
- [Customer Support](#)

New in This Release

Information about new features, enhancements, and changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the Fusion Compiler Release Notes on the SolvNetPlus site.

Related Products, Publications, and Trademarks

For additional information about the Fusion Compiler tool, see the documentation on the Synopsys SolvNetPlus support site at the following address:

<https://solvnetplus.synopsys.com>

You might also want to see the documentation for the following related Synopsys products:

- TestMAXTM ATPG
- TestMAXTM Advisor

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates syntax, such as <code>write_file</code> .
<i>Courier italic</i>	Indicates a user-defined value in syntax, such as <code>write_file design_list</code>
Courier bold	Indicates user input—text you type verbatim—in examples, such as prompt> write_file top
Purple	<ul style="list-style-type: none"> • Within an example, indicates information of special interest. • Within a command-syntax section, indicates a default, such as <code>include_enclosing = true false</code>
[]	Denotes optional arguments in syntax, such as <code>write_file [-format fmt]</code>
...	Indicates that arguments can be repeated as many times as needed, such as <code>pin1 pin2 ... pinN</code> .
	Indicates a choice among alternatives, such as <code>low medium high</code>
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Bold	Indicates a graphical user interface (GUI) element that has an action associated with it.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy .
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.

Customer Support

Customer support is available through SolvNetPlus.

Accessing SolvNetPlus

The SolvNetPlus site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNetPlus site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNetPlus site, go to the following address:

<https://solvnetplus.synopsys.com>

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

If you need help using the SolvNetPlus site, click REGISTRATION HELP in the top-right menu bar.

Contacting Customer Support

To contact Customer Support, go to <https://solvnetplus.synopsys.com>.

1

DFT Insertion in the Fusion Compiler Tool

The Fusion Compiler tool unifies RTL synthesis and DFT insertion, which provides improved QoR compared to traditional multistage synthesis and DFT insertion flows.

The Fusion Compiler tool supports the in-compile flow; it optimizes the location of DFT operations within the stages of the `compile_fusion` command.

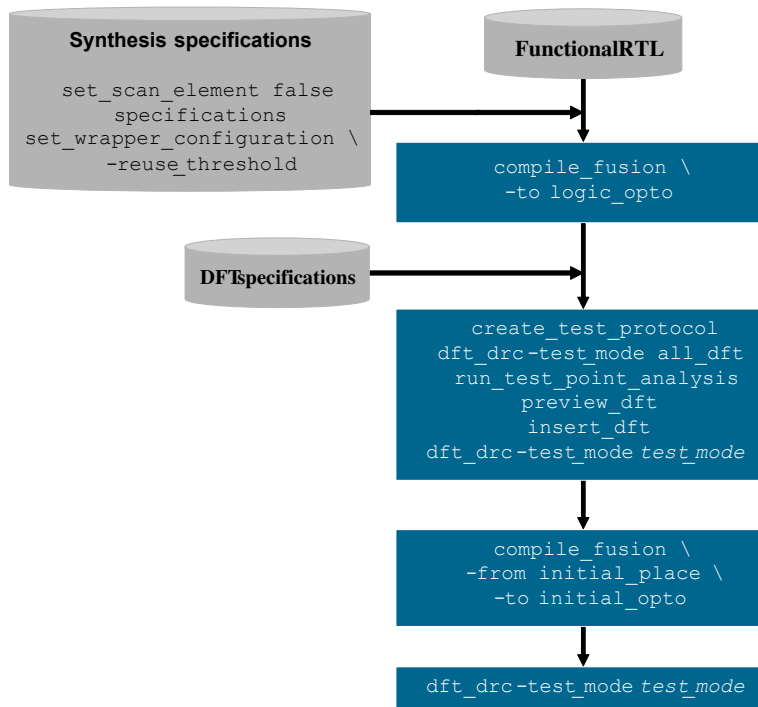
The DFT in-compile flow provides the flexibility to use the `preview_dft` command at the gate level. In addition, you can perform a second set of test DRC checks after the `insert_dft` command. During the final compile step (`compile_fusion -from initial_place`), the tool accounts for scan reordering during placement if the SCANDEF information is available.

In this flow, a few configuration steps must be performed before the first `compile_fusion` execution, which stops at the `logic_opto` stage, as follows:

- The `set_scan_element false` command controls scan replacement of sequential cells during sequential cell mapping.
- The `set_wrapper_configuration` command enables core wrapping analysis to prevent multibit banking of core wrapping registers of different types.

Other DFT configuration commands should be executed after the `compile_fusion -to logic_opto` command but before the `insert_dft` command. [Figure 1](#) illustrates the flow.

Figure 1 The DFT In-Compile Flow



Note:

The tool issues an NDMUI-441 warning message when you set the application option to enable the in-compile flow and issues a DFT-1158 warning message, when you issue any other DFT-related command.

To remove these warning messages, remove this application option from your script:

```
set_app_options -name dft.insertion_post_logic_opto -value true
```

The DFT Post-Compile Flow

Some third-party tools require a mapped netlist to generate a DFT IP netlist to integrate LBIST, MBIST, or codec information.

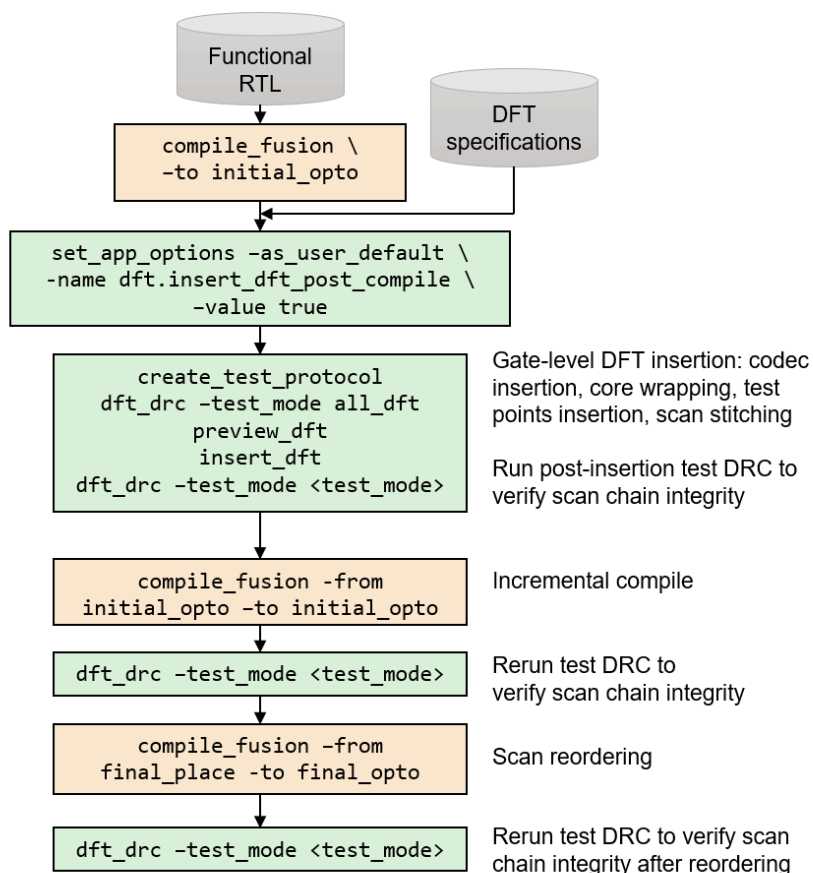
In this case, the DFT post-compile flow allows you to insert third-party DFT codecs into a gate-level netlist after the `compile_fusion` command. However, you must subsequently execute an incremental compile step to incorporate the new logic by running the `compile_fusion -from initial_opto -to initial_opto` command.

Performing a standalone DFT insertion on an optimized gate-level netlist allows the tool to better consider the DFT configuration options. Debugging the DFT setup is simplified

because it requires iterating only over the DFT insertion step. However, the incremental compile step requires additional runtime.

Figure 2 illustrates the steps in the post-compile flow.

Figure 2 The DFT Post-Compile Flow



Setting DFT Optimization Goals

The `optimize_dft` command performs physical DFT optimization and is automatically executed after coarse placement occurs in commands such as `create_placement`, `compile_fusion`, `place_opt`, and `clock_opt`.

You can optionally specify a priority for physical DFT optimization goals while running the `optimize_dft` command by using the `set_optimize_dft_options` command. By default, the DFT optimization goals, the `opt.dft.use_ng_engines`

application option is set to `true`. To switch back to a legacy optimization engine, set the `opt.dft.use_ng_engines` application option to `false`.

For the in-compile flow, set the application option and the goal before executing the `compile_fusion -from_initial_place` command. For the post-compile flow, set the option and the goal before executing the `compile_fusion -from_initial_opto -to_initial_opto` command.

The DFT optimization goals are as follows:

- `total_interconnect_length` (default)
This goal minimizes the total scan data net length across all scan chains. In general, this setting improves routability by reducing the routed wire length.
- `max_interconnect_length`
This goal minimizes the length of the longest scan data net across all scan chains. This setting improves timing on the scan data paths. This is useful if the scan chains are shifted at a high frequency.

For example, the following command enables DFT optimization goals and sets the goal to `total_interconnect_length` by default:

```
fc_shell> set_app_options -list {opt.dft.use_ng_engines true}
```

The following commands enable DFT optimization goals and set the goal to `max_interconnect_length`:

```
fc_shell> set_app_options -list {opt.dft.use_ng_engines true}  
fc_shell> set_optimize_dft_options {-goal max_interconnect_length}
```


2

Basic DFT Configuration

This chapter shows basic DFT configuration tasks that are common to all DFT insertion flows in Fusion Compiler.

It includes the following topics:

- [Defining DFT Signals](#)
- [Specifying a Hookup Pin for DFT-Inserted Clock Connections](#)
- [Specifying a Location for DFT Logic Insertion](#)

Defining DFT Signals

This topic describes how to define test clocks. It includes the following:

- [Defining Scan Signals](#)
- [Defining Test Clocks](#)

Defining Scan Signals

Fusion Compiler requires that you explicitly define all needed DFT-related signals on existing ports (or pins); it does not create new signals. Use the `set_dft_signal` command to define these signals for the current design. [Table 1](#) lists the basic scan-related signals.

Table 1 signal_type Attribute Values for Test Signals

Test I/O port signal	signal_type value	Port direction
Scan-in	ScanDataIn	Input
Scan-out	ScanDataOut	Output
Scan-enable	ScanEnable	Input
Scan clock	ScanClock	Input
Asynchronous sets/resets	Reset	Input

The following is an example of the `set_dft_signal` command specifying a scan-in port. If you enter

```
fc_shell> set_dft_signal -view spec -port SI -type ScanDataIn
```

the tool responds with the following:

```
Accepted dft signal specification for all_dft mode
```

In this example, the `-view spec` option indicates that the specified ports are to be used during DFT scan insertion and that Fusion Compiler is to perform the connections. In this example, `SI` is the name of the scan-in port that DFT insertion uses. (The other value of the `-view` argument is `-existing_dft`, which directs the tool to use the specified ports as is because they are already connected.)

Follow these guidelines when using the `set_dft_signal` command:

- Use the `set_dft_signal -view existing_dft` command if there are existing connections to this signal in your design.
- Use the `set_dft_signal -view spec` command if you expect Fusion Compiler to make connections to this signal for you.

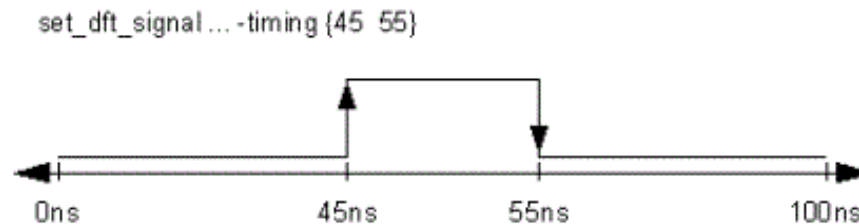
Defining Test Clocks

To explicitly define test clocks in your design, use the `set_dft_signal` command. For example,

```
fc_shell> set_dft_signal -view existing_dft -type ScanClock \  
-port CLK -timing {45 55}
```

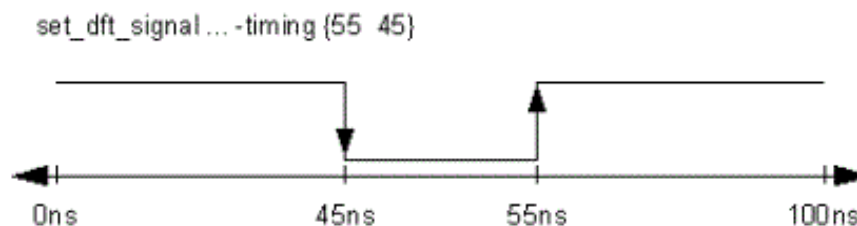
Define the test clock waveform with the `-timing` option. The waveform definition is a pair of values that specifies the rising-edge arrival time followed by the falling-edge arrival time. [Figure 3](#) shows a return-to-zero clock waveform definition.

Figure 3 Return-to-Zero Test Clock Waveform Definition



[Figure 4](#) shows a return-to-one clock waveform definition.

Figure 4 Return-to-One Test Clock Waveform Definition



For all test clocks, the clock period is the value defined by the `dft.test_default_period` application option, which defaults to 100 ns.

The rise and fall clock waveform values are the same as the values specified in the statements that make up the STIL waveform section. The rise argument becomes the value of the rise argument in the waveform statement in the test protocol clock group. The fall argument becomes the value of the fall argument in the waveform statement in the test protocol clock group.

Specifying a Hookup Pin for DFT-Inserted Clock Connections

In some cases, Fusion Compiler might need to make a connection to an existing scan clock network during DFT insertion. Some examples are

- Clock connections for DFT-inserted pipelined scan data registers
- Clock connections for test point registers
- ATE clock connections for DFT-inserted OCC controllers
- Codec clock connections for serialized and streaming scan compression
- Self-test clock connections to the LogicBIST self-test controller and codec

By default, Fusion Compiler makes the clock connection at the source port specified in the `-view existing_dft` signal definition. However, if you want the tool to make the clock connection at an internal pin, such as a pad cell or clock buffer output, you can specify it with the `-hookup_pin` option in a subsequent `-view spec` signal definition. For example,

```
fc_shell> set_dft_signal -view existing_dft -type ScanClock \  
           -port CLK -timing {45 55}  
fc_shell> set_dft_signal -view spec -type ScanClock \  
           -port CLK -hookup_pin UCLKBUF/Z
```

You do not specify the clock waveform timing for the `-view spec` signal definition, but you must specify the associated port with the `-port` option.

Specifying a Location for DFT Logic Insertion

By default, Fusion Compiler places global test logic, such as test-mode decode logic and compressed scan codecs, at the top level of the current design. Other test logic types, such as lock-up latches and reconfiguration MUXs, are placed at the local point of use.

You can specify alternative insertion locations for different types of test logic with the `set_dft_location` command:

```
set_dft_location dft_hier_name  
[-include test_logic_types]
```

The specified instance name must be a hierarchical cell. It cannot be a library cell, black box, or black-box CTL model.

If the specified hierarchical cell does not exist, the `insert_dft` command ignores the specification.

By default, all test logic is synthesized inside the specified hierarchical cell. To synthesize only some types of test logic at that location, use the `-include` option and list the test logic types to be included in the specified cell.

For example, to place the scan compression logic inside `my_cell`, and place the remaining test logic at the top level:

```
fc_shell> set_dft_location my_cell -include {CODEC}
```

The valid test logic types are:

- CODEC

This type includes the compressor and decompressor (codec) logic inserted by the tool for compressed scan, serialized compressed scan, and streaming compressed scan modes.

- WRAPPER

This type includes core wrapping cells and wrapper mode logic configured by the `set_wrapper_configuration` command. For more information, see [Chapter 9, Core Wrapping](#).

3

Scan Architecture Concepts

When Fusion Compiler creates a test protocol, it uses defaults for the clock timing, based on the clock type, unless you explicitly specify clock timing.

This topic shows you how to set test clocks and handle multiple clock designs. It includes the following:

- [Requirements for Valid Scan Chain Ordering](#)
- [Lock-Up Latch Insertion Between Clock Domains](#)
- [Automatically Creating Skew Subdomains Within Clock Domains](#)
- [Manually Creating Skew Subdomains at Associated Internal Pins](#)
- [Manually Creating Skew Subdomains With Scan Skew Groups](#)
- [Building Scan Chains From Base Chains](#)
- [Retiming Registers](#)
- [Multivoltage Support](#)
- [Scan and Stub Chains](#)

Requirements for Valid Scan Chain Ordering

This topic describes the requirements for valid scan chain ordering in the multiplexed flip-flop scan style.

Fusion Compiler generates valid mixed-clock scan chains based on the ideal test clock timing. Scan chain cells are ordered by the ideal test clock edge times, as defined with the `-timing` option of the `set_dft_signal` command. Cells clocked by later clock edges are placed before cells clocked by earlier clock edges. This guarantees that all cells in the scan chain get the expected data during scan shift.

[Figure 5](#) shows the ideal test clock waveforms for two test clocks. The clock edges are numbered by their edge timing order, with the latest clock edge indicated by (1).

Figure 5 Ideal Test Clock Waveforms for Two Test Clocks

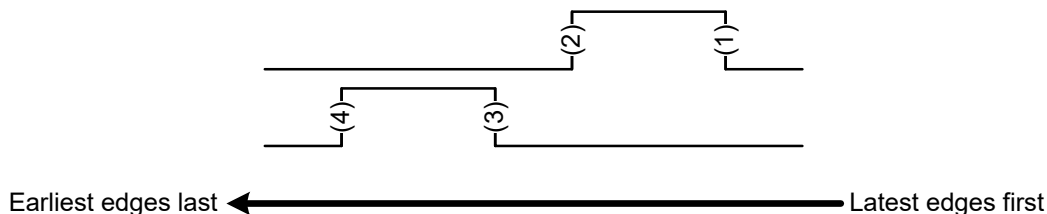
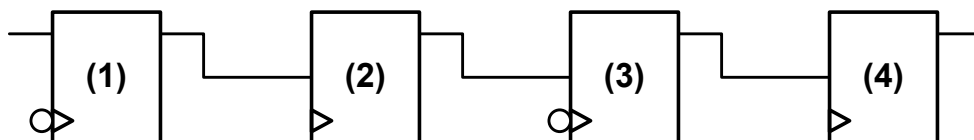


Figure 6 shows how Fusion Compiler constructs a scan chain containing a scan cell clocked by each clock edge. The scan cells are ordered with the cells clocked by the latest clock edges coming first.

Figure 6 Scan Chain Cells for Two Test Clocks



To maintain the validity of your scan chains, do not change the test clock timing after assembling the scan structures.

Although Fusion Compiler chooses an order that ensures correct shift function under ideal clock timing, it cannot guarantee that capture problems will not occur. Capture problems are caused by your logic functionality; modify your design to correct capture problems.

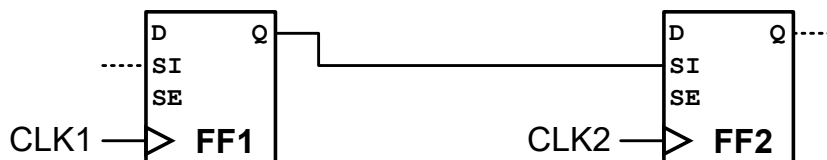
By default, when you request clock mixing within a multiplexed flip-flop scan chain, Fusion Compiler inserts lock-up latches to prevent timing problems. For more information, see [Lock-Up Latch Insertion Between Clock Domains](#).

Lock-Up Latch Insertion Between Clock Domains

A *scan lock-up latch* is a retiming sequential cell on a scan path that can address skew problems between adjacent scan cells when clock mixing or clock-edge mixing is enabled. Fusion Compiler inserts them to prevent skew problems that might occur.

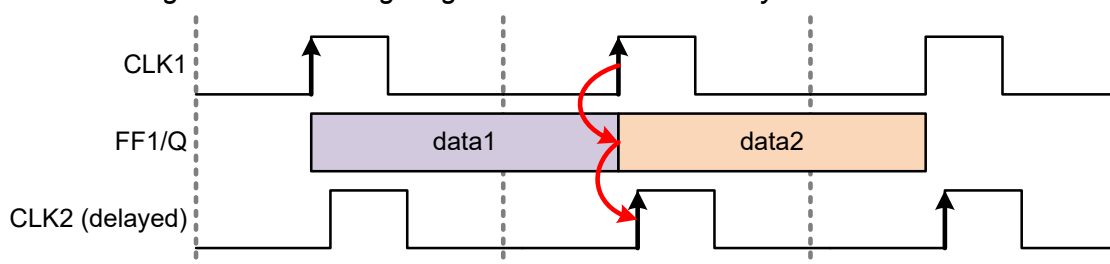
Consider the scan structure in Figure 7, where a scan cell clocked by CLK1 feeds a scan cell clocked by CLK2, and both clocks are defined with the same ideal waveform definition.

Figure 7 Two Scan Cells Clocked by Two Different Clocks



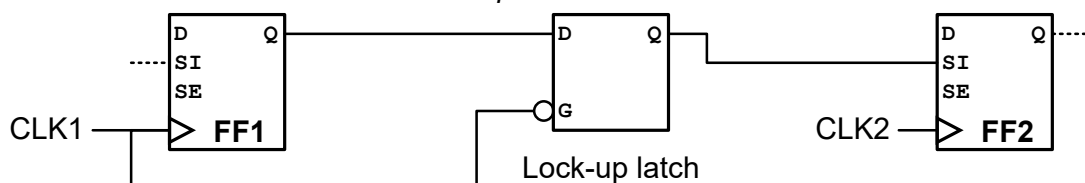
If both scan cells receive a clock edge at the same time, no timing violations occur. However, if the CLK2 waveform at FF2 is delayed, perhaps due to higher clock tree latency, a hold violation might result where FF2 incorrectly captures the current cycle's data instead of the previous cycle's data. Figure 8 shows this hold violation for leading-edge scan cells.

Figure 8 Timing for Two Leading-Edge Scan Cells Clocked by Two Different Clocks



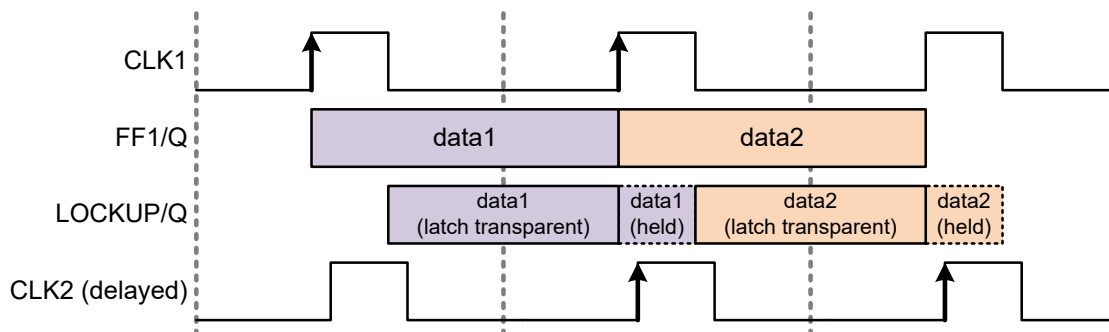
A lock-up latch prevents hold violations for scan cells that might capture data using a skewed clock edge. It is a latch cell that is inserted between two scan cells and clocked by the inversion of the previous scan cell's clock. Figure 9 shows the same two scan cells with a lock-up latch added.

Figure 9 Two Scan Cells With a Lock-Up Latch



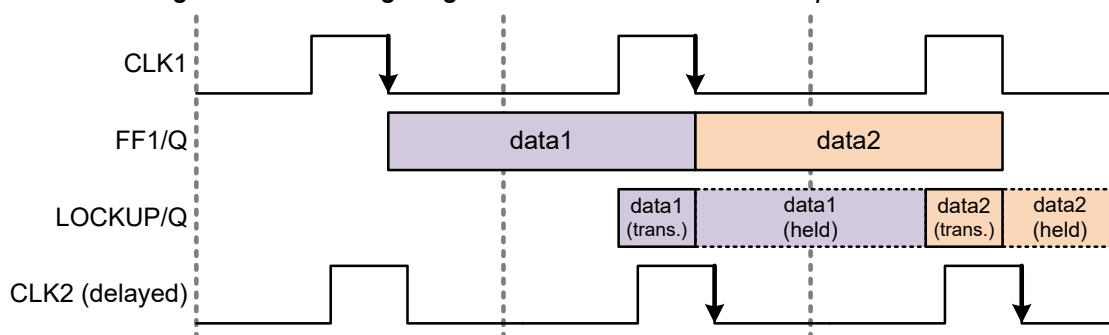
The lock-up latch cell works by holding the previous cycle's scan data while the current cycle's scan data is captured, effectively delaying the output data transition to the next edge of the source clock. Figure 10 shows the lock-up timing behavior for the example. Although this example uses return-to-zero clock waveforms, lock-up latch operation is similar for return-to-one clock waveforms.

Figure 10 Timing for Two Leading-Edge Scan Cells With a Lock-Up Latch



Lock-up latch operation for trailing-edge scan cells is similar to that of leading-edge scan cells, except that the data is held into the next clock cycle as shown in Figure 11.

Figure 11 Timing for Two Trailing-Edge Scan Cells With a Lock-Up Latch



By default, Fusion Compiler adds scan lock-up latches as needed to multiplexed flip-flop scan chains. Scan chain cells are ordered by the ideal test clock edge times, as defined with the `-timing` option of the `set_dft_signal` command. Cells clocked by later clock edges are placed before cells clocked by earlier clock edges. Adjacent scan chain cells clocked by different clock edges are handled as follows:

- When the scan cells are clocked by clock edges with different ideal clock edge timing, Fusion Compiler does not insert a lock-up latch. The second scan cell captures data using an earlier clock edge, and the tool assumes this difference in the ideal clock edge timing is sufficient to avoid a hold time violation.
- When the scan cells are clocked by clock edges with identical ideal clock edge timing, Fusion Compiler inserts a lock-up latch to avoid a potential hold violation due to clock skew.

Fusion Compiler builds scan paths that meet zero-delay timing (without clock propagation delay or uncertainty). In Figure 10, if CLK2 is skewed later than CLK1 by more than the active-high pulse width of CLK1, a hold violation can still occur.

Figure 12 shows a set of ideal test clock waveforms for a set of overlapping test clocks. The clock edges are numbered by their edge timing order, with the latest clock edge indicated by (1). Clock edges with identical ideal clock edge timing are highlighted.

Figure 12 Ideal Test Clock Waveforms for Overlapping Test Clocks

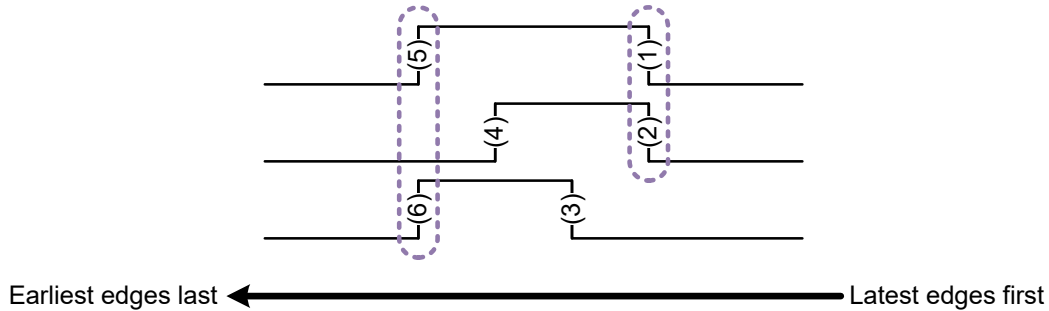
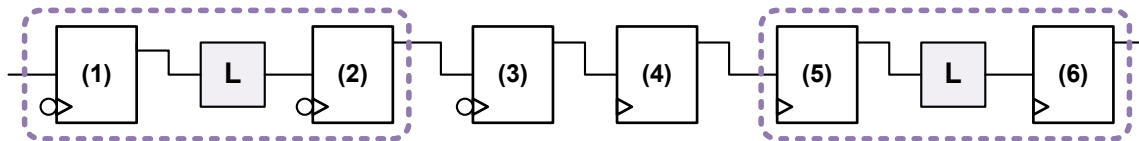


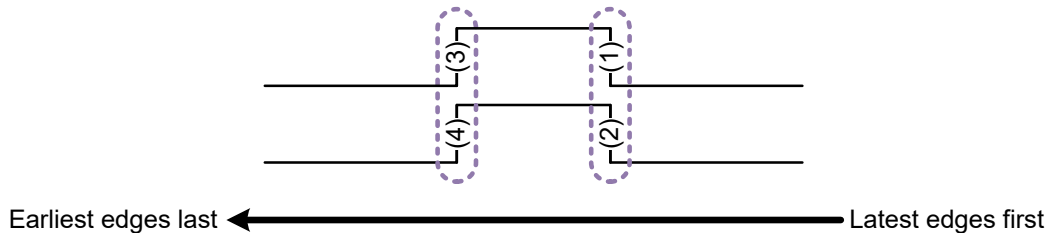
Figure 13 shows how Fusion Compiler constructs a scan chain containing a scan cell clocked by each clock edge. The scan cells are ordered with the cells clocked by the latest clock edges coming first. Lock-up latches are inserted between scan cells clocked by the clock edges with identical ideal clock edge timing.

Figure 13 Scan Chain Lock-Up Latches for Overlapping Test Clocks



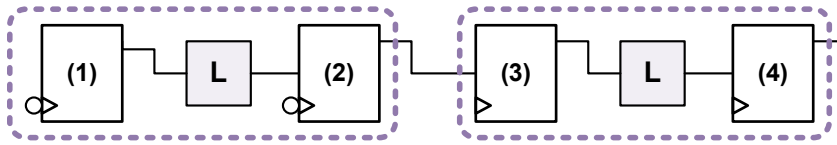
However, in most designs, all test clocks share identical return-to-zero test clock waveforms, as shown in Figure 14.

Figure 14 Ideal Test Clock Waveforms for Simple Test Clocks



In this case, the ordering behavior is simplified. The scan cells are ordered with all falling-edge scan cells first and all rising-edge scan cells last, as shown in Figure 15. Lock-up latches are inserted between differently-clocked scan cells within the rising-edge and falling-edge sections of the scan chain.

Figure 15 Scan Chain Lock-Up Latches for Simple Test Clocks



The `set_scan_configuration` command provides options to control lock-up latch insertion. To add lock-up latches at the end of each scan chain to assist with potential block-to-block timing issues during core integration, use the `-insert_terminal_lockup` option of the `set_scan_configuration` command:

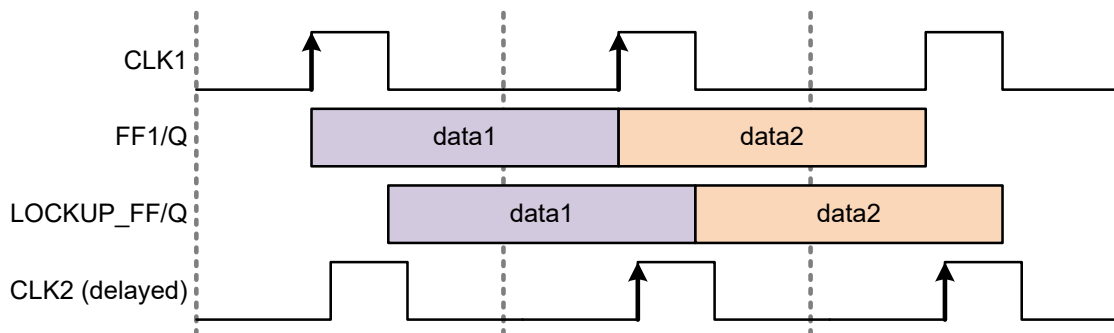
```
fc_shell> set_scan_configuration -insert_terminal_lockup true
```

The default lock-up element type is a level-sensitive lock-up latch. To use a lock-up flip-flop instead, use the `-lockup_type` option of the `set_scan_configuration` command:

```
fc_shell> set_scan_configuration -lockup_type flip_flop
```

When a lock-up flip-flop is used, the data is held as shown in [Figure 16](#).

Figure 16 Timing for Two Scan Cells With a Lock-Up Flip-Flop



Regardless of your selected scan style or configuration, you can explicitly add scan lock-up elements to your scan chain by using the `set_scan_path` command.

For successful lock-up operation, the falling edge of the current scan cell must occur after or concurrent with the rising edge of the next scan cell. This requirement is always inherently met when Fusion Compiler inserts lock-up elements between scan chain cells. However, when you are manually inserting lock-up elements with the `set_scan_path` command, you must ensure that this requirement is met.

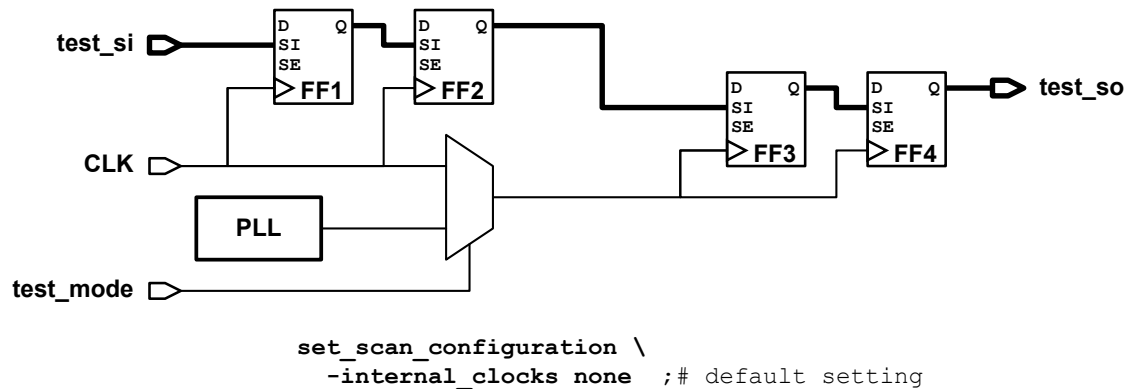
Automatically Creating Skew Subdomains Within Clock Domains

This topic describes how the tool can add lock-up latches within a clock domain between subdomains that might have higher skew between them.

For the purpose of building scan chains, DFT insertion, by default, treats the entire clock network driven by a given clock source as the same-skew clock signal.

Consider the netlist shown in [Figure 17](#), which shows a clock network structure before clock tree synthesis. By default, DFT insertion treats all four flip-flops as belonging to the same-skew top-level clock signal, CLK.

Figure 17 Circuit With Same Top-Level Clock Driving Internal Clock Signals



Note that the MUX cell introduces a delay in the clock network. If clock tree synthesis balances the test mode clock latency equally to all flip-flops, the MUX cell should not cause any timing problems. However, because clock tree synthesis might not consider the test mode clock tree latencies used for scan shift, a potential scan path hold violation could occur at FF3/SI.

To avoid creating this potential hold time violation, you can treat the scan cells downstream from any multi-input cell as a different *skew subdomain* within the clock domain, driven by their own internal clock pin (such as the MUX output pin).

To do this, use the following command:

```
fc_shell> set_scan_configuration -internal_clocks multi
```

This command instructs DFT insertion to

- Identify all multiple-input gates (such as MUX cells) in each clock network.

Note:

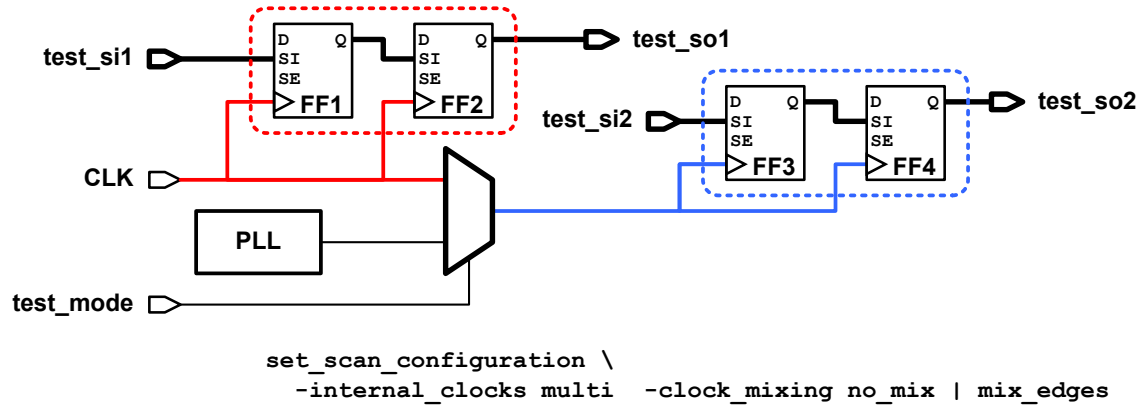
Integrated clock-gating cells, which have the `clock_gating_integrated_cell` attribute defined, are not considered; they are transparent for the determination of skew subdomains.

- Create an internal clock at the output pins of these gates.
- Treat these internal clocks as skew subdomains of the parent clock.

This feature affects only scan chain architecture; the clock network is still a single test clock domain for all other DFT operations, including writing out the test protocol.

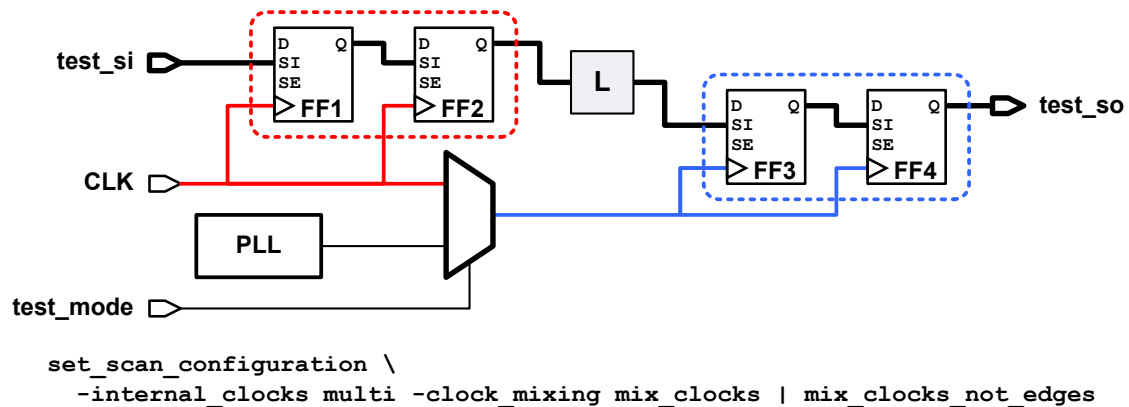
The resulting scan chains depend on the current clock-mixing setting, which is controlled by the `-clock_mixing` option of the `set_scan_configuration` command. If clock mixing is disabled by specifying the `no_mix` or `mix_edges` clock-mixing mode, DFT insertion creates separate scan chains for each internal clock, as shown in Figure 18.

Figure 18 Circuit With Internal Clocks With Clock Mixing Disabled



If clock mixing is enabled by specifying the `mix_clocks` or `mix_clocks_not_edges` clock-mixing mode, DFT insertion can use lock-up latches to keep the scan cells from different internal clocks on the same scan chain, as shown in Figure 19.

Figure 19 Circuit With Internal Clocks With Clock Mixing Enabled



You can create skew subdomains within specific clock domains by using the `-internal_clocks` option of the `set_dft_signal` command. The following command tells DFT insertion to create internal clocks at multiple-input cells only for the CLK domain:

```
fc_shell> set_dft_signal -view existing_dft \  
                  -type ScanClock -timing [list 45 55] \  
                  -internal_clocks multi -port CLK
```

If you set different `-internal_clocks` values using the `set_scan_configuration` and `set_dft_signal` commands, the more specific setting applied with the `set_dft_signal` command takes precedence. For example, assume that you set the following opposing `-internal_clocks` values by using these two commands:

```
fc_shell> set_scan_configuration -internal_clocks none  
  
fc_shell> set_dft_signal -view existing_dft \  
                  -type ScanClock -timing [list 45 55] \  
                  -internal_clocks multi -port CLK
```

Because the value set by the `set_dft_signal` command takes precedence, signals driven by CLK via MUX cells or other multiple-input gates are treated as separate clocks. All other clocks in the design are treated according to the default configuration.

This feature is similar to the `-associated_internal_clocks` feature described in [Manually Creating Skew Subdomains at Associated Internal Pins](#), except that the internal clocks are created at all multi-input cell output pins instead of only user-specified pins.

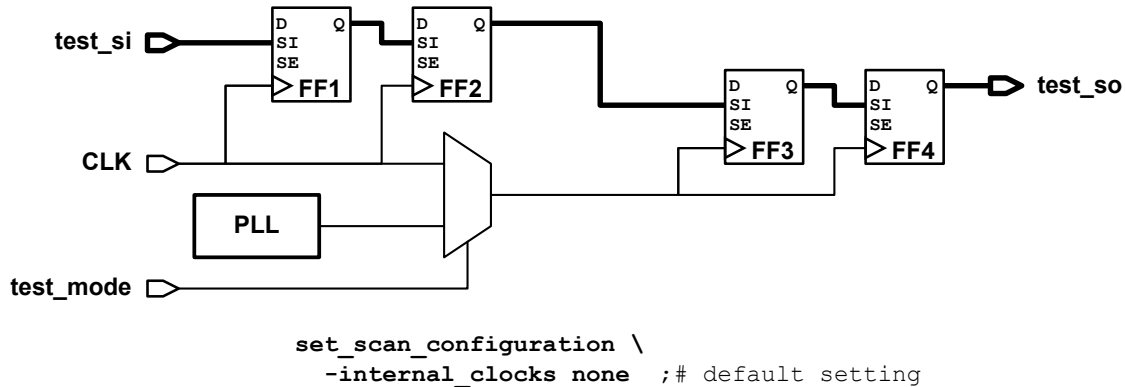
Manually Creating Skew Subdomains at Associated Internal Pins

This topic describes how you can manually define subdomains of a parent clock network that might have higher skew between them.

For the purpose of building scan chains, DFT insertion, by default, treats the entire clock network driven by a given clock source as the same-skew clock signal.

Consider the netlist shown in [Figure 20](#), which shows a clock network structure before clock tree synthesis. By default, DFT insertion treats all four flip-flops as belonging to the same-skew top-level clock signal, CLK.

Figure 20 Circuit With Same Top-Level Clock Driving Internal Clock Signals



Note that the MUX cell introduces a delay in the clock network. If clock tree synthesis balances the test mode clock latency equally to all flip-flops, the MUX cell should not cause any timing problems. However, because clock tree synthesis might not consider the test mode clock tree latencies used for scan shift, a potential scan path hold violation could occur at FF3/SI.

To avoid creating this potential hold time violation, you can treat the scan cells downstream from the MUX cell as a different *skew subdomain* within the clock domain, driven by their own internal clock pin (the MUX output pin).

To do this, define the scan clock as follows:

```
fc_shell> set_dft_signal -view existing_dft -type ScanClock \
    -timing {45 55} -associated_internal_clocks {UMUX/Z}
```

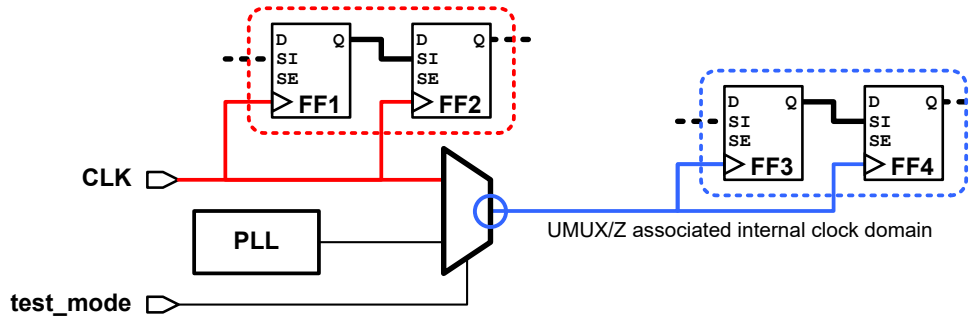
This command instructs DFT insertion to

- Create an internal clock at each associated internal pin in the list
- Treat these internal clocks as skew subdomains of the parent clock

This feature affects only scan chain architecture; the clock network is still a single test clock domain for all other DFT operations, including writing out the test protocol.

In the previous example, the clock network contains a skew subdomain driven by UMUX/Z, plus the remainder of the parent clock domain, as shown in [Figure 21](#). These are treated as separate clock domains according to the `set_scan_configuration -clock_mixing` setting applied to the design.

Figure 21 Circuit With Top-Level Clock Source and Associated Internal Clock Pin



```
set_dft_signal -view existing_dft -type ScanClock -timing {45 55} \
-port CLK -associated_internal_clocks {UMUX/Z}
```

This feature is similar to the `-internal_clocks` feature described in [Automatically Creating Skew Subdomains Within Clock Domains](#), except that the internal clocks are created only at the user-specified pins instead of all multi-input cell output pins.

Associated internal clocks take precedence over any `-internal_clocks` specifications.

Limitations

Note the following requirements and limitations:

- Associated internal clocks can be defined only on leaf pins, not hierarchical pins.
- Pre-DFT DRC drives the clock signal directly at both the clock source and the internal pins, allowing the clock signal to bypass cells that are black boxes in synthesis. However, post-DFT DRC drives the clock signal only at the clock source.

To propagate the clock through blockages during post-DFT DRC, use a custom `test_setup` procedure .

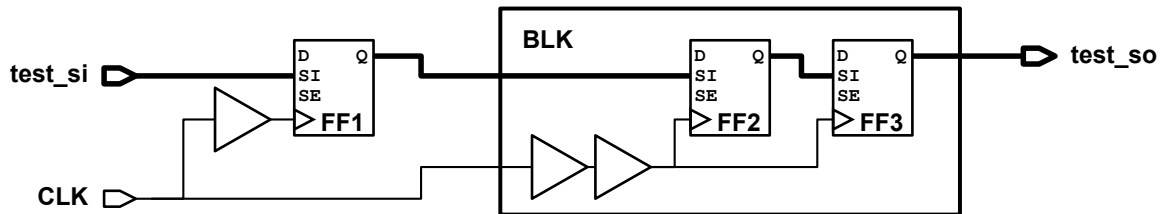
- This feature is used only with `-view existing_dft` clock signal definitions.
- The association is valid only when `-type` is MasterClock, ScanMasterClock, ScanSlaveClock, or ScanClock.
- To remove the internal pin associations, you must use the `remove_dft_signal` command to remove both the DFT signal and the association list.
- The `-hookup_sense` option has no effect. You can associate only the same clock edge from a top-level clock edge to of a list of pins.
- The `report_dft_signal` command does not show the associated internal pins.

Manually Creating Skew Subdomains With Scan Skew Groups

This topic describes how to define a skew subdomain on any arbitrary set of scan cells. Such a set of scan cells is called a *scan skew group*.

In some cases, you might want to provide manual guidance for lock-up latch insertion between areas of the design with potentially differing clock tree latencies. Consider [Figure 22](#), in which block BLK has a different clock latency than the top-level logic.

Figure 22 Circuit With Clock Tree Containing Multiple Latency Regions



A lock-up latch at the scan input pin of BLK would prevent hold violations along the scan path. However, the scan chain is entirely within the same scan clock domain, and there are no multi-input cells that allow the `-internal_clocks` option to be used.

You can use *scan skew groups* to provide manual guidance for lock-up latch insertion. A scan skew group is a group of scan cells that might have a different clock latency characteristic than other parts of the design. The DFT architect treats the scan skew group as a unique *skew subdomain*.

To define a scan skew group, use the `set_scan_skew_group` command:

```
set_scan_skew_group
  group_name
  -include_elements {include_list}
```

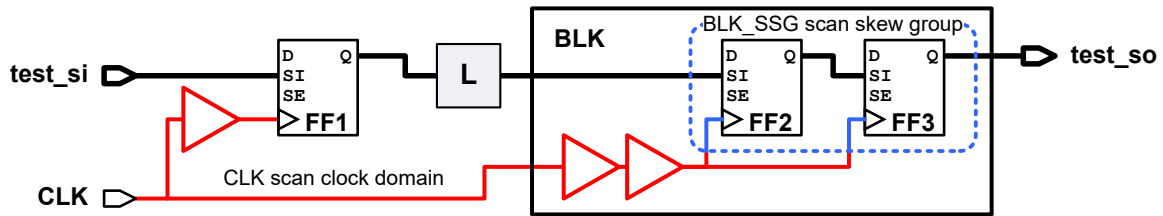
Each scan skew group has a unique name for identification. The include list can contain leaf cells, hierarchical cells, and CTL-modeled cells. Wildcards and collections are supported. You can define as many scan skew groups in your design as needed. You cannot include the same scan cell in multiple scan skew groups.

For the previous example, consider the following scan skew group definition:

```
fc_shell> set_scan_skew_group BLK_SSG -include_elements {BLK/FF*}
```

If clock mixing is enabled, DFT insertion adds a lock-up latch between the top-level and block-level scan cells, as shown in [Figure 23](#). (If clock mixing is disabled, DFT insertion keeps the top-level and block-level scan cells in separate scan chains, not shown.)

Figure 23 Circuit With Lock-Up Latch Due to Scan Skew Group Definition

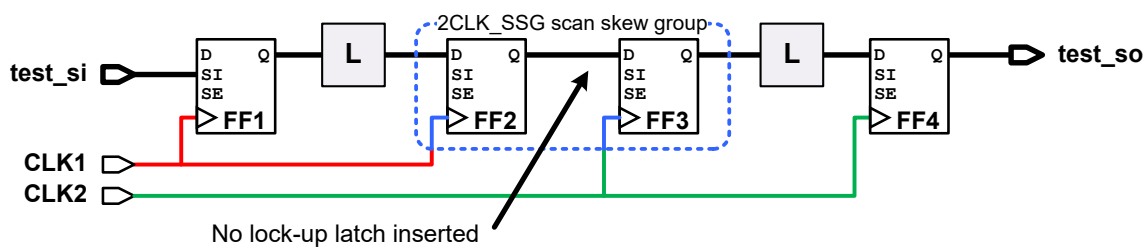


Scan skew groups override the normal scan clock domain identification behaviors such as

- Scan clock name
- The `-internal_clocks` option of the `set_scan_configuration` command

The DFT architect treats all cells in the scan skew group as if they are in the same clock domain. No lock-up latches are inserted between them, as shown in Figure 24. This can occur even if clock mixing is disabled because the scan cells in the scan skew group are no longer treated as belonging to different clock domains.

Figure 24 Circuit With Scan Skew Group Spanning Multiple Clock Domains

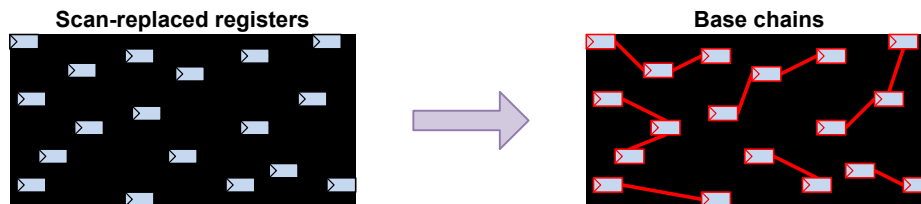


Scan skew groups override only clock identity considerations. They do not override the clock timing considerations of scan chain architecture such as

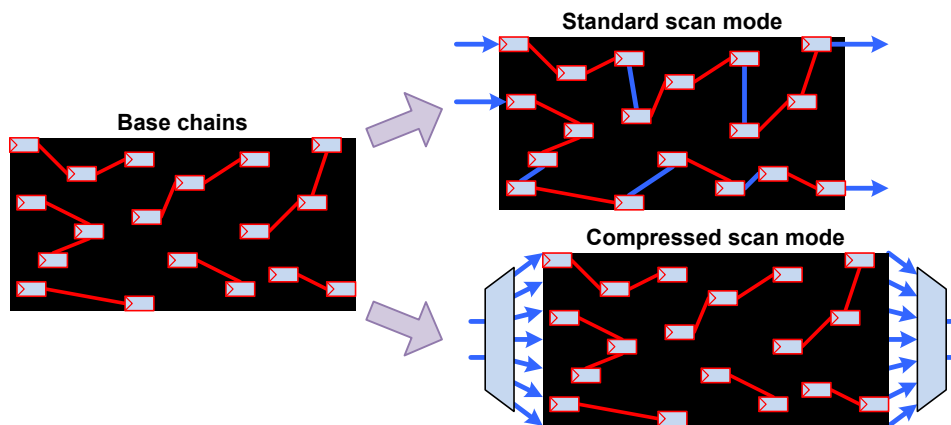
- Scan clock waveform timing
- Scan clock edge polarity

Building Scan Chains From Base Chains

To build scan chains for one or more test modes, Fusion Compiler first builds a set of physically compact *base chains*:



These base chains are then assembled to form the scan chains for each test mode:

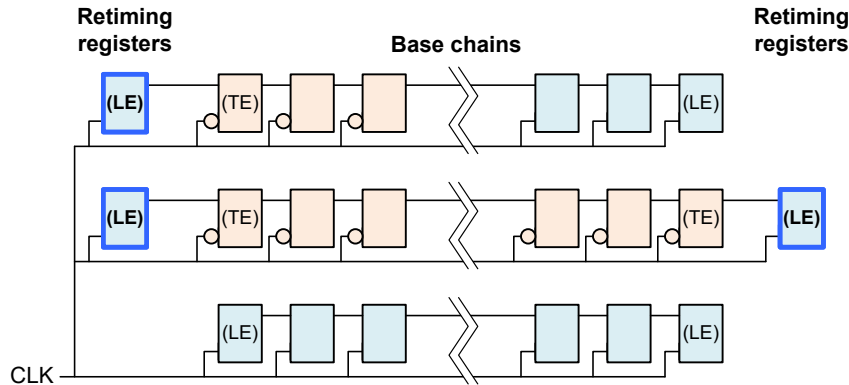


All of this happens automatically during synthesis (the `compile_fusion` command) based on the DFT configuration you have applied.

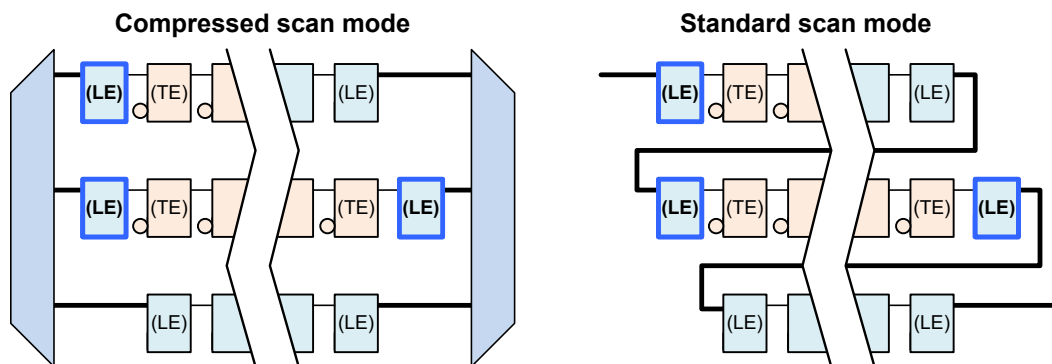
Base chains are the building blocks for architecting scan chains. They can be concatenated together, but they cannot be subdivided.

Retiming Registers

In designs with scan compression, Fusion Compiler automatically inserts leading-edge *retiming registers* at any trailing-edge base chain beginnings and ends to improve quality of results.

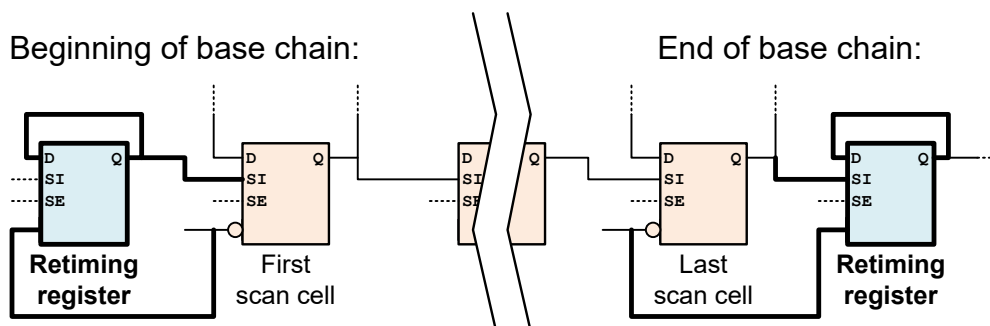


By retiming both ends of the base chains to the leading edge, the tool can concatenate them without being restricted by scan cell clocking edges.



Note that retiming registers are the only case where the tool allows a leading-edge register (blue) to be followed by a trailing-edge register (red).

Retiming registers are implemented using scan registers with a functional loopback path, as shown below. This prevents timing violations when the clock runs at functional frequencies. The clock of the associated scan cell is used.



Retiming registers are not enabled by default for standard-scan-only designs. However, you can manually enable it by using the `-add_test_retiming_flops` option of the `set_scan_configuration` command.

Multivoltage Support

In multivoltage designs, scan chains that include cells from different power domains or voltage areas require special cells to avoid power-related problems. By default, the tool does not place cells from different domains in the same scan chain. However, some power domains or voltage areas have a small number of cells, which results in short scan chains.

You can allow power domain and voltage area mixing for scan chains by using the `set_scan_configuration` command, as follows:

- Use the `-voltage_mixing true` option to build scan chains that can cross voltage areas (the default is `false`). The tool attempts to minimize voltage crossings to reduce the number of level shifters added.
- Use the `-power_domain_mixing true` option to build scan chains that can cross power domains (the default is `false`). The tool attempts to minimize power domain crossings to reduce the number of isolation cells added.
- You can use the `-voltage_mixing true` and `-power_domain_mixing true` options together.

You must also use the `-test_mode all` option to apply power domain or voltage area mixing.

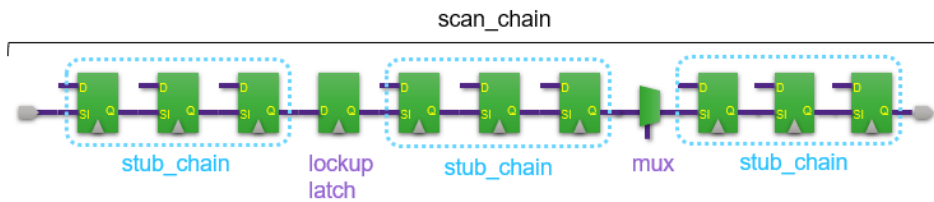
If you use any commands that violate the power domain or voltage area mixing specifications, such as the `set_scan_path` command, the tool issues a warning message at the `preview_dft` or `insert_dft` commands and continues with scan insertion.

You can specify power domain and voltage area mixing only for post-compile DFT flows. After you execute the `insert_dft` command, use the `check_mv_design` command to check for voltage violations introduced by the `insert_dft` and `create_mv_cells` commands. Add missing multivoltage cells as needed.

Scan and Stub Chains

SCANDEF files specify scan elements and their pins used for the scan path along with reordering constraints. A chain obtained from a SCANDEF file is called a *stub chain*. The term *scan chain* refers to the complete logical structure added during the DFT insertion step. A scan chain can contain stub chains, as shown in [Figure 25](#). However, a scan chain might not contain any stub chains. A stub chain can belong to multiple scan chains.

Figure 25 Scan Chain Containing Stub Chains



Use the following commands to examine or report scan chains and stub chains:

- The `get_scan_chains` command creates a collection of scan chains.
- The `report_scan_chains` command displays information about the specified scan chains
- The `get_stub_chains` command creates a collection of stub chains.
- The `report_stub_chains` command displays information about the specified stub chains

The tool classifies scan chains and stub chains as `scan_chain` and `stub_chain` design objects, respectively. You can examine the attributes of these design objects by using the `get_attribute`, `list_attributes`, and `report_attributes` commands. To see a list of attributes, use the `man scan_chain_attributes` or `man stub_chain_attributes` commands.

You can work directly with scan and stub chains by using the following commands:

- The `create_scan_chain` command creates one scan chain from a collection of stub chains.
- The `remove_scan_chain` command removes scan chains.
- The `add_to_scan_chain` command adds stub chains to a scan chain.
- The `remove_from_scan_chain` command removes stub chains from a scan chain.
- The `create_stub_chain` command creates one stub chain from individual elements.
- The `remove_stub_chain` command removes stub chains.
- The `set_stub_chain` command modifies an existing stub chain.

The `-include` and `-exclude` options of the `write_script` command accept an argument value of `scan_chain` to control scan chain output to the `design.tcl` file in the output directory.

The `-of_objects` options of the `get_ports`, `get_pins`, and `get_cells` commands accept scan chains and stub chains.

4

DFTMAX Compression

DFTMAX scan compression provides synthesis-based scan data compression technology to lower the cost of testing larger designs. DFTMAX compression reduces manufacturing test costs by delivering a significant test data and test time reduction with very low silicon area overhead.

The following topics describe DFTMAX scan compression:

- [The DFTMAX Compression Architecture](#)
- [Configuring DFTMAX Compression](#)
- [Excluding Scan Chains From Scan Compression](#)
- [Scan Compression and OCC Controllers](#)
- [Using Compressed Clock Chains](#)
- [Defining External Clock Chains](#)

The DFTMAX Compression Architecture

The DFTMAX compression architecture is described in the following topics:

- [The DFTMAX Codec](#)
- [Decompressor Operation](#)
- [Compressor Operation](#)
- [Compressor X-Blocking](#)
- [Codec Implementation](#)

The DFTMAX Codec

DFTMAX compressed scan appears similar to standard scan at the chip-level interface, but it contains combinational compression logic and uses many more scan chains of shorter lengths within the chip core. As scan input values are shifted in, the decompressor distributes them across numerous scan chains. The distribution method is accomplished

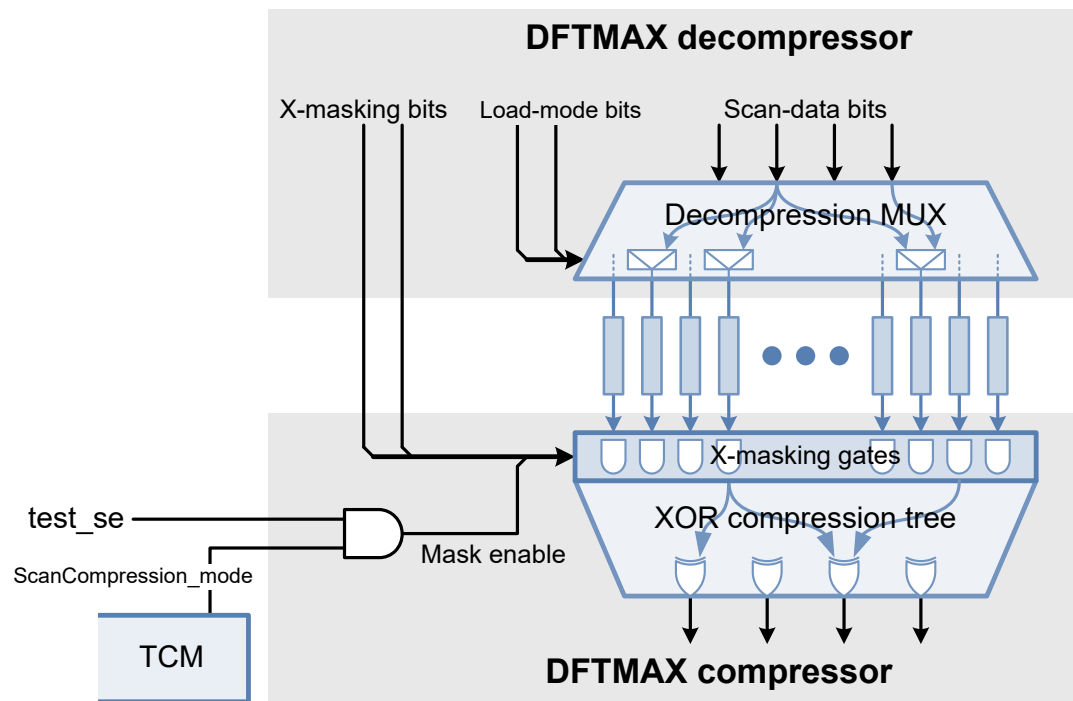
with a patented hardware scheme and a process that allows chip-level scan input values to be placed into various scan chains. To maximize test coverage and minimize pattern count, the decompressor adapts to the needs of automatic pattern generation to supply the required values in the scan chain cells.

DFTMAX compression provides the following key benefits and features:

- A significant test time and test data reduction compared to standard scan
- Similar ease-of-use as standard scan
- Concurrent optimization of area, power, timing, physical constraints, and test constraints through a synthesis-based implementation
- Unknown logic value (X) handling
- Flexible scan channel configurations to support multisite testing and wafer-level burn-in

Figure 26 shows the DFTMAX compression architecture.

Figure 26 DFTMAX Compression Architecture



DFTMAX compression uses a larger number of shorter chains, called *compressed scan chains*, which reduces tester time. The decompressor controls the flow of scan data into the scan chains. The compressor reduces the captured data from the larger number of compressed scan chains so that it can be observed through the scan-out ports. The

combination of the decompressor and compressor wrapped around the scan chains is called the *codec*, which is short for compressor-decompressor.

The codec significantly reduces the amount of test data needed to comprehensively test the chip. In turn, this lowers automatic test equipment (ATE) memory requirements.

Decompressor Operation

The decompressor uses MUX logic to drive the scan chains with different combinations of scan-data pins. One or more scan-in pins, called *load-mode* pins, are dedicated as the MUX select signals.

This logic structure takes advantage of the fact that not every scan cell must be uniquely controllable in every pattern. Typically, only a sparse set of scan cells are required to be controlled in a pattern. In each shift clock cycle, ATPG can choose load-mode and scan data values that steer these required values into the compressed scan chains.

Some designs might have complex logic that requires more scan cells to be controlled in each pattern. As the decompressor input width increases, the number of load-mode and scan-data pins increases to provide additional controllability at the decompressor outputs.

Compressor Operation

Each compressor output is driven by a different combination of compressed scan chains, combined using XOR logic. Thus, an incorrect data value (fault) from a single compressed scan chain results in a specific signature of incorrect values at the compressor outputs.

X-masking gates prevent X (unknown) values from propagating through the XOR logic and destroying other chain observations. See [Compressor X-Blocking](#).

To reduce power, all compressor inputs are masked whenever the codec is not active and shifting.

The compressor input width is equal to the number of compressed scan chains. As the compressed chain count increases, more XOR configurations are needed. If the chain count is increased too high, the XOR configurations (and therefore the compressed chain signatures) must repeat, which can impact the diagnosability of the design.

As the compressor output width increases, the number of fault signatures observed at each output port decreases. This can improve the diagnosability of the design, especially when multiple faults must be simultaneously diagnosed.

Compressor X-Blocking

The X-masking logic in the compressor provides the following observe modes:

- A fully unmasked observe mode, which observes all chains
- Additional masked observe modes, each of which masks and observes a different pattern of chains

Information about the X-blocking observe modes is contained in the SPF (in the `CompressorStructures` section, then in the `Compressor "U_compressor_modename"` definition).

Codec Implementation

When scan compression is used, the compressed chains are architected, then used as the base chains for other test modes. This reduces the number of reconfiguration MUXs needed to implement the other modes.

The decompressor and compressor use congestion-reduction logic structures to efficiently connect to the physically distributed base chains.

The X-masking logic introduces a combinational path between the scan-in and scan-out ports (through the mask signal and XOR reduction logic). This path can potentially contain long routes as well as combinational logic. You can use the pipelined scan data feature to relax these timing requirements. For more information, see [Chapter 7, Pipelined Scan Data](#).

Configuring DFTMAX Compression

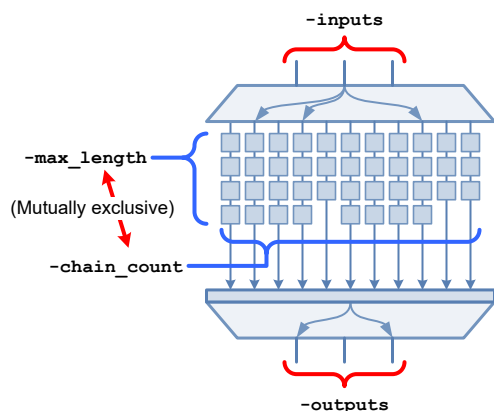
To enable and configure DFTMAX compression, use the following command:

```
set_scan_compression_configuration
  -inputs num_scanins
  -outputs num_scanouts
  -chain_count num_chains | -max_length chain_length
  [-test_mode mode_name]
```

The `-inputs` and `-outputs` options are required. There must be enough scan-in and scan-out signals defined to satisfy the specification.

The `-chain_count` and `-max_length` options are mutually exclusive; one is required.

These options apply to the compression architecture as follows:



Scan compression is automatically enabled when you configure it.

By default, the command creates and configures a mode named `ScanCompression_mode`. If you are using user-defined test modes, you specify the mode to configure with the `-test_mode` option.

If you are also implementing a standard scan mode, keep in mind that these compressed chains become the base chains used to create other modes.

If the specified compressed scan chain count cannot be satisfied, the tool issues an error message that contains information about how many compressed scan chains were requested and how many compressed scan chains can be built for the current scan pin configuration. [Example 1](#) shows the error message issued when 20 compressed scan chains are requested, but only 12 scan chains can be built.

Example 1 High X-Tolerance Error Message for Insufficient Scan-In Pins

```
Error: Architecting of Load/Unload compressor failed with the given set
of parameters. (TEST-1722)
    Number of internal chains architected:      20
    Number of available compression channels:  12
    Number of load compressor inputs:          3
    Number of unload compressor outputs:       3
```

You can also use the TestMAX ATPG `analyze_compressors` command to determine if a codec can be built for a given set of parameters. For more information, see TestMAX ATPG and TestMAX Diagnosis Online Help.

Scan-In and Scan-Out Requirements

The codec architecture imposes a limit on the number of compressed scan chains you can build with a given number of scan-in and scan-out pins. [Table 2](#) shows the maximum number of compressed scan chains that can be built for a given set of scan-in and scan-out pins. The listed maximum number of chains assumes that a single decompressor input

is dedicated to OCC clock chains. Additional dedicated clock chain decompressor inputs will reduce the limit.

Table 2 *Compressed Scan Chain Limits*

Number of scan-in and scan-out pins	Maximum number of chains without OCC controller	Maximum number of chains with OCC controller
2	4	
3	12	6
4	32	16
5	80	40
6	192	96
7	448	224
8	1024	512
9	2304	1152
10	5120	2560
11	11264	5632
12	24576	12288
13	32000	26624
14 and higher	32000	32000

If the on-chip clocking (OCC) feature is used with compressed clock chains, the dedicated decompressor scan input lowers the limit. For more information about compressed clock chains, see [Scan Compression and OCC Controllers](#).

[Table 3](#) shows the maximum compressed scan chain count when high X-tolerance is used with some asymmetrical low-pin-count configurations:

Table 3 *Compressed Scan Chain Limits for Asymmetrical Scan Pins*

Asymmetrical scan-in, scan-out pin configuration	Maximum number of chains without OCC controller	Maximum number of chains with OCC controller
2 scan-ins, 1 scan-out	2	
3 scan-ins, 1 scan-out	4	

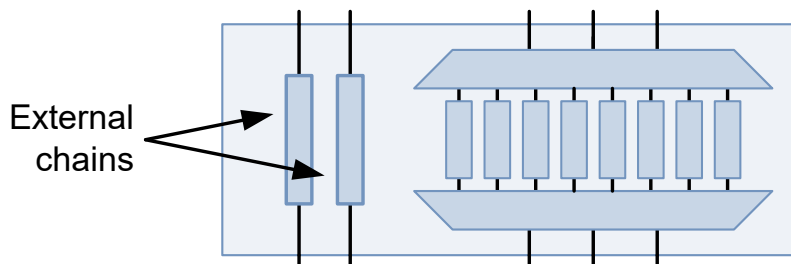
Table 3 Compressed Scan Chain Limits for Asymmetrical Scan Pins (Continued)

Asymmetrical scan-in, scan-out pin configuration	Maximum number of chains without OCC controller	Maximum number of chains with OCC controller
3 scan-ins, 2 scan-outs	8	4
4 scan-ins, 3 scan-outs	24	12
5 scan-ins, 4 scan-outs	64	32
6 scan-ins, 5 scan-outs	160	80

Excluding Scan Chains From Scan Compression

In some cases, you might need to exclude specific scan cells from scan compression by keeping them on a separate uncompressed scan chain. Such a scan chain is called an *external chain*. Figure 27 shows two external chains in a compressed scan design.

Figure 27 External Chains in a Compressed Scan Design



To define an external chain, use the `-scan_data_in` and `-scan_data_out` options of the `set_scan_path` command:

```
set_scan_path scan_chain_name
-test_mode all
-complete true
-include_elements element_list
-scan_data_in port_name -scan_data_out port_name
```

Note the following aspects of external chain definitions:

- The `-test_mode all` option applies the external chain definition to both the standard scan and compressed scan modes. To limit the definition to a specific compressed scan mode, specify it with the `-test_mode` option. The specified test mode must be previously defined with the `define_test_mode` command.
- The scan input and output ports must be previously defined with the `set_dft_signal` command.

- If you are using the pipelined scan data feature, external chains are treated the same as other scan chains. This means, in the automatically inserted pipelined scan data flow, the tool inserts pipeline registers around the external scan chains the same way it does with other scan chains.

Scan Compression and OCC Controllers

On-chip clocking (OCC) controllers allow on-chip clock sources to be used for at-speed capture during device testing. In an OCC controller flow, the *clock chain* is a special scan segment that provides control over the at-speed capture pulse sequence generated by the OCC controller.

In a scan compression flow, the clock chain can be compressed or external (uncompressed), as described in the following topics:

- [Using Compressed Clock Chains](#)
- [Defining External Clock Chains](#)

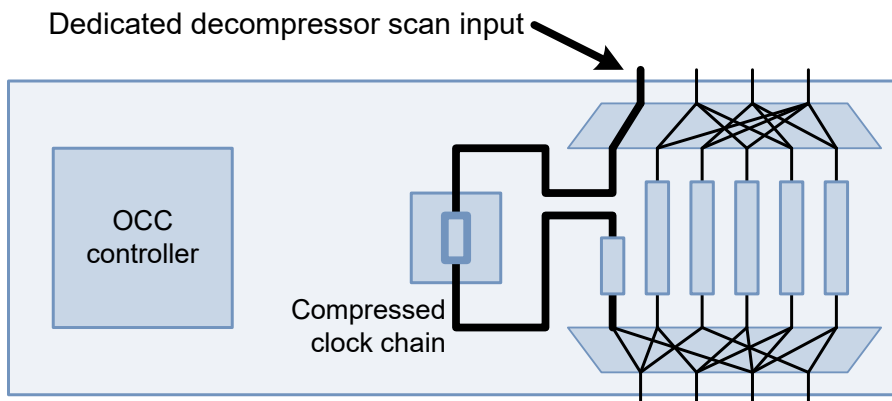
See Also

- [On-Chip Clocking \(OCC\)](#)

Using Compressed Clock Chains

In a DFTMAX flow, when you insert scan compression in a DFT-inserted or user-defined OCC controller flow, the clock chain is placed between the decompressor and compressor by default. The decompressor drives the clock chain along with the other compressed scan chains, but it dedicates a decompressor scan input to the clock chain as shown in [Figure 28](#).

Figure 28 *Compressed Clock Chain in a Compressed Scan Design*



The decompressor scan input path passes through the decompressor to the clock chain input. This allows the clock chain values to be controlled without imposing constraints on other scan cells. Such a clock chain is called a *compressed clock chain* because it exists between the decompressor and compressor, even though it is driven by a dedicated scan-in signal as if it was uncompressed.

For codecs with few scan inputs and high compression ratios, DFTMAX compression might be forced to share the decompressor scan input with other compressed chains. This happens if the codec would otherwise not be implementable.

The dedicated scan-in signal reduces the number of scan-in signals available for data decompression into the remaining compressed chains. You should consider this when determining compression architecture parameters such as scan input count and compressed chain count.

The clock chain, which is clocked on the trailing edge, is always placed at the beginning of its compressed scan chain. Additional scan cells can follow the clock chain for length-balancing purposes, as allowed by the clock-mixing settings applied to the current design. The compressed scan chain then proceeds into the compressor in the normal way.

The compressed clock chain reduces the maximum compressed scan chain limit that can be created for a given number of scan-in signals. For more information, see [Scan-In and Scan-Out Requirements](#).

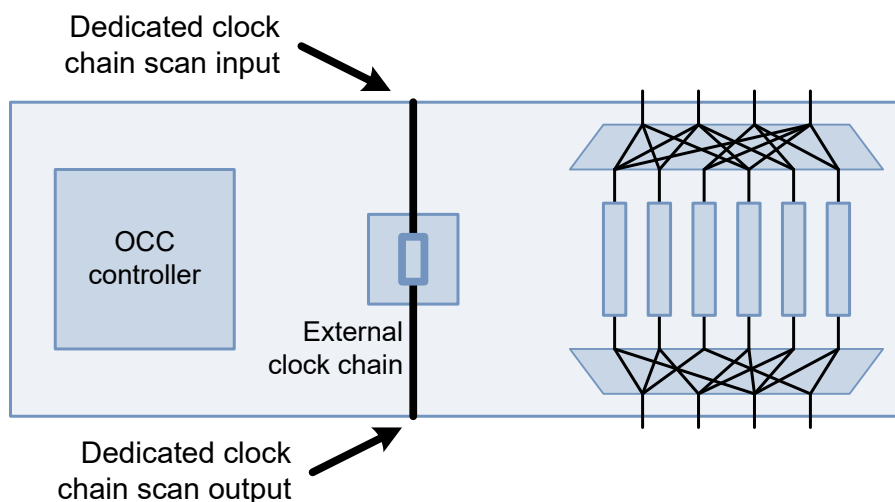
You can use the `preview_dft -show {cells scan_clocks}` command to see which compressed scan chain contains the clock chain. The clock chain is marked with a clock chain segment attribute (o) and a scan segment attribute (s):

```
*****
Current mode: ScanCompression_mode
*****
Number of chains: 16
Scan methodology: full_scan
Scan style: multiplexed_flip_flop
Clock domain: no_mix
  (l) shows cell scan-out drives a lockup latch
  (s) shows cell is a scan segment
  (m) shows cell scan-out drives a multi-mode multiplexer
  (o) shows cell is a clock chain segment
  (w) shows cell scan-out drives a wire
Scan chain '1' contains 7 cells
Active in modes: ScanCompression_mode :
snps_clk_chain_0/clock_chain (s) (o) (UPLL/CLKO, 55.0, falling)
  Z1_reg[0] (UPLL/CLKO, 45.0, rising)
  Z1_reg[1]
  Z1_reg[2] (m)
```

Defining External Clock Chains

External clock chains are uncompressed and exist outside the codec as shown in [Figure 29](#).

Figure 29 External Clock Chain in a Compressed Scan Design



External clock chains are normally used only with the following features:

To manually define the complete external clock chain, use the `set_scan_path` command. This method allows you to use specific scan-in and scan-out signals for the clock chain.

For example,

```
set_dft_signal -view spec -type ScanDataIn -port OCC_SI
set_dft_signal -view spec -type ScanDataOut -port OCC_SO
set_scan_path \
  MY_clock_chain -class occ \
  -include_elements {BLK1/OCC} \
  -scan_data_in OCC_SI -scan_data_out OCC_SO \
  -test_mode all
```

The `-class occ` option indicates that the scan path specification defines a clock chain. Use the `-include_elements` option to allow the tool to change the element order, or use the `-ordered_elements` option to use only your specified order. The `-test_mode all` option must be specified.

You can define multiple external clock chains, if needed.

If you are using DFT partitions, all clock chains to be concatenated must belong to the same partition. See [SolvNetPlus article 2675107](#), “Concatenating OCC Clock Chains From Multiple DFT Partitions.”

For more information on configuring clock chains, see [Configuring the Clock Chain](#).

5

DFTMAX Ultra Compression

DFTMAX Ultra compression uses a shift-register scan-data architecture and a single scan clock to deliver very high compression without restriction on the number of I/O pins. The input shift register feeds the decompression logic that provides data to many internal scan chains. The output shift register compresses the scan-out data using XOR logic. This architecture delivers high scan compression levels while providing a scan-compatible interface that retains the simplicity of a basic scan design.

The following topics describe the DFTMAX Ultra compression architecture:

- [DFTMAX Ultra Compression Architecture](#)
- [Configuring DFTMAX Ultra Compression](#)
- [Using OCC Controllers With DFTMAX Ultra Compression](#)

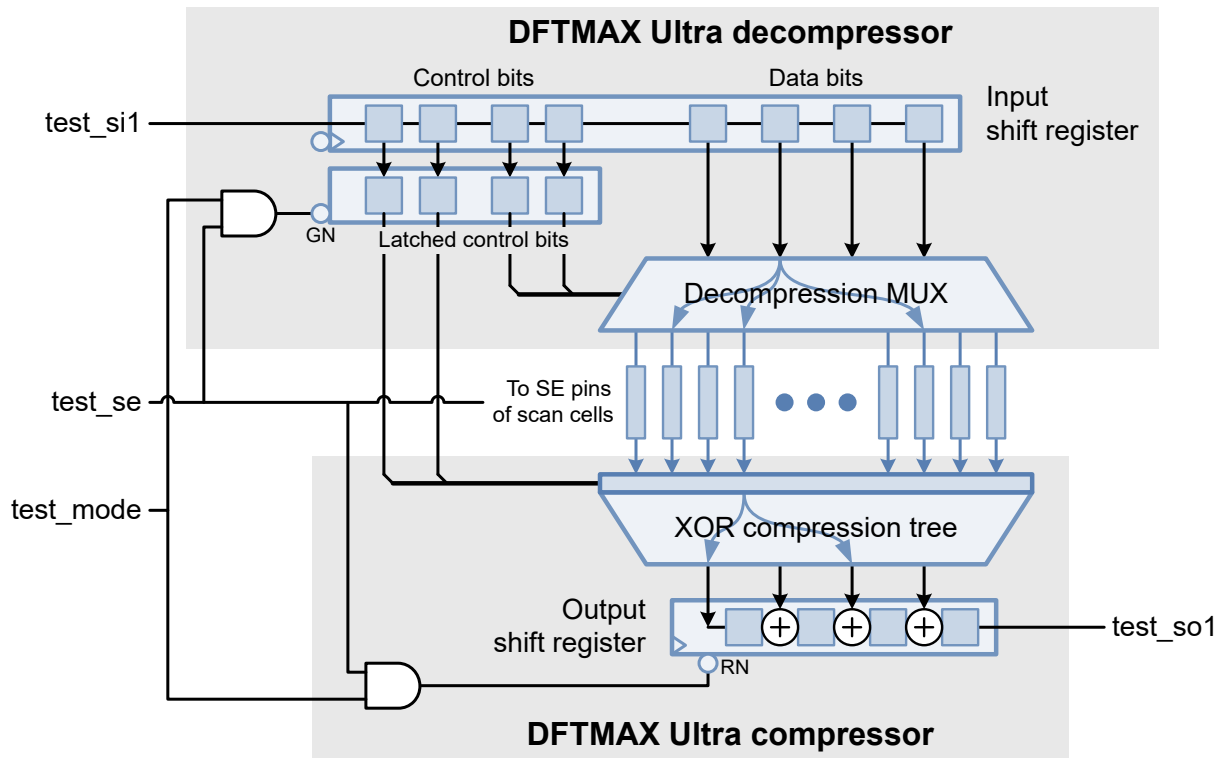
DFTMAX Ultra Compression Architecture

DFTMAX Ultra compression supports high levels of compression, high fault coverage, short scan chains, and low pin count. The scan architecture uses shift-register structures to feed in and read out the scan data streams, which enables high shift frequencies. All scan circuits operate at the same frequency; no codec clock controller circuit is needed for scan operations.

To insert DFTMAX Ultra scan compression, you specify the number of scan inputs and scan outputs, and the target number of compressed scan chains. The tool then implements an architecture based on your configuration. The DFTMAX Ultra architecture supports high compression levels using as few as one scan input and one scan output, even for a large number of internal scan chains.

[Figure 30](#) shows a block diagram for a single-input, single-output DFTMAX Ultra codec. Clock and control signals are active-high. The codec logic uses an existing scan clock; for simplicity, clock connections are not shown.

Figure 30 DFTMAX Ultra Compression Architecture, Single Scan-In and Scan-Out Pins



The features and function of the DFTMAX Ultra architecture are covered in the following topics:

- [Input Shift Register and Decompression MUX](#)
- [Control Register](#)
- [Output XOR Compression Tree and Shift Register](#)
- [Test Pattern Scan Procedure](#)
- [Scan-Enable Signal Requirements for Codec Operation](#)
- [Multiple-Input, Multiple-Output Architecture](#)

Input Shift Register and Decompression MUX

The input decompressor circuit uses a shift register and a decompression multiplexer (MUX). The input scan data at the scan-in pin feeds into the shift register, which is clocked on the trailing clock edge at the normal scan clock rate. In this implementation example, the register has eight bits. The four register bits farthest from the scan-in pin feed into the decompression MUX.

The decompression MUX is a combinational logic block that causes each of the four scan data bits to fan out to multiple scan chains. The mapping of the four bits from the shift register to the scan chains remains constant for a particular pattern. However, the mapping can change from one pattern to the next. The mapping is controlled by bits in the control register. (For more details, see the next section, [Control Register](#).)

The shift-register structure that feeds into the decompression MUX causes the input data to stream into the scan chains multiple times. Therefore, the timing is shifted by one clock cycle from successive register bits in the shift register. This architecture allows the TestMAX ATPG tool to disperse the input data stream both in space and time – in space by fanning out to multiple chains under the control of the decompression MUX, and in time by controlling the stream of bits feeding the shift register. Although a single data stream enters the design, different chains receive different data by this time shifting.

The four last-arriving bits in the shift register are latched into the control register when the scan enable signal, `test_se`, is de-asserted. This de-assertion occurs exactly once per pattern. Other than this once-per-pattern latching function, the input shift register bits feeding the control register operate only as a time-delay pipeline for the input data stream.

Control Register

The control register is a bank of latch cells that stores the configuration of the scan circuit for a given pattern. Some of the register bits control the mapping of input shift-register bits to scan chains through the decompression MUX, while others control the X-masking logic at the ends of the scan chains. In this example, two bits control the decompression MUX and two bits control the X-masking logic. The control register latches remain constant during scan shifting, so the scan configuration stays the same within a given pattern.

To program the control register, TestMAX ATPG appends the desired string of control bits to the end of the previous pattern's data stream. When scan-in completes, the control bits occupy the register positions that feed into the control register. The de-asserted scan-enable signal, `test_se`, latches these bits into the control register. Thus, the final bits of the scan-in pattern control X-masking for the current pattern to be scanned out and the decompression MUX mapping for the next pattern to be scanned in.

Output XOR Compression Tree and Shift Register

The output compressor circuit uses a combinational XOR compression tree and a sequential XOR output shift register.

The XOR compression tree is a multilevel combinational network of XOR gates that compresses the output bits from the scan chains into a smaller number of bits. Each scan chain feeds into multiple XOR tree outputs; this redundant logic helps to minimize the propagation of X values. In this example, the scan chain outputs are compressed into four bits that feed the output shift register.

If TestMAX ATPG determines that there are too many X values for the redundant XOR tree to isolate the X values, it invokes X-masking to block one or more scan chains during scan-out. The X-masking bits from the control register specify the chain or chains to mask for the current pattern and also specify the order of the signals feeding into the XOR compression logic.

The compressed bits feed into a chain of flip-flops that operate as an output shift register, which is clocked on the leading clock edge at the normal scan clock rate. During scan capture, the register is reset by the scan-enable signal. During scan shift, the XOR gate between each stage of the shift register merges scan data from an XOR compressor output into the scan data already moving through the output shift register. This architecture further compresses the scan data outputs from the XOR compression tree into a single data stream.

Test Pattern Scan Procedure

The ATE equipment performs the scan-in, scan-out procedure as specified by TestMAX ATPG. In this example, the scan procedure uses 15 extra clock cycles to flush out the extra bits from the input and output shift registers. For example, if the longest scan chain is 10,000 bits long, then the scan-in, scan-out procedure takes 10,015 scan clock cycles.

Consider two consecutive test patterns, 1 and 2, starting from the point at which pattern 1 has just been scanned in:

1. The first four bits of the input shift register (shifted in at the end of pattern 1) contain the desired scan control bits to be latched into the control register. The output XOR shift register contains leftover data from the previous pattern.
2. The scan-enable signal `test_se` changes from high to low, transitioning the device from scan shift mode to scan capture mode. The de-assertion of the `test_se` pin performs the following:
 - It latches the four control bits from the input shift register into the control register latches. This configures the X-masking circuit to scan out the data from pattern 1, and it configures the decompression MUX to decompress the data for incoming pattern 2.
 - It resets the output XOR shift register to known zero values.
3. The ATE equipment applies the test vector to the primary inputs of the device and reads the output vector from the primary outputs.
4. A clock pulse applied to the clock input causes the capture event, which changes the contents of the scan flip-flops.
5. The scan-enable signal `test_se` is asserted, which transitions the device from scan capture mode back into scan shift mode.

6. The ATE equipment applies a sequence of clock pulses at the scan clock rate. This reads out the captured scan data for pattern 1 through the test_so1 output and, at the same time, scans in the data for pattern 2 through the test_si1 input.
7. Scan-in and scan-out continue until the scan chains are filled with the data for pattern 2 (and the first four bits of the input shift register are filled with the control bits for the next pattern).

This same sequence is repeated for each pattern until the device is fully tested.

Before the initial scan-in of test pattern data, the MUX control bits of the control register must be programmed with values for proper decompression of the first test pattern. Therefore, the first test cycle uses an abbreviated padding pattern that contains only the MUX control bits and no actual scan data. For more information about padding patterns, see “Optimizing Padding Patterns” in TestMAX ATPG and TestMAX Diagnosis Online Help.

Scan-Enable Signal Requirements for Codec Operation

When the streaming codec scan-enable signal is de-asserted, the control register latches new control bit values, and the output shift register resets to a known state. Therefore, for proper operation, *this scan-enable signal must be held in the inactive state in all capture procedures.*

If you use the STIL protocol file created by the tool, the protocol already meets this requirement. In the capture procedures, the tool constrains all scan-enable signals that drive streaming codecs to the inactive state.

In some flows, streaming codecs cannot use signals defined with the `-usage` option of the `set_dft_signal` command.

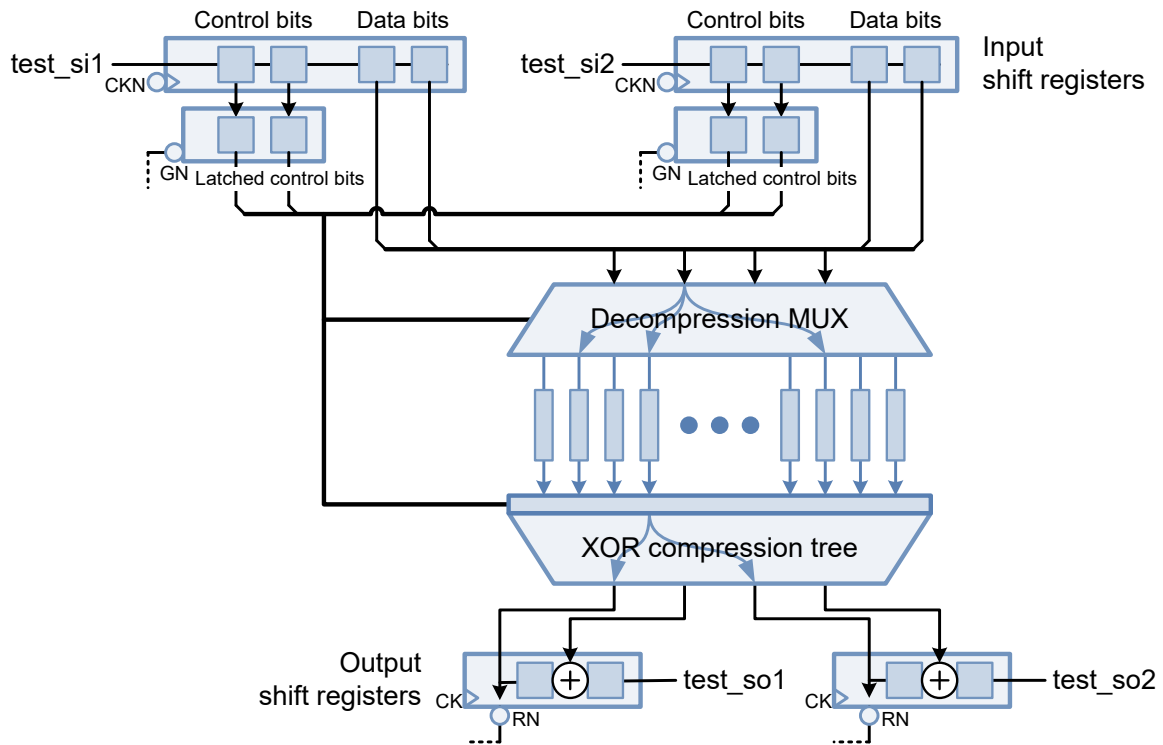
If you use a custom STIL protocol file, make sure that all scan-enable signals used by DFTMAX Ultra codecs are constrained to the inactive state in all capture procedures.

Multiple-Input, Multiple-Output Architecture

If the codec is configured to use multiple scan-in and scan-out connections, the tool synthesizes the scan circuitry in a manner similar to the single-pin circuit, but it splits the input and output shift registers into smaller segments and connects them to the available input and output pins. By using more scan I/O pins, you get shorter shift registers, improved controllability, and improved observability into the design.

Figure 31 shows a DFTMAX Ultra codec with two scan data inputs and two scan data outputs. The codec has four input shift-register scan data bits feeding into the decompression MUX, four control register bits that control the decompression and compression logic, and four output shift-register bits.

Figure 31 DFTMAX Ultra Compression Architecture, Multiple Scan-In and Scan-Out Pins



The tool determines the total input and output shift register lengths based on the number of compressed scan chains, then it then splits these shift register lengths across the scan inputs and outputs (rounding up shorter registers as needed). Therefore, as you increase the number of scan inputs and outputs, the scan shift overhead of the shift registers is reduced.

This bit distribution is more flexible than the single-input, single-output design because there are multiple independent data streams rather than one. This flexibility might allow the same fault coverage to be achieved with fewer patterns, but at the cost of using more device pins.

In this example, the distribution of bits in the input shift registers allows TestMAX ATPG to generate two independent data streams at the same time, one each for test_si1 and test_si2. Each input shift register provides its own scan data bits for multiplexing to the scan chains. For each compressed scan chain, TestMAX ATPG has a choice of up to four different bit streams: two from test_si1 and two from test_si2.

The control register bits are the last data bits shifted into the device for a pattern. Therefore, the bits of the shift register used for loading the control register are located closest to the input pin, whereas the bits of the shift register that are available to the decompression MUX are located farthest from the input pin.

On the output side, the output XOR shift register is divided into two smaller shift registers, one each for the output pins test_so1 and test_so2. This reduces the amount of data compression performed in the shift register, which reduces the propagation of X values and provides greater observability into the design.

Configuring DFTMAX Ultra Compression

To enable and configure DFTMAX Ultra compression, do the following:

1. Enable compression with the following command:

```
fc_shell> set_dft_configuration -scan_compression enable
```

2. Configure streaming compression by using the following command:

```
set_scan_compression_configuration  
-streaming true  
-inputs num_scanins  
-outputs num_scanouts  
-chain_count num_chains  
  | -max_length chain_length  
-clock codec_clock  
[-test_mode mode_name]
```

The `-streaming` option indicates a DFTMAX Ultra streaming compression specification.

The `-inputs` and `-outputs` options are required. There must be enough scan-in and scan-out signals defined to satisfy the specification.

The `-chain_count` and `-max_length` options are mutually exclusive; one is required.

The `-clock` option specifies which clock to use for the streaming codec. By default, the tool selects clocks for the decompressor and compressor that minimize the number of lock-up latches at the decompressor outputs and compressor inputs. The selection rules are as follows:

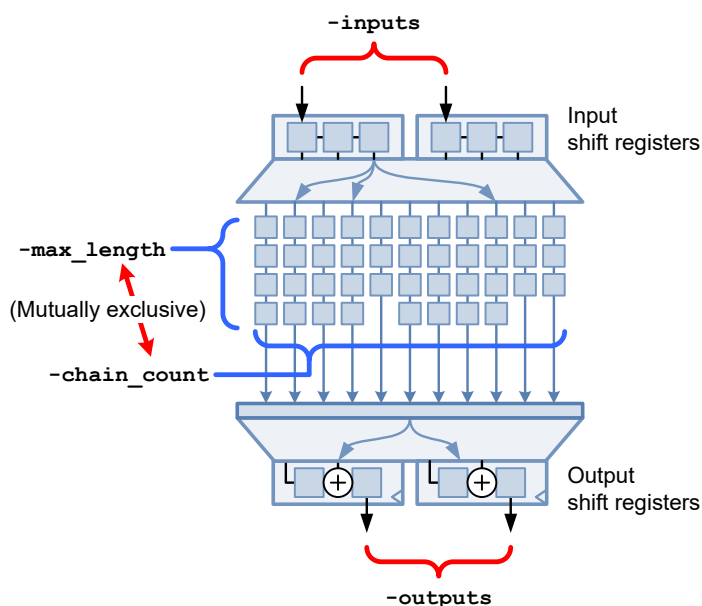
- The decompressor uses a clock whose trailing edge is as late or later than all head scan elements.
- The compressor uses a clock whose leading edge is as early or earlier than all tail scan elements.
- If multiple clocks meet these criteria for the decompressor or compressor, the dominant clock across the head or tail scan elements is used, respectively.

By default, the command creates and configures a mode named `ScanCompression_mode`. If you are using user-defined test modes, you specify the mode to configure with the `-test_mode` option.

3. Prevent QN pins of registers from being used by the DFTMAX Ultra IP:

```
fc_shell> set_app_options -as_user_default \  
-list {compile.seqmap.enable_output_inversion false}
```

The configuration options apply to the compression architecture as follows:



DFTMAX Ultra compression automatically computes the input and output shift register lengths that are optimal for the number of scan inputs and outputs and the number of compressed scan chains.

If you are also implementing a standard scan mode, keep in mind that these compressed chains become the base chains used to create other modes.

Using OCC Controllers With DFTMAX Ultra Compression

DFTMAX Ultra requires that the clock chains of OCC controllers be uncompressed so that the control bits are directly controllable without conflict by ATPG. Otherwise, significant pattern inflation will result.

To define an external clock chain, use the `-scan_data_in` and `-scan_data_out` options of the `set_scan_path` to specify the input and output ports of the external chain:

```
set_dft_clock_controller -cell_name BLK/OCC ...  
set_scan_path OCC_CHAIN -class occ \  
  -include_elements {BLK/OCC} \  
  -scan_data_in SI_OCC -scan_data_out SO_OCC
```

For details on defining external clock chains, see [Configuring the Clock Chain](#).

6

DFT Partitions

In some cases, you might want to apply different DFT configuration to different parts of the design. For example, you might want to use different scan-enable signals for different blocks, or you might want to enable clock-mixing for some blocks but not others.

You can use *DFT partitions* to do this. They allow you to divide up your design logic into multiple partitions, then you apply DFT configuration commands to each partition.

The following topics describe how to use DFT partitions:

- [Defining DFT Partitions](#)
- [Configuring DFT Partitions](#)
- [Per-Partition Scan Configuration Commands](#)
- [Known Issues of the DFT Partition Flow](#)

Defining DFT Partitions

To use DFT partitions, you must first define them. You can use the following commands to define and manage partition definitions in your top-level run:

- `define_dft_partition`
- `report_dft_partition`
- `remove_dft_partition`

You can use the `define_dft_partition` command to define a partition. The most commonly used options are:

```
define_dft_partition
  partition_name
  [-include list_of_cells_or_references]
  [-clocks list_of_clocks]
```

You must provide a unique name for each partition definition. This name is used to reference the partition when providing codec information, as well as to identify the partition in subsequent DFT reports.

A partition definition can include design references, hierarchical cells, scan cells, or clock domains. Although partitions are usually defined along physical or logical hierarchy boundaries, it is not a requirement.

To specify a set of cells, design references, or core scan segments, use the `-include` option. Leaf cells can be specified, although only sequential cells are relevant to the partition definition. Design references are converted to the set of all hierarchical instances of those designs. A particular cell or design reference can exist in only one partition definition. In [Example 2](#), two partitions are defined using hierarchical cells, and a third partition is defined using a design reference.

Example 2 Defining Three Partitions Using Cells and References

```
define_dft_partition P1 -include [get_cells {U_SMALL_BLK1 U_SMALL_BLK2}]
define_dft_partition P2 -include [get_cells {U_BIG_BLK}]
define_dft_partition P3 -include [get_references {my_ip_design}]
define_dft_partition P1 -clocks {CLK1 CLK2}
define_dft_partition P2 -clocks {CLK3}
```

The tool creates a default partition named `default_partition` that includes the flip-flops not included in any user-defined partitions. You cannot use the name `default_partition` when defining a partition.

You can use the `report_dft -partitions` command to see what partitions have been defined. [Example 3](#) shows the output for the three partitions previously defined in [Example 2](#). Note that the design reference has been converted to the corresponding set of hierarchical instances of that design.

Example 3 Example of Output From the `report_dft -partitions` Command

Cell Name	Partition Name
U_SMALL_BLK1/	P1
U_SMALL_BLK2/	P1
U_BIG_BLK/	P2
U_MY_IP_BLK/	P3

Configuring DFT Partitions

After the DFT partitions are defined, you can configure the scan configuration for each partition. Use the `current_dft_partition` command to set the current partition, then apply one or more supported test configuration commands to configure scan for that partition.

All DFT partitions share a common global configuration. Partition-specific configuration commands are applied incrementally on top of the global configuration.

In a DFT partition flow, the sequence of configuration commands is:

- Apply global DFT configuration settings
- Define DFT partitions with `define_dft_partition`
- Apply partition-specific DFT configuration settings to each partition with `current_dft_partition`

[Example 4](#) shows an example of global and partition-specific configuration commands.

Example 4 *Configuring Two DFT Partitions*

```
# apply global DFT configuration settings
set_scan_configuration -clock_mixing mix_clocks
set_dft_signal -view existing_dft \
    -type ScanClock -timing [list 45 55] -port CLK
set_dft_signal -view spec -type TestMode -port TM

# define DFT partitions
define_dft_partition P1 -include {BLK1}
define_dft_partition P2 -include {BLK2}

# configure DFT partition P1
current_dft_partition P1
set_dft_signal -view spec -type ScanEnable -port SE1
set_dft_signal -view spec -type ScanDataIn -port {SI1 SI2}
set_dft_signal -view spec -type ScanDataOut -port {SO1 SO2}
set_scan_configuration -chain_count 2

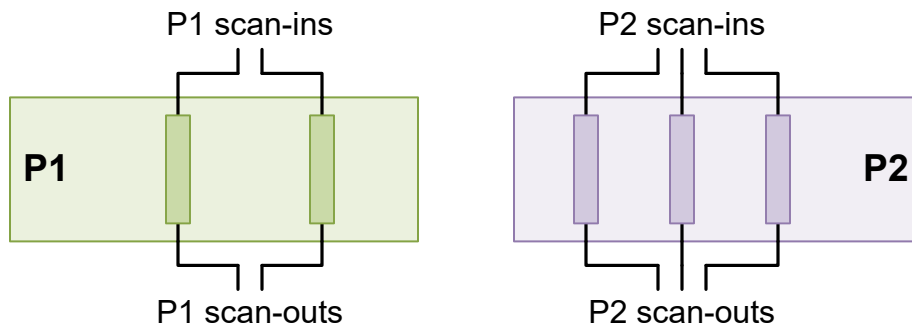
# configure DFT partition P2
current_dft_partition P2
set_dft_signal -view spec -type ScanEnable -port SE2
set_dft_signal -view spec -type ScanDataIn -port {SI3 SI4 SI5}
set_dft_signal -view spec -type ScanDataOut -port {SO3 SO4 SO5}
set_scan_configuration -chain_count 3
```

If you are defining scan-in or scan-out signals using the `set_dft_signal` command, you must define them as a part of each partition's scan configuration. Scan-enable signals can be defined globally or on a per-partition basis. However, if a scan-enable signal is defined for only one partition, it is automatically applied to the remaining partitions. Other signal types, such as reset, clock, and test-mode signals, must be defined globally before any partitions are defined, or as part of the default partition configuration.

For the entire design, the total scan chain count is the sum of the scan chain counts across all partitions. Each scan chain requires its own scan-in and scan-out pin pair, just as a scan chain does in an unpartitioned flow. Scan chains are not combined or rebalanced across the partitions.

[Figure 32](#) shows the scan mode chain connections for [Example 4](#). A total of five scan-in and scan-out pins are used, two for partition P1 and three for partition P2.

Figure 32 DFT Partition Scan Chains



After configuring the partitions, you can use the `preview_dft` command to display the scan chain distribution across partitions. [Example 5](#) shows a report example.

Example 5 Output From the `preview_dft` Command for Multiple DFT Partitions

```
*****
Current mode: Internal_scan
*****
Number of chains: 5
Scan methodology: full_scan
Scan style: multiplexed_flip_flop
Clock domain: no_mix
Scan chain '1' (SI1 --> SO1) contains 32 cells (Partition 'P1')
Active in modes: Internal_scan
Scan chain '2' (SI2 --> SO2) contains 32 cells (Partition 'P1')
Active in modes: Internal_scan
Scan chain '3' (SI3 --> SO3) contains 22 cells (Partition 'P2')
Active in modes: Internal_scan
Scan chain '4' (SI4 --> SO4) contains 21 cells (Partition 'P2')
Active in modes: Internal_scan
Scan chain '5' (SI5 --> SO5) contains 21 cells (Partition 'P2')
Active in modes: Internal_scan
```

Per-Partition Scan Configuration Commands

This topic lists the commands you can use to configure DFT insertion on a per-partition basis. Commands not listed in this section should be applied as part of the global DFT configuration.

set_scan_configuration

The following `set_scan_configuration` options can be specified on a per-partition basis:

- `-chain_count`
- `-max_length`
- `-clock_mixing`
- `-insert_terminal_lockup`
- `-test_mode`
- `-exclude_elements`

set_dft_signal

The following `set_dft_signal` options can be specified on a per-partition basis:

- `-view`
- `-type ScanDataIn | ScanDataOut | ScanEnable`
- `-port`
- `-hookup_pin`
- `-hookup_sense`
- `-active_state`

For other test logic types, you can only specify the location for the default partition.

set_scan_path

The following `set_scan_path` options can be specified on a per-partition basis:

- `-include_elements`
- `-ordered_elements`
- `-complete`
- `-scan_enable`
- `-scan_data_in`
- `-scan_data_out`

set_wrapper_configuration

The following `set_wrapper_configuration` options can be specified on a per-partition basis:

- `-chain_count`
- `-mix_cells`

Known Issues of the DFT Partition Flow

The following known issues apply to the partition flow:

- If you define a scan-enable signal for a partition, that scan-enable signal is reused for other partitions for which the signal has not been defined. This reuse avoids having to create new scan-enable signals.
- The `define_dft_partition` command does not perform duplicate object checking between cell instances and design modules. The overlapping objects will be included only in one of the partitions.
- If you apply commands or options that do not support per-partition specification to a DFT partition, they are ignored with no warning.

7

Pipelined Scan Data

You can use the pipelined scan data feature to resolve delay problems associated with long routes in compressed scan chain logic.

The following topics provide more detail:

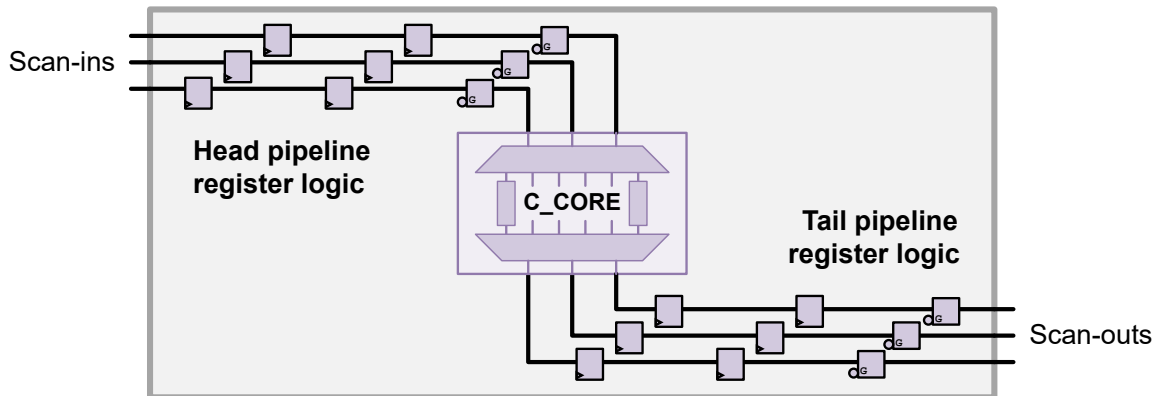
- [Introduction to Pipelined Scan Data](#)
- [The Pipelined Scan Data Logic Structure](#)
- [Configuring Pipelined Scan Data](#)
- [Pipelined Scan Data Test Protocol Format](#)
- [Pipelined Scan Data Limitations](#)

Introduction to Pipelined Scan Data

In typical scan flows, long wires between the scan chain input and the first flip-flop and between the last flip-flop and the scan chain output can cause delay problems. Scan compression logic and higher scan frequencies further amplify the problem. These delays are reduced by placing pipeline registers at the beginning and end of the scan chains. They divide the long routes between the scan chain terminals into smaller wires between the registers and therefore help reduce the path delay.

[Figure 33](#) shows an example of a compressed scan design with pipeline registers. The head pipeline registers are placed between the scan inputs and the decompressor, and the tail pipeline registers are placed between the compressor and the scan outputs.

Figure 33 Head and Tail Pipeline Register Logic



RTL DFT insertion adds these pipeline registers to the scan paths. Synthesis places them along the scan paths using physical information. Post-DFT DRC verifies their correct operation.

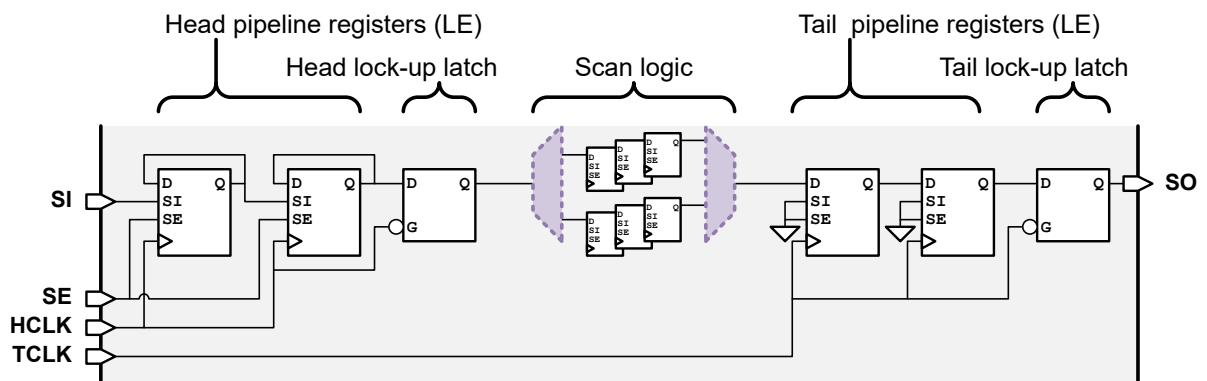
The Pipelined Scan Data Logic Structure

The head pipeline registers are implemented with state-holding scan cells. This prevents unknown X values from being captured, which would then get propagated through the decompressor into the compressed chains.

The tail pipeline registers are implemented using nonscan cells for minimal area. (There is no state-holding requirement.)

At the end of each head or tail pipeline register sequence, a lockup latch is inserted to provide additional timing margin. The head lockup latches exist outside the codec, and thus serve all fanout registers (through reconfiguration MUXs, decompressor logic, etc.). The tail lockup latches drive the scan-out signals. These lockup latches are always inserted, even when not crossing clock domains.

Figure 34 Pipeline Registers (Depth == 2) in a Compressed Scan Design



Post-DFT DRC verifies whether the head pipeline registers hold their values during capture and issues an R-18 violation message if the check is unsuccessful.

Configuring Pipelined Scan Data

To configure pipelined scan data, do the following:

1. Enable the pipelined scan data feature:

```
fc_shell> set_dft_configuration -pipeline_scan_data enable
```

2. Specify the head and tail clock and pipeline depths using the following options:

```
fc_shell> set_pipeline_scan_data_configuration \  
-head_pipeline_clock clock_name \  
-tail_pipeline_clock clock_name \  
-head_pipeline_stages integer \  
-tail_pipeline_stages integer
```

All options are required.

After RTL DFT insertion, the newly inserted pipeline registers and lock-up latches have names of the form

```
DFT_block/U_dft_head_p/pipe_stage_reg[signum]  
DFT_block/U_dft_head_p/lul_out_reg[signum]  
...  
DFT_block/U_dft_tail_p/pipe_stage_reg[signum]  
DFT_block/U_dft_head_p/lul_out_reg[signum]
```

where *DFT_block* is the DFT logic block name, *stage* is the stage depth, and *signum* is the ordinal number of the scan-in or scan-out signal.

Use the `report_dft -pipeline` command to report the current pipelined scan data configuration.

Pipelined Scan Data Test Protocol Format

The test protocol file generated for uncompressed scan modes does not contain any pipeline register information, as it is not needed for these modes.

The test protocol file generated for compressed scan modes does contain information about the pipeline registers. The test protocol file has the following additional information:

- The number of head pipeline stages is indicated by the `LoadPipelineStages` keyword.
- The number of tail pipeline stages is indicated by the `UnloadPipelineStages` keyword.

For example,

```
CompressorStructures {
  LoadPipelineStages 3;
  UnloadPipelineStages 2;
  Compressor des2_U_decompressor {
    ModeGroup mode_group;
    LoadGroup load_group;
    CoreGroup core_group;
    Modes 3;
    ...}
  Compressor des2_U_compressor {
    UnloadGroup unload_group;
    CoreGroup core_group;
    Mode {{...}}
  }
}
```

Pipelined Scan Data Limitations

These are the requirements and limitations for implementing pipeline registers:

- The tool does not insert pipeline scan data registers to external scan chains created by the `set_scan_path` command. However, external clock chains are pipelined.
- Return-to-one clocks are not supported. (The pipeline registers are not inserted with the correct polarity.)
- Designs without scan compression are not supported.
- When scan clocks of differing timing waveforms are used, the timing relationship between the head pipeline registers, scan chains, and tail pipeline registers is not checked and must be correct by construction.
- If you are using external (uncompressed) chains, such as external clock chains or other user-defined external chains, their pipeline depths must match other scan chains. For more information, see [Excluding Scan Chains From Scan Compression](#).
- When using internal pins for the scan-in and scan-out signals, any combinational logic between the scan ports and the pipeline registers must be sensitized to a known state.
- An ATE clock defined with the `set_dft_signal -type Oscillator` command cannot be used to directly clock either the head or tail pipelined scan data registers. If the DFT-inserted OCC controller flow is used together with pipeline register insertion, specification of the ATE clock as the pipeline clock results in the OCC-controlled clock being used by default. If the ATE clock is manually connected to the pipeline registers, no violations are reported, but incorrect ATPG patterns might be generated.

8

Pipelined Scan-Enable Signals

You can use pipelined scan-enable signals to provide launch-on-extra-shift (LOES) transition-delay timing in TestMAX ATPG, which improves ATPG efficiency and reduces pattern count.

This is described in the following topics:

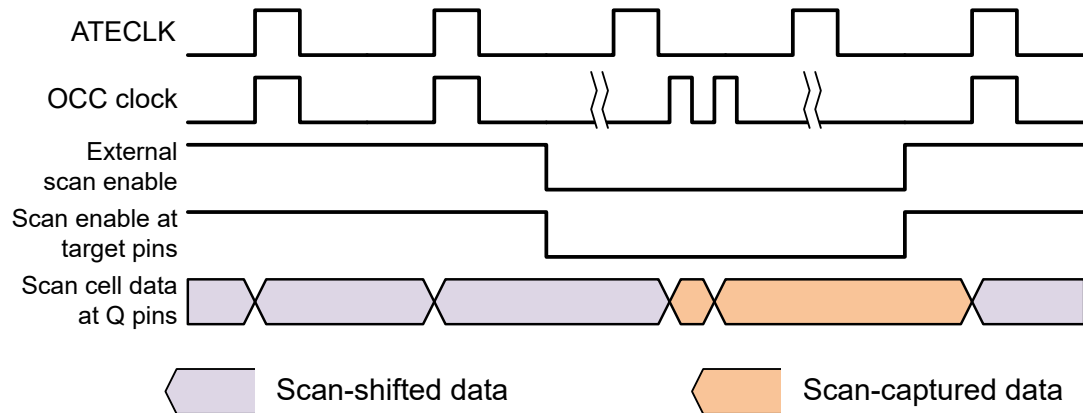
- [The Pipelined Scan-Enable Architecture](#)
- [Pipelined Scan-Enable Requirements](#)
- [Excluding Elements From Pipelined Scan-Enable Configurations](#)
- [Implementing Pipelined Scan-Enable Signals](#)
- [Reporting the Pipelined Scan-Enable Clusters](#)
- [Implementation Considerations for Pipelined Scan-Enable Signals](#)
- [Pipelined Scan Enable Limitations](#)
- [PSE max Fanout per Clock Specification](#)

The Pipelined Scan-Enable Architecture

Transition-delay fault testing requires two at-speed clock cycles in a row—one to launch and one to capture—so that the launched data must propagate through the logic cones at-speed to be captured properly. These at-speed clock pulses are typically provided by an on-chip clocking (OCC) controller, as described in [Chapter 9, Core Wrapping](#).

By default, Fusion Compiler implements a simple scan-enable signal, which is externally controlled at test clock frequencies. This requires the use of Launch-on-Capture (LOC) transition-delay timing, shown in [Figure 35](#), where scan enable is de-asserted *before* both the launch and capture clock pulses.

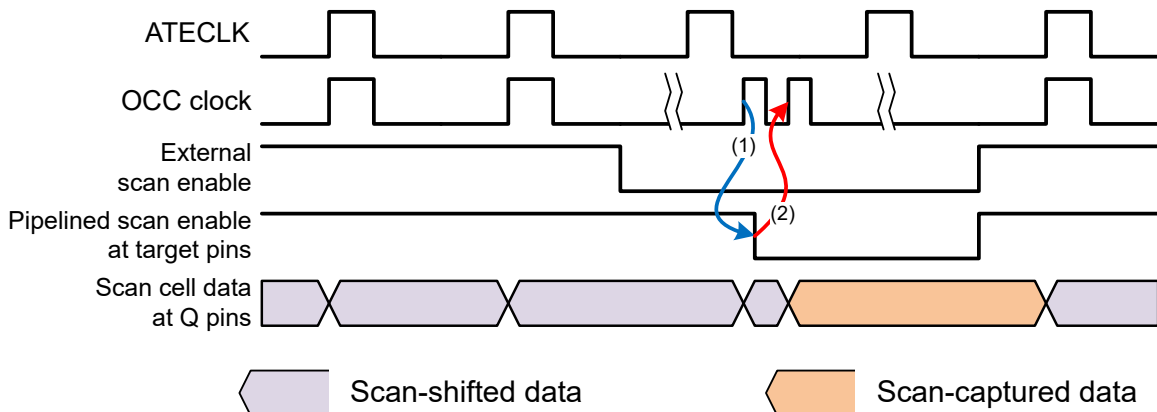
Figure 35 Launch-on-Capture (LOC) Transition-Delay Timing



Because scan enable is de-asserted for the launch clock pulse, the launch data must be controlled indirectly (captured through the logic cones) by the data from the previous scan-shift cycle. This requires fast-sequential ATPG, which imposes additional constraints and overhead on ATPG.

To avoid this limitation, you can implement *pipelined scan-enable signals*, which can generate the scan-enable transition at-speed. This allows launch-on-extra-shift (LOES) transition-delay timing, as shown in Figure 36, where TestMAX ATPG directly controls the launch data.

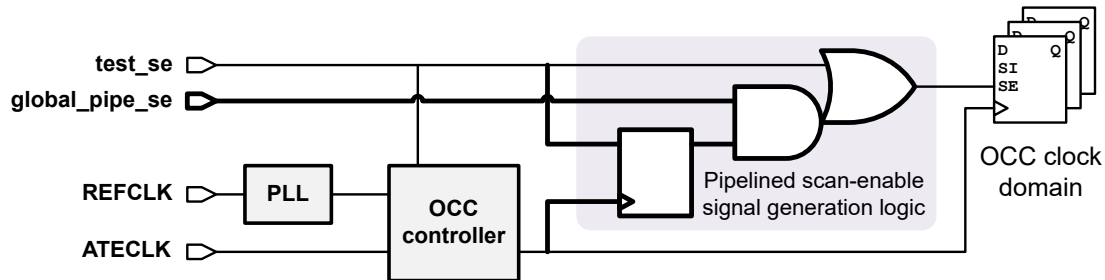
Figure 36 Launch-On-Extra-Shift (LOES) Transition-Delay Timing



The waveforms for the external signals are identical between LOC and LOES, but the scan-enable signal that reaches the scan cells is different.

The pipelined scan-enable feature works by creating a registered scan-enable transition on the chip, just as an OCC controller generates controlled at-speed clock pulses on the chip. Figure 37 shows the pipelined scan-enable logic structure.

Figure 37 Pipelined Scan-Enable Logic



The global_pipe_se signal controls whether the output scan-enable signal operates in nonpipelined or pipelined mode:

- When the global_pipe_se signal is de-asserted, the register is bypassed and the output scan-enable signal is a simple nonpipelined scan-enable signal.
- When the global_pipe_se signal is asserted,
 - When the input scan-enable signal is asserted, the output scan-enable signal is immediately asserted.
 - When the input scan-enable signal is de-asserted, the register holds the output scan-enable asserted until the next leading clock edge of the *at-speed* clock.

Each clock domain and clock edge has its own pipelined scan-enable logic construct that de-asserts the scan-enable signal synchronized to that clock edge.

The pipelined scan-enable feature and the pipelined scan data feature are independent and unrelated. Either can be used separately, or they can be used together, but the commands, limitations, and messages are specific to one or the other and must not be confused.

See also “Transition-Delay Fault ATPG Timing Modes” in TestMAX ATPG and TestMAX Diagnosis Online Help for more information about launch-on-capture (LOC) and launch-on-extra-shift (LOES) timing.

Pipelined Scan-Enable Requirements

Launch-on-extra-shift (LOES) with pipelined scan-enable signals has the following requirements:

- All scan-enable signals must have the same pipelined scan-enable depth after DFT insertion. Valid depths are zero (not pipelined) and one (pipelined).
- Each clock domain and clock edge must have its own pipelined scan-enable signal, which all scan flip-flops clocked by that domain and edge must use.

- Clock-gating cells should also use the pipelined scan enable for clock domains and edges.

It is possible to use LOES when clock-gating cells use the nonpipelined scan enable. However, you should use the pipelined scan enable because it maximizes the amount of data that can be launched in the extra shift, which usually results in a smaller pattern set. It also allows the use of the legacy ATPG method Launch-On-Last-Shift, where the last scan shift doubles as the transition launch cycle (which might be important in the case of design reuse).

- The nonpipelined scan enable must be used for OCC controllers. This allows the OCC controller to generate the extra-shift launch clock followed by the capture clock as an at-speed clock pair.
- The nonpipelined scan enable must be used for clock chains used by OCC controllers. This holds the clock chain data steady during capture. If the pipelined scan enable is used, the clock chain bits are corrupted by the launch clock.
- When clock mixing is used, lock-up latches might be ineffective because the clock timing in the extra shift uses the launch waveform table rather than shift timing. As a result, DRC in the TestMAX ATPG tool marks the first flip-flop following the clock domain crossing as disturbed, which might reduce coverage slightly (although an incremental LOC run can detect these faults).
- All on-chip scan-enable signals (nonpipelined and pipelined for each clock edge) must be derived from a single scan-enable signal source.

Excluding Elements From Pipelined Scan-Enable Configurations

When you enable the pipelined scan enable feature, all scan-enable connections are subject to pipelining behavior. However, in some designs, you might want to exclude specific scan elements from pipelining. You can exclude specific elements or entire classes of elements.

When a scan-enable connection is excluded, the tool routes that connection as if the pipelined scan-enable is not enabled. In other words, the tool tries to connect directly from the port or hookup pin to the scan cell's scan-enable pin.

To exclude specific elements, use the following command:

```
fc_shell> set_pipeline_scan_enable_configuration \  
-exclude element_list
```

The argument list can contain any of the following elements:

- Instance name

The tool excludes any scan-enable connection to that cell.

Chapter 8: Pipelined Scan-Enable Signals

Excluding Elements From Pipelined Scan-Enable Configurations

- Pin

The tool excludes only that pin, which must be a scan-enable pin. If the cell is not mapped, the pin name might change when the tool maps the cell. To avoid this, exclude the entire register instead of the pin.

- Hierarchy

The tool excludes all scan-enable connections to instances contained in that hierarchy and its child hierarchies.

- Scan clock port

The tool excludes all elements that belong to that clock domain.

- Scan clock hookup port

The tool excludes all elements that belong to that clock domain.

If multiple clock pins share the same scan enable, all of the clocks must share the same configuration, either pipelined or not pipelined.

To exclude entire classes of elements, use the following command:

```
fc_shell> set_pipeline_scan_enable_configuration \  
-exclude_types category_list
```

The argument list can contain any of the following categories:

- shared_wrapper
- dedicated_wrapper
- input_wrapper
- output_wrapper
- input_shared_wrapper
- output_shared_wrapper
- input_dedicated_wrapper
- output_dedicated_wrapper
- testpoint
- clock_gate

Implementing Pipelined Scan-Enable Signals

To implement pipelined scan-enable signals, do the following:

1. Enable the pipelined scan-enable feature:

```
fc_shell> set_scan_configuration -pipeline_scan_enable true
```

2. Define the enable signal:

```
fc_shell> set_dft_signal -view spec -type LOSPipelineEnable \  
-port PSE_EN -active_state 1 -test_mode all
```

This enable signal selects launch-on-extra-shift (LOES) when asserted and Launch-on-Capture (LOC) operation when de-asserted.

You can drive the enable signal from an on-chip configuration register by using the internal pins flow. For more information, see [Chapter 11, The Internal Pins Flow](#).

3. Specify the maximum number of scan cells to be driven by each pipeline register:

```
fc_shell> set_scan_configuration -pipeline_fanout_limit 64
```

4. (Optional) Exclude specific elements from implementation:

```
fc_shell> set_pipeline_scan_enable_configuration \  
-exclude element_list
```

5. (Optional) Exclude specific categories of scan chain elements from implementation:

```
fc_shell> set_pipeline_scan_enable_configuration \  
-exclude_types category_list
```

The tool creates multiple physically compact clusters of scan cells, each with their own local pipelined scan-enable (PSE) construct. This improves the critical path delay between the pipeline registers and their scan cells.

The tool creates as many clusters as needed, each with exactly the specified number of scan cells. (The last-created cluster in each clock domain might have fewer scan cells.) The pipeline registers have the `size_only` attribute set, which prevents duplicate register merging by the `compile_fusion` command.

See [Implementation Considerations for Pipelined Scan-Enable Signals](#) for details on clustering and timing. See [SolvNetPlus article 2543967, "The Pipelined Scan-Enable Fanout Limit, Duplicate Scan-Enable Signals, and the TEST-1073 Error,"](#) for more information about how the fanout limit is applied.

The default is not to apply a fanout limit.

If you are using a DFT-inserted OCC controller, the clock connection might be incorrect. See [Pipelined Scan Enable Limitations](#) for details.

The test protocol created by the tool does not constrain the pipeline enable signal, so post-DFT DRC checks both the LOC and LOES modes. The protocol written out by the `write_test_protocol` command also does not constrain the enable signal. Use the `add_pi_constraint` command in TestMAX ATPG to assert or de-assert the pipeline enable signal. See “Using Launch-On Extra-Shift Timing” in TestMAX ATPG and TestMAX Diagnosis Online Help for more information.

Reporting the Pipelined Scan-Enable Clusters

When the pipelined scan-enable feature is enabled, the `compile_fusion` command creates a `scan_dft.rpt` file that reports the cluster information:

```
=====
Pipelining Scan Enable Report :
=====
Number of pipelining levels : 1
Number of Flops driven per stage: 48
-----
Clock/Scan Enable      Stage 1      Leaf flops/elems
-----+-----+-----
"pll_1/CLK_4X"/SE      | PSE1179 | dd_decrypt/ein_reg
                        |         | dd_decrypt/kdin_reg[107]
                        |         | dd_decrypt/kdin_reg[111]
...                     |         |
                        |         | dd_a/o_tval_reg
                        |         | dd_a/state_reg
Global pipeline scan enable signal: ctmi_1183/Y
-----
Number of pipelining levels : 1
Number of Flops driven per stage: 48
-----
Clock/Scan Enable      Stage 1      Leaf flops/elems
-----+-----+-----
"pll_1/CLK_4X"/SE      | PSE1184 | dd_decrypt/kdin_reg[100]
                        |         | dd_decrypt/kdin_reg[101]
                        |         | dd_decrypt/kdin_reg[103]
...                     |         |
                        |         | dd_b/o_data_reg[7]
                        |         | dd_b/o_data_reg[8]
Global pipeline scan enable signal: ctmi_1188/Y
-----
```

Implementation Considerations for Pipelined Scan-Enable Signals

Note the following considerations when implementing pipelined scan-enable signals.

Timing Considerations

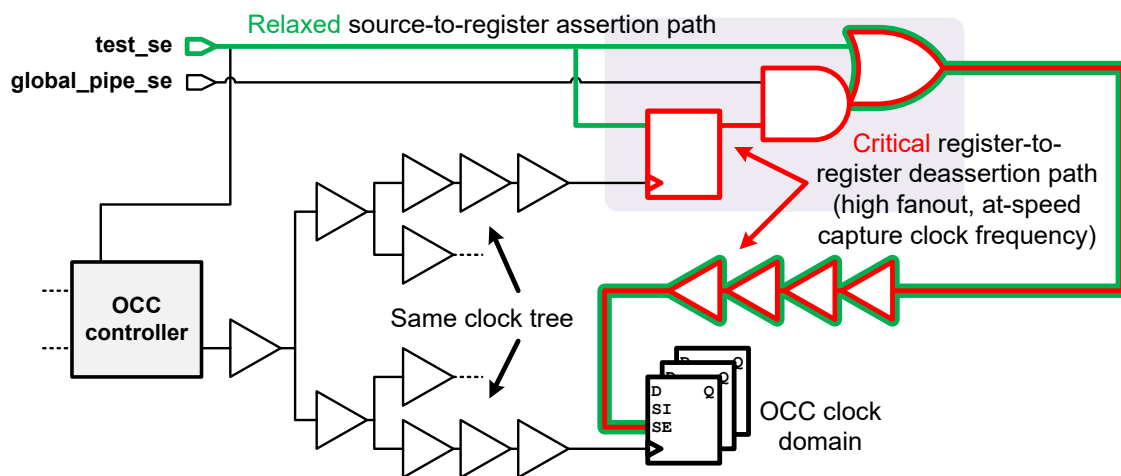
For a regular scan-enable signal, the signal path is a single-cycle path at the shift clock frequency, which is likely to be fairly slow—and even that requirement can be relaxed

by adding extra cycles in TestMAX ATPG using the `-use_delay_capture_start` and `-use_delay_capture_end` options of the `write_patterns` command.

For a pipelined scan-enable signal, this relaxed timing applies only to the external signal source. The de-assertion event from the pipeline register must reach *all* downstream flip-flop scan-enable pins in a single cycle at the capture clock frequency, which is much faster than shift. This path cannot be relaxed because a multicycle path would result in bad patterns, and slowing down the clock makes the test less-than-at-speed.

Figure 38 shows the relaxed externally driven assertion path in green and the critical register-driven de-assertion path in red.

Figure 38 Pipelined Scan-Enable Pipeline Register Timing Path



The pipeline register and its fanout scan cells are all typically driven by the same skew-balanced clock tree. As a result, the scan-enable buffer tree paths are standard register-to-register timing paths, with allowable delays between nearly zero and almost a full clock cycle. The buffer tree does not need to be skew-balanced.

In addition,

- For high-frequency clocks driving large clock domains, the buffer tree delay might be greater than the clock period. In this case, you can move the pipeline register's clock connection earlier in the clock tree, as long as hold constraints are met in all operating conditions.

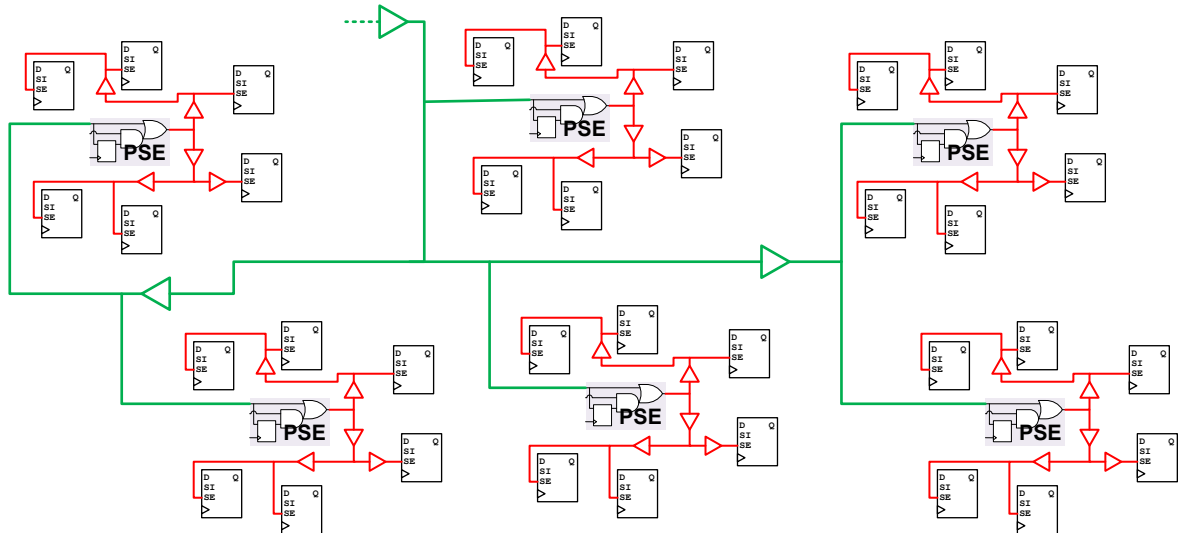
Physical Implementation Considerations

In Fusion Compiler, the tool creates physically aware buffer trees on the pipelined scan-enable nets to avoid synthesis design rule violations.

If you set a fanout limit to implement pipelined scan-enable clusters, the tool analyzes scan cell locations to create compact clusters. Then, it builds physically aware buffer trees

to all cluster pipeline registers (green) and within each cluster from the pipeline register to the scan cells (red), as shown in [Figure 39](#).

Figure 39 Pipelined Scan-Enable Signal With Fanout-Limited Scan Cell Clusters



Pipelined Scan Enable Limitations

Note the following requirements and limitations of pipelined scan-enable signals:

- Scan-enable signals defined with the `-usage scan` option of the `set_dft_signal` command are not supported. Scan-enable signals defined with the `-usage clock_gating` option are supported, and they are pipelined.
- When using DFT-inserted OCC controllers, the clock connection to the pipeline scan-enable registers might be wrong.

This usually means that the tool uses the clock coming from the uncontrolled PLL source. In this case, TestMAX ATPG will generate patterns, but those patterns will fail Verilog simulation. The correct clock connection to the pipeline scan enable register is the output of the OCC controller, at the leaf level of the clock tree. If the register is in a module with scan flip-flops that it controls, the same clock signal that drives the scan flip-flops should also drive the pipeline scan-enable register. The register instance names begin with “PSE”. There might be more than one for each clock, and they might be buried in the design hierarchy, depending on the settings.

PSE max Fanout per Clock Specification

The Pipeline Scan-Enable (PSE) logic is inserted during DFT scan insertion to provide at-speed transition of scan enable signals to allow launch-on-extra-shift (LOES) transition-delay timing.

To enable PSE max fanout per clock specification, use the `set_pipeline_scan_enable_configuration-clock_based_fanout_limit` command. You must define the new input using the `set_pipeline_scan_enable_configuration` command before calling `preview_dft` or `insert_dft`.

For example, `set_pipeline_scan_enable_configuration -clock_based_fanout_limit {clock1 100 clock2 500}`

In the preceding command, PSE groups driven by `clock1` have a max fanout limit of 100, whereas PSE groups driven by `clock2` have a fanout of 500. If other PSE groups driven by different clocks are not affected by these two values, then they must use the global PSE fanout limit, if defined.

The clocks used in the command must be defined as scan clocks. The valid scan clocks are ports or hookup pins.

For a scan clock specification with both `port` and `hookup_pin` options, the fanout limit for `hookup_pin` gets precedence if different values are specified for the options. In the following example, for all the PSE groups, a max fanout limit of 100 is applied, as `hookup_pin` gets precedence.

```
set_dft_signal -type ScanClock -port clk -hookup_pin U1/clkout  
  
set_pipeline_scan_enable_configuration -clock_based_fanout_limit {U1/  
clkout 100 clk 500 }
```

For the clocks that do not have PSE max fanout specified, specify the value from the global PSE max fanout specification using the `set_scan_configuration -pse_fanout_limit` command. If a specific value for the PSE fanout limit is set for a clock, it overrides the global PSE fanout limit, if any.

To report the per-clock fanout specifications, use the `report_pipeline_scan_enable_configuration` command.

Limitations

The PSE max fanout per clock specification feature has the following limitations:

- Synopsys-inserted OCC output pins cannot be specified because they do not exist at the time of specification.
- For Synopsys OCC flows:
 - Specifying the fanout limit is allowed on PLL clocks. The PLL clock fanout specification translates to the associated OCC output pins.
 - Specifying the fanout limit is not allowed on a port defined as an ATE clock. You can use the global specification, `set_scan_configuration -pse_fanout_limit`.
- The fanout limit specification is not stored in the NDM database. You must specify the fanout limit in the same session of the `insert_dft` step.

9

Core Wrapping

This chapter shows you how to add a test wrapper to a design, which creates a *wrapped core*. A wrapped core provides both test access and test isolation during hierarchical testing.

The following topics provide more information:

- [Core Wrapping Concepts](#)
- [Wrapping a Core](#)
- [Defining Core Wrapping Test Modes](#)
- [Top-Down Flat Testing With Transparent Wrapped Cores](#)
- [SCANDEF Generation for Wrapper Chains](#)
- [Limitations of Core Wrapping](#)

Core Wrapping Concepts

When you integrate a DFT-inserted core into a top-level design, the core-level scan structures are integrated into the top-level scan structures. However, to test the core-level logic separately from the top-level logic, the core must be a *wrapped core*.

The core wrapping flow in Fusion Compiler is called a *maximized reuse* core wrapping flow because it reuses existing functional registers to minimize the area and timing impact.

Core wrapping is described in the following topics:

- [Wrapper Cells and Wrapper Chains](#)
- [Wrapper Test Modes](#)
- [Core Wrapper Cells](#)
- [Core Wrapper Chains](#)
- [Controlling Register Reuse](#)
- [Special Cases for Register Reuse](#)

- [Wrapper Shift Signals](#)
- [Dedicated Wrapper Cell Insertion in UPF Designs](#)
- [Low-Power Core Wrapping Features](#)

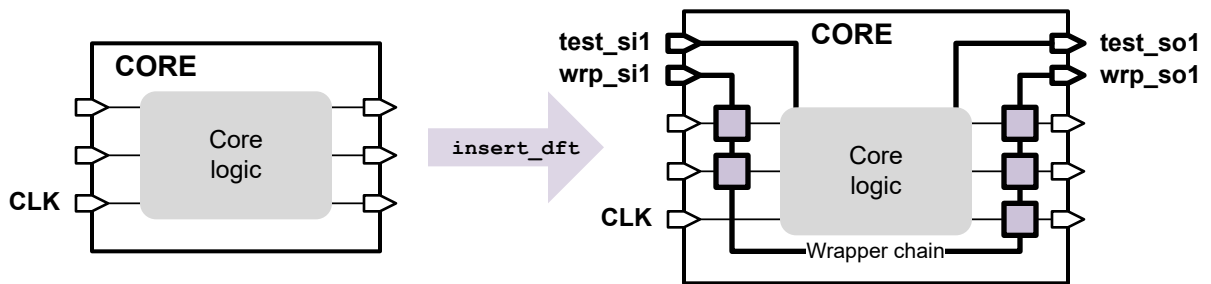
See Also

- [SolvNetPlus article 1918995, “How Do Wrapper Chains and Wrapper Cells Work in Detail?”](#) for additional reference information about wrapper chains and wrapper cells

Wrapper Cells and Wrapper Chains

A wrapped core has a *wrapper chain* that allows the core to be isolated from the surrounding logic. A wrapper chain is composed of *wrapper cells* inserted between the I/O ports and the core logic of the design. [Figure 40](#) shows an example of a wrapped core.

Figure 40 A Wrapped Core



A wrapper cell consists of a scan cell and MUX logic. It can transparently pass the I/O signal through, or it can capture values at its input and launch values at its output. Wrapper chains are shift chains (separate from regular scan chains) that allow known values to be scanned into the wrapper cells and captured values to be scanned out.

Core wrapping is primarily intended to wrap core data ports. The following ports are excluded from wrapping:

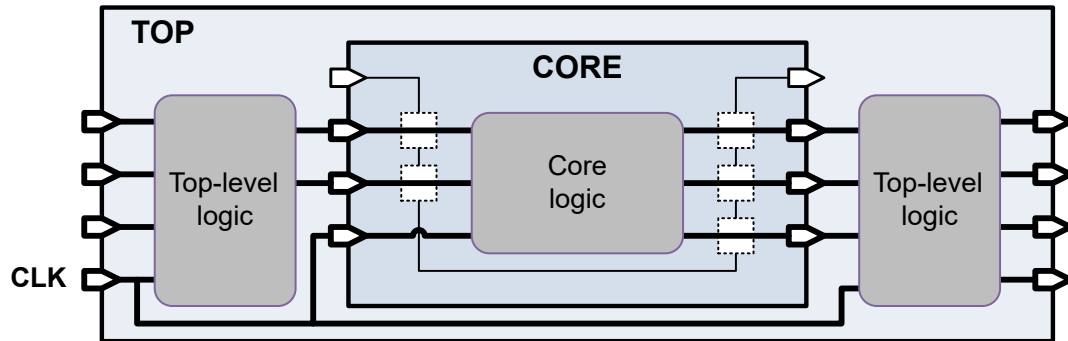
- Functional and test clock ports
- Asynchronous set or reset signal ports
- Scan-input, scan-output, scan-enable, and other global test signal ports
- Wrapper signal ports
- Any port with a constant test signal value defined

The wrapper chain operates in one of four modes—inactive, inward-facing, outward-facing, or safe. These wrapper operation modes behave as follows:

- **Inactive mode**

The wrapper chain is inactive and I/O signals pass through it. This is the behavior used in mission mode and all non-wrapper test modes.

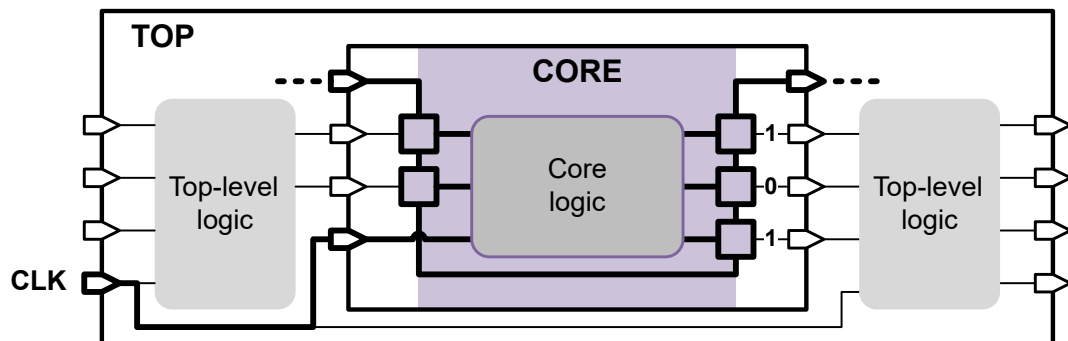
Figure 41 Inactive Mode of Wrapper Chain



- **Inward-facing mode (INTEST)**

This mode is used to test the core in isolation of the surrounding logic. It includes the wrapper chain and internal chains. The input wrapper cells provide controllability, and the output wrapper cells provide observability. If safe values are specified to protect the surrounding fanout logic from the core output response, they are driven from the output wrapper cells.

Figure 42 Inward-Facing Mode of Wrapper Chain

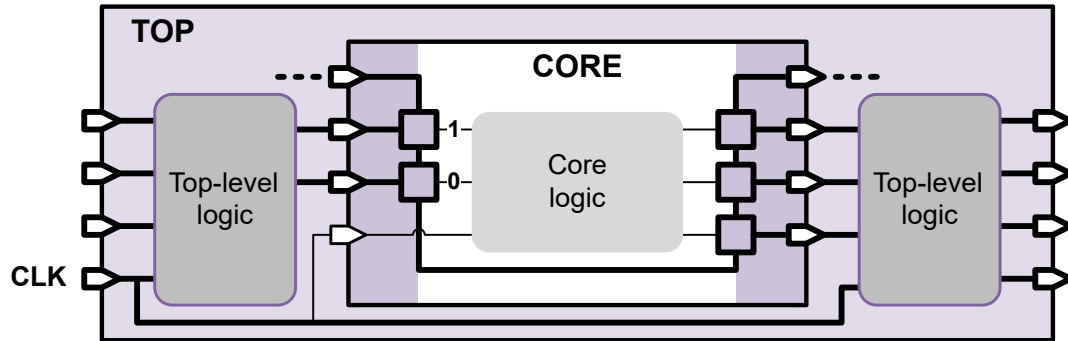


- **Outward-facing mode (EXTEST)**

This mode tests the logic surrounding the core in isolation from the core itself. It includes only the wrapper chain. The input wrapper cells provide observability, and the output wrapper cells provide controllability. If safe values are specified to protect the

core inputs from the surrounding fanin logic responses, they are driven from the input wrapper cells. Clock inputs to the core remain unaffected.

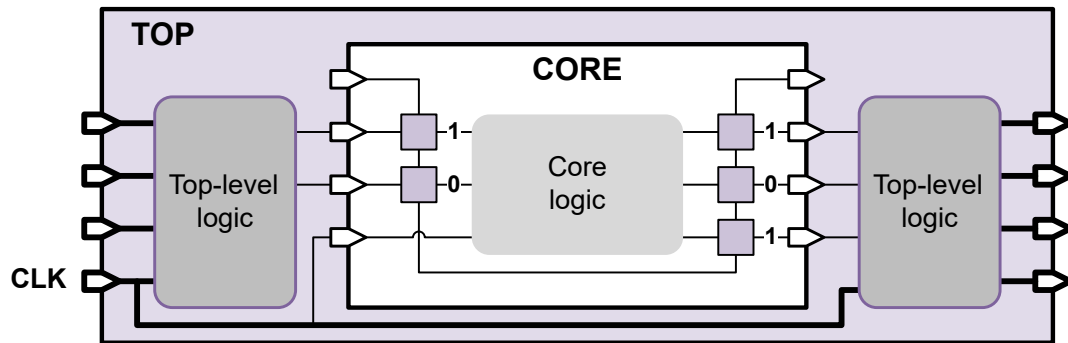
Figure 43 Outward-Facing Mode of Wrapper Chain



- **Safe mode**

This optional mode drives safe values from all wrapper cells that have a safe value specified. Safe values are driven into core inputs by any such input wrapper cells, and safe values are driven into the surrounding fanout logic by any such output wrapper cells. There are no scan chains (internal or wrapper). Clock inputs to the core remain unaffected.

Figure 44 Safe Mode of Wrapper Chain



During core creation, DFT insertion does not mix wrapper chains and regular scan chains (although they can be compressed by the same codec).

See Also

- [SolvNetPlus article 3017081, "How Does Core Wrapping Identify Clock Ports for Exclusion?"](#) for more information about how clock ports are identified

Wrapper Test Modes

When core wrapping is enabled, you can define the following test mode types:

- `ScanCompression_mode`

This is an inward-facing compressed scan mode. The wrapper chain is placed in the `INTEST` mode of operation. Both wrapper chains and internal core chains are active and compressed by the scan compression codec.

This mode is created only if scan compression is also enabled.

- `Internal_scan`

This is an inward-facing uncompressed scan mode. The wrapper chain is placed in the `INTEST` mode of operation. Both wrapper chains and internal core chains are active.

This mode is created if the `set_scan_configuration -chain_count` command is used to configure an uncompressed scan chain architecture.

- `wrp_of`

This is an outward-facing uncompressed scan mode. Only the wrapper chain is active; it is placed in the `EXTEST` mode of operation.

This mode is always created when core wrapping is enabled.

These test modes must be explicitly defined. You can use your own test mode names, if desired. For more information, see [Defining Core Wrapping Test Modes](#).

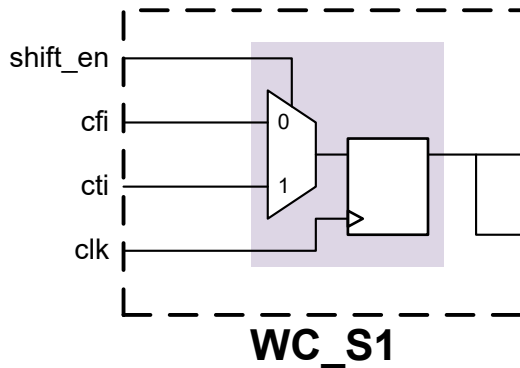
Core Wrapper Cells

Core wrapping can use two types of wrapper cells, *shared wrapper cells* and *dedicated wrapper cells*.

Shared-Register Wrapper Cells

By default, core wrapping uses *shared wrapper cells* for all ports that meet the sharing criteria. [Figure 45](#) shows the internal logic for the `WC_S1` shared wrapper cell.

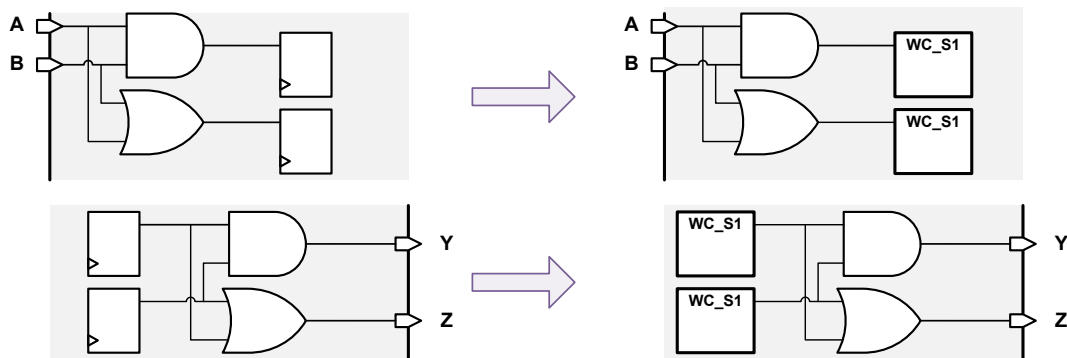
Figure 45 The WC_S1 Shared Wrapper Cell



The wrapper cell uses an *in-place* wrapper register implementation, which means it has no hierarchy around it. The wrapper cell functionality is implemented around the reused functional scan register.

Core wrapping also allows functional I/O registers connected to I/O ports through combinational logic to be shared. Figure 46 shows functional I/O registers that are replaced by shared wrapper cells.

Figure 46 Port Wrapping Examples



Diagrams in the core wrapper documentation show WC_S1 shared wrapper cell instances for clarity. However, the in-place register implementation used by the tool ensures that the names and locations of the wrapped functional registers are not disturbed.

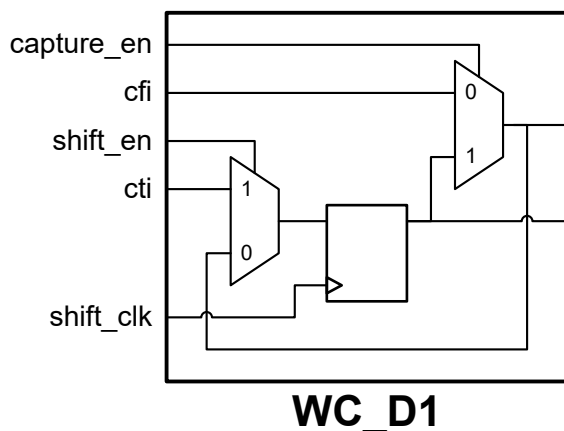
The tool provides a register-fanout threshold to limit how many functional registers can be reused for an I/O port; see [Controlling Register Reuse](#).

Using existing functional registers as shared wrapper cells can reduce the area requirements for core wrapping. Any combinational logic between the shared wrapper cell and the ports is effectively placed outside the block as far as core wrapping logic is concerned. This logic must be tested using the EXTEST wrapper mode that exercises the surrounding logic.

Dedicated Wrapper Cells

Dedicated wrapper cells are used for I/O ports to be wrapped that exceed the sharing thresholds. [Figure 47](#) shows the internal logic for the WC_D1 dedicated wrapper cell.

Figure 47 WC_D1 Wrapper Cell

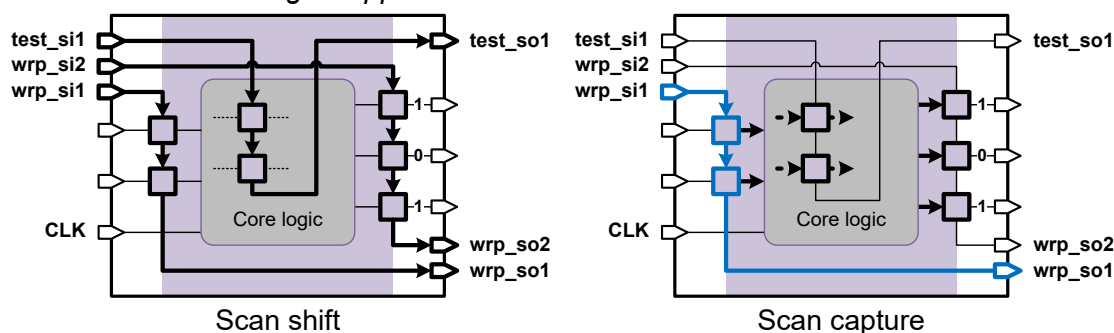


Core Wrapper Chains

Input and output wrapper cells are placed in separate wrapper chains with separate input and output wrapper shift-enable signals. During capture, wrapper chains are kept in scan shift mode as needed to block values from being captured by wrapper cells.

[Figure 48](#) shows the shift and capture behaviors used for inward-facing operation. In scan capture, input wrapper chains are kept in scan shift (highlighted in blue) to block external values at core inputs.

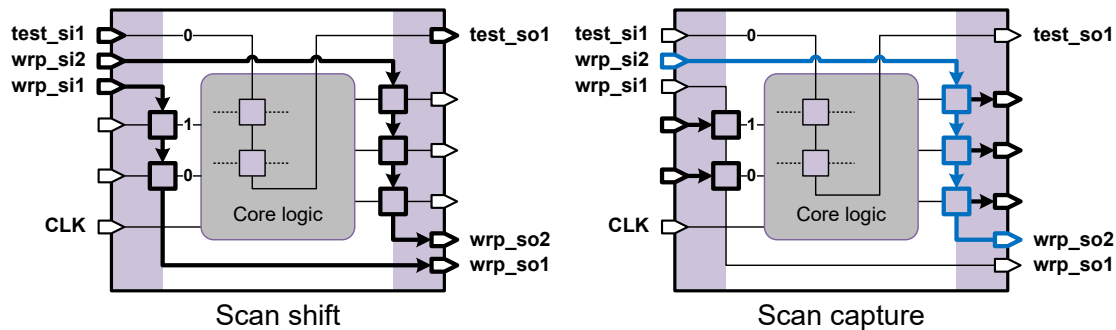
Figure 48 Inward-Facing Wrapper Chain Behavior



[Figure 49](#) shows the shift and capture behaviors used for outward-facing operation. In scan capture, output wrapper chains are kept in scan shift (highlighted in blue) to block

core-driven values at core outputs. Core wrapper chain scan-ins are driven with logic 0 to reduce power consumption.

Figure 49 Outward-Facing Wrapper Chain Behaviors



These shift and capture behaviors apply to all wrapper cells, shared and dedicated, in the input and output wrapper chains.

In transition-delay ATPG, the highlighted wrapper chains that are kept in scan shift generate transitions by shifting the opposite value into a wrapper cell from the preceding wrapper cell.

Controlling Register Reuse

By default, the tool reuses all registers associated with a port as shared wrapper cells, regardless of number. However, if some ports have many I/O registers in their fanin or fanout,

- The number of shared wrapper cells increases accordingly.
- The amount of external combinational logic between the ports and I/O registers increases (which then becomes tested in outward-facing mode instead of inward-facing mode).

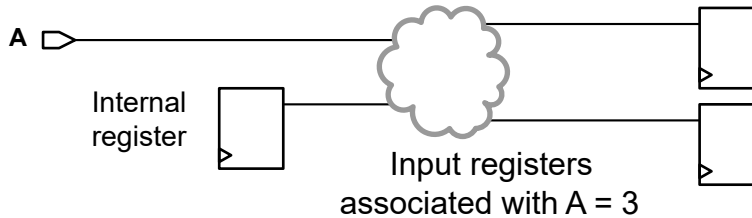
The maximized reuse feature considers how many registers exist in the fanout from an input port or the fanin to an output port. A single port might have many registers in its fanin or fanout, which would require many shared wrapper cells to fully wrap the port. If the number of fanin or fanout registers for a port exceeds a reuse threshold value, a single dedicated wrapper cell is used for that port.

To limit register reuse for ports with large numbers of I/O registers, you can apply a *reuse threshold* to the design. If the number of registers exceeds the reuse threshold, a dedicated wrapper cell is placed at the port.

Applying Reuse Thresholds to Input Ports

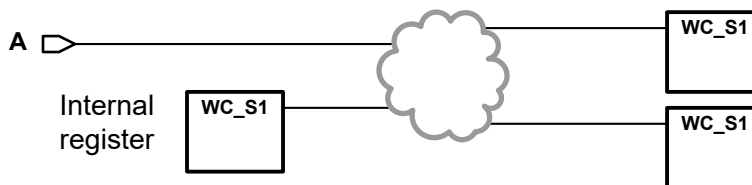
For an input port, the tool determines the number of registers in the fanout of the port. In addition, it includes any internal registers that feed data pins or synchronous set/reset pins of the fanout registers. Figure 50 shows an input port with an associated register count of three.

Figure 50 Input Port Register Count Computation Example



In this example, if the reuse threshold is set to a value of three or higher, the tool replaces the registers with shared wrapper cells, as shown in Figure 51.

Figure 51 Input Port Registers After Wrapper Cell Replacement



The tool places the shared wrapper cells for the fanout registers in the input wrapper chain. Because any internal registers feeding these fanout registers capture values from the core logic instead of input ports, the tool places the shared wrapper cells for these internal registers in the output wrapper chain.

Dedicated wrapper cells are added to input ports according to the following rules:

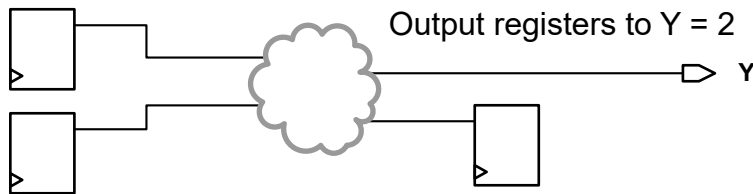
- If the sum of the input fanout registers and the internal registers feeding them exceeds the reuse threshold, then a dedicated wrapper cell is added to the input port.
- When the input register cells exceed the reuse threshold for a bidirectional port, dedicated wrapper cells are added to the data-out, enable, and data-in paths.
- If an input port is associated with a CTL-modeled cell, then a dedicated wrapper cell is added to the port. See [Wrapping Ports Associated With CTL-Modeled Cells](#) for details.

A warning is issued if all registers associated with an input port do not use the same clock.

Applying Reuse Thresholds to Output Ports

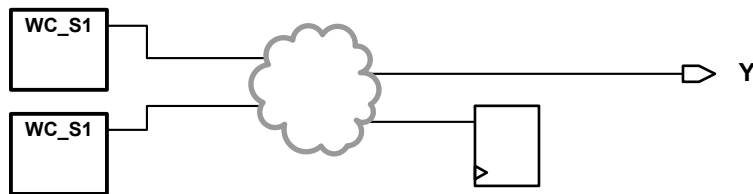
For an output port, the tool determines the number of registers in the fanin of the port. It does not include any additional surrounding registers. [Figure 52](#) shows an output port with an associated register count of two.

Figure 52 Output Port Register Count Computation Example



In this example, if the reuse threshold is set to a value of two or higher, the tool replaces the registers with shared wrapper cells, as shown in [Figure 53](#).

Figure 53 Output Port Registers After Wrapper Cell Replacement



The tool places the shared wrapper cells for the fanin registers in the output wrapper chain.

Dedicated wrapper cells are added to output ports according to the following rules:

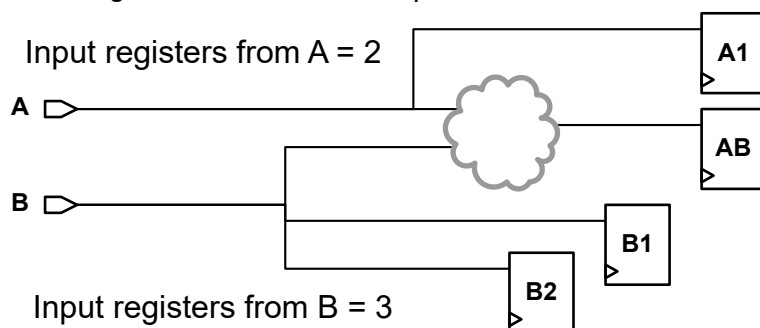
- If the sum of the output fanin registers exceeds the reuse threshold, then a dedicated wrapper cell is added to the output port.
- When the number of the output register cells exceeds the reuse threshold value for a three-state output port, dedicated wrapper cells are added to both data-out and enable paths.
- If an output port is associated with a CTL-modeled cell, then a dedicated wrapper cell is added to the port. See [Wrapping Ports Associated With CTL-Modeled Cells](#) for details.
- For three-state and bidirectional ports, only a functional register that directly controls the pad is considered as a shared wrapper cell for the enable path of the pad. There can be inverting or noninverting buffers between the register and the pad enable pin. If no such register is found, a dedicated wrapper cell is added at the driver cell enable pin to control the port.

A warning is issued if all registers associated with an output port do not use the same clock.

Registers Associated With Multiple Ports

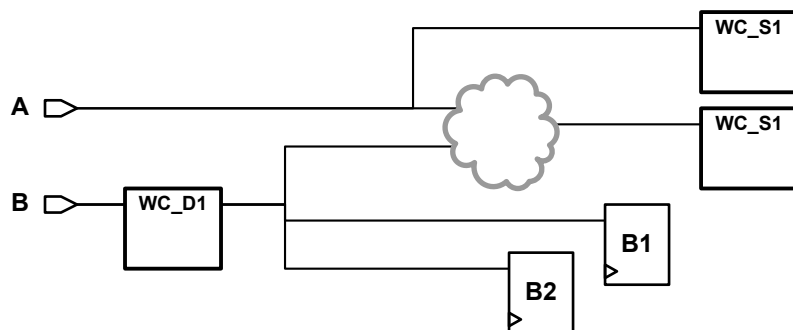
If the same register is associated with multiple ports, the register is replaced with a shared wrapper cell if *any* port meets the threshold. In [Figure 54](#), register AB is in the fanout of ports A and B. Input port A has two associated registers, and input port B has three associated registers.

Figure 54 Register in Fanout of Multiple Ports



In this example, if the reuse threshold is set to a value of two, the tool replaces all registers for input port A with shared wrapper cells, and it inserts a dedicated wrapper cell at input port B, as shown in [Figure 55](#). Register AB is replaced with a shared wrapper cell to completely wrap port A, even though it is also in the fanout of the dedicated wrapper cell for input port B.

Figure 55 Register in Fanout of Multiple Ports After Wrapper Cell Replacement



Registers that become shared wrapper cells for one port do not affect the associated register count for other ports. In [Figure 54](#), the associated register count for input port B is three even though register AB is wrapped for input port A.

If a register is classified as both an input shared register and an output shared register, the register becomes an input shared register and is removed from the output shared register

list. Unlike the simple core-wrapping flow, no dedicated wrapper cell is placed at the output port.

Special Cases for Register Reuse

In some cases, ports are associated with logic constructs that require special-case handling for register reuse.

An information message is issued if a feedthrough port is detected and no dedicated wrapper cell is manually specified for the port:

```
Information: No I/O registers are found for port 'B';
             not adding any dedicated wrapper cells to the port. (TEST-1180)
Information: No I/O registers are found for port 'Y';
             not adding any dedicated wrapper cells to the port. (TEST-1180)
```

Wrapping Ports Associated With CTL-Modeled Cells

CTL-modeled cells, such as memories or DFT-inserted cores, cannot be used as shared wrapper cells. When a port is associated with a CTL-modeled cell, as shown in the examples in [Figure 56](#) and [Figure 57](#), a dedicated wrapper cell is added to the port.

Figure 56 Core Wrapping of Input Ports Associated With CTL-Modeled Cells

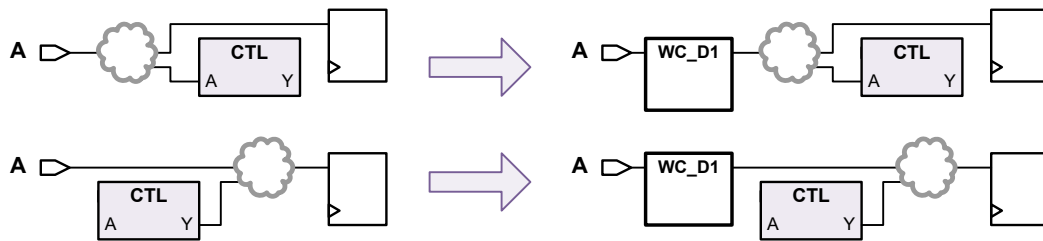
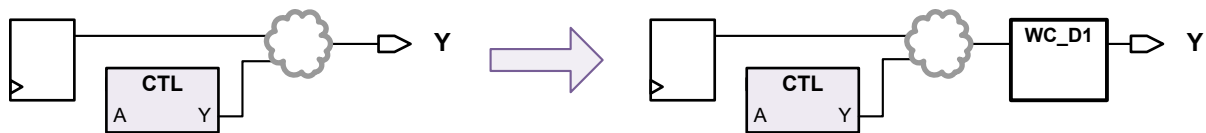


Figure 57 Core Wrapping of Output Ports Associated With CTL-Modeled Cells



In addition, a warning message is issued:

```
Warning: Port 'DACK' is connected to cell 'IP_CORE', which is represented
         by a CTL model; a dedicated wrapper cell is added to the port.
```

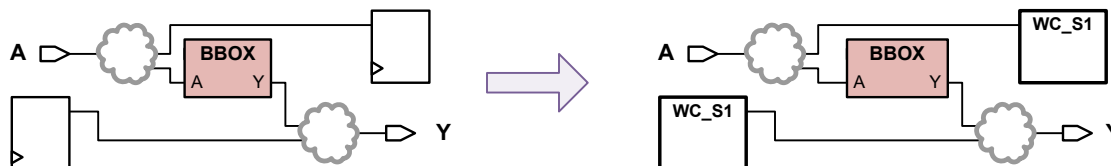
The scan clocks of the CTL-modeled cell are considered when determining the dominant system clock to use for the dedicated wrapper cell.

If the CTL-modeled cell has a netlist and there is no path from the port to a register inside the CTL-modeled cell, this behavior does not apply. In other words, connections to clock, reset, or DFT-related pins of a CTL-modeled cell do not force a dedicated wrapper cell.

Port Wrapping and Other Black-Box Cells

Black-box cells without CTL models do not affect core wrapping. Wrapper cells are not added unless needed by other design logic. See [Figure 58](#).

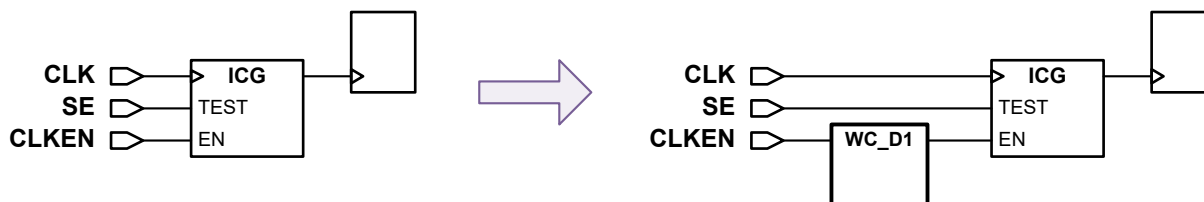
Figure 58 Black-Box Cell With No CTL Model



Wrapping Ports Associated With Clock-Gating Cells

A dedicated wrapper cell is added to an input port that drives the enable signal of a clock-gating cell. In [Figure 59](#), an integrated clock-gating cell has both a functional enable pin and a test-mode pin. A dedicated wrapper cell is added for the functional enable signal driven by input port CLKEN. No wrapper cells are inserted for the clock signal or the global test-mode signal.

Figure 59 Integrated Clock Gating Cell Enable Signals



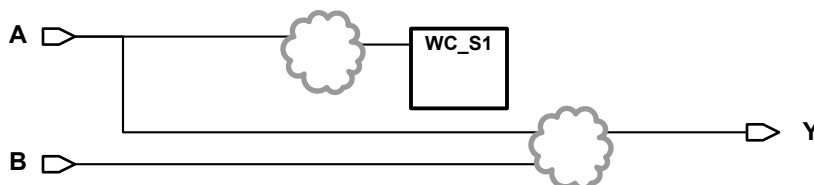
A warning message is issued if such a clock-gating enable signal is detected:

```
Warning: Port 'CLKEN' is connected to a clock gating cell 'UICG';
a dedicated wrapper cell is added to the port.
```

Port Wrapping and Feedthrough Paths

No wrapper cell is added to a port that is bounded only by combinational logic unless you explicitly specify a dedicated wrapper cell for that port. See [Figure 60](#).

Figure 60 Combinational Feedthrough Paths



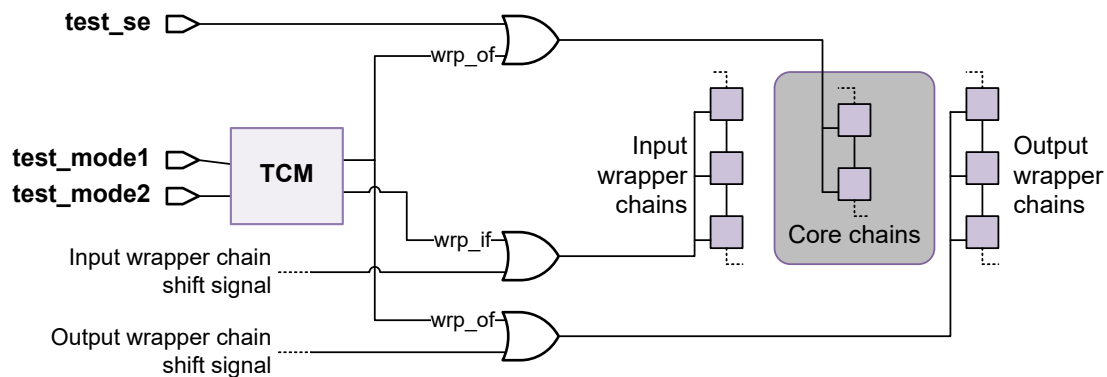
Wrapper Shift Signals

The scan-enable and wrapper shift signals are conditioned by the currently active test mode as follows:

- In inward-facing wrapper modes, the input wrapper shift signal is always asserted.
- In outward-facing wrapper modes, the output wrapper shift signal is always asserted.
- In outward-facing wrapper modes, the scan-enable signal for the internal core scan chains is always asserted (to load constant values into all scan chain elements).

Figure 61 shows an example of the conditioning logic.

Figure 61 Scan-Enable and Wrapper Shift Signals



The wrapper shift signals going to the input and output wrapper chains are conditioned separately as shown in the diagram, even when a single scan-enable signal is used for all three shift signals (scan, input wrapper, and output wrapper).

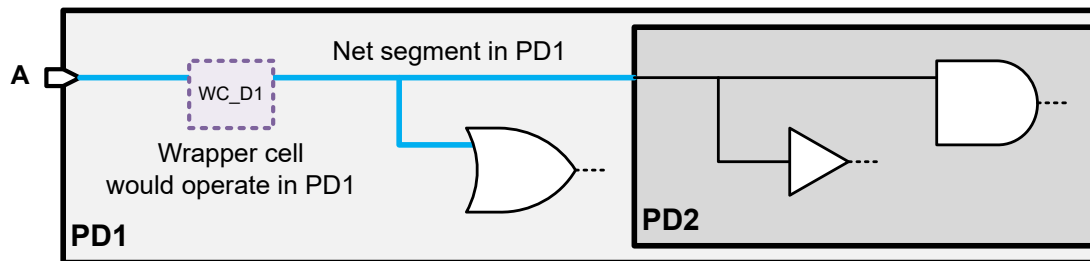
See Also

- [Defining Wrapper Shift Signals](#)

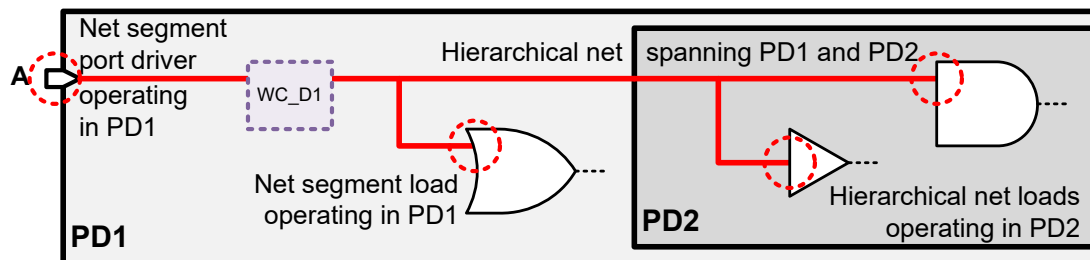
Dedicated Wrapper Cell Insertion in UPF Designs

When the tool inserts dedicated wrapper cells in a design that contains UPF information, it prefers the highest possible hierarchical nets that do not result in multivoltage violations.

When a dedicated wrapper cell is inserted on a net segment, its logic inherits—and thus operates in—the power domain containing that particular net segment:



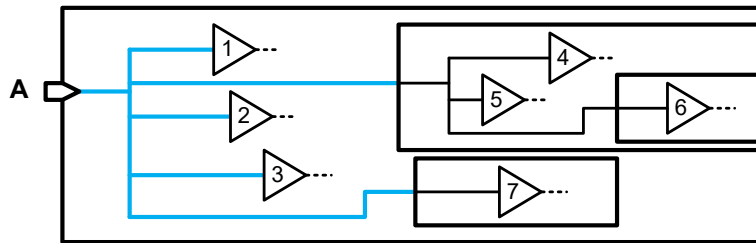
But if the net segment is part of a hierarchical net that crosses power domains, the wrapper cell power domain must be compatible with the driver and load power domains of the *entire* hierarchical net:



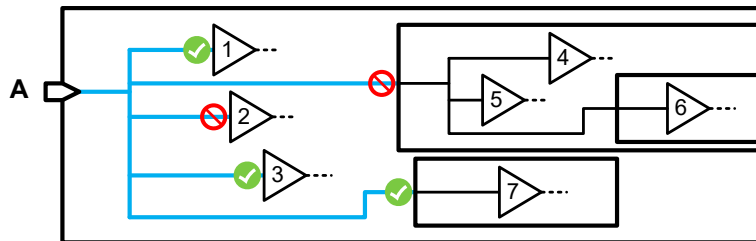
Dedicated Wrapper Cell Insertion Algorithm

To meet these requirements, the tool inserts dedicated wrapper cells for input ports as follows:

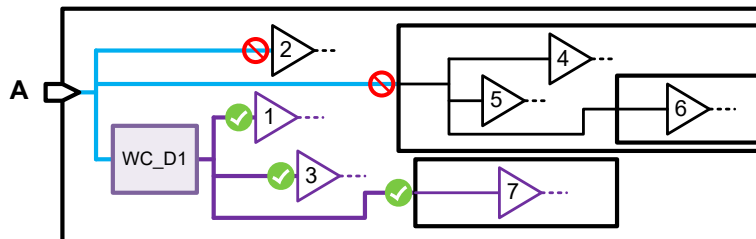
1. Starting at the port, the tool searches for the highest hierarchical net segment (closest to the port) where a dedicated wrapper cell could be inserted.



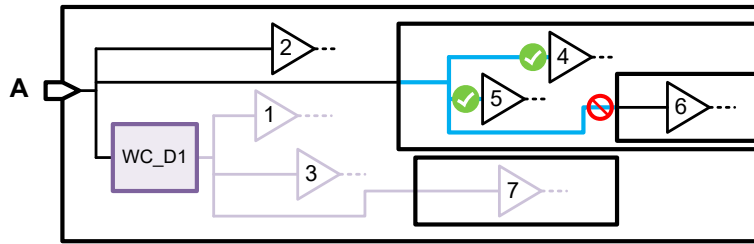
2. At each net segment, the tool determines which segment load pins—hierarchical and leaf—can be driven by a dedicated wrapper cell without introducing a multivoltage violation. A hierarchical load pin violates if any downstream leaf load pin would violate.



3. If any segment load pins can be driven without violation, the tool inserts a dedicated wrapper cell to drive them. Their downstream logic is now wrapped.



4. The search then continues forward through any unwrapped load pins.



The search can also cross buffers, inverters, isolation cells, and level-shifter cells; it stops at all other cells.

5. When the search completes, if any unwrapped load pins remain, the tool repeats the search but this time allowing *fixable* multivoltage violations to occur.

Caution:

The first search (for nonviolating locations) considers the existing isolation strategies and power state tables. The second search (for fixable locations) considers a location fixable if an isolation strategy *could* be applied to insert the isolation cell, even if no such strategy currently exists.

The process works similarly for output ports, except that the search proceeds backward through each net segment driver pin instead of forward through the load pins. Only one dedicated wrapper cell can be inserted for an output port.

A dedicated wrapper cell cannot be inserted in the following locations:

- Between an isolation cell and the upper boundary of the power domain where that isolation strategy is applied (using the `-domain` option)
- Between a zero-pin retention cell and its clamp cell
- Between an isolation, retention, or switch control signal and the power management cells that it controls

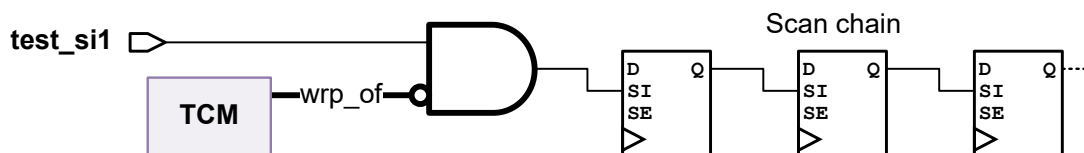
Low-Power Core Wrapping Features

Core wrapping implements the following power-saving logic.

Loading Constant Core Scan Data in EXTEST Mode

To minimize power consumption in `wrp_of` (EXTEST) mode, all core scan chain cells are loaded with logic 0. This reduces toggle activity inside the block during outward-facing (EXTEST) modes. To accomplish this, the logic shown in [Figure 62](#) drives the first scan cell inputs of the internal scan chains.

Figure 62 Scan Data Gating Logic for Low-Power EXTEST



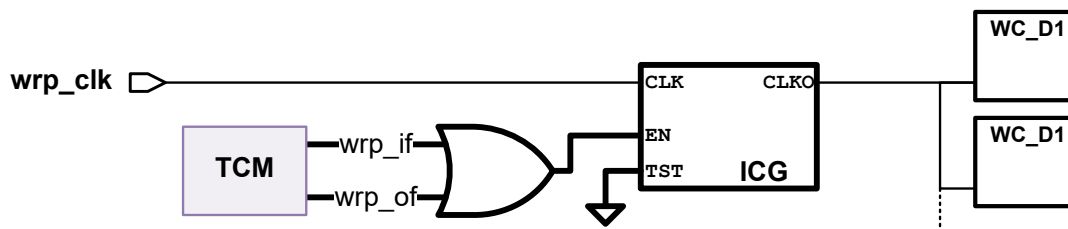
This feature is part of the wrapped core scan architecture; there is no option that controls it.

Gating Dedicated Wrapper Cell Clocks in Non-Wrapper Modes

The tool inserts clock-gating cells to disable the clock used for dedicated wrapper cells when a wrapper mode is not active.

The tool adds a clock-gating cell to each dedicated wrapper cell clock source, including any scan clocks used for dedicated wrapper cells, as shown in [Figure 63](#). Shared wrapper cells and scan cells are not affected, even if they use the same clock as a dedicated wrapper cell.

Figure 63 Clock Gating Logic for Low-Power Dedicated Clock Operation



This feature requires that an integrated clock-gating cell be specified with the `dft.test_icg_p_ref_for_dft` application option.

Wrapping a Core

Core wrapping is performed during RTL DFT insertion. Core wrapping configuration is described in the following topics:

- [Enabling Core Wrapping](#)
- [Enabling Soft Core Wrapping for Non-abutted Flow](#)
- [Defining Wrapper Shift Signals](#)
- [Configuring Wrapper Clock Signals](#)
- [Controlling Wrapper Chain Count](#)
- [Setting a Reuse Threshold](#)

- [Configuring Port-Specific Wrapper Settings](#)
- [Mixing Input and Output Wrapper Cells](#)
- [Previewing the Wrapper Cells](#)
- [Creating an EXTEST-Only Core Netlist](#)

Enabling Core Wrapping

To enable core wrapping, use the following command after loading and linking the design:

```
fc_shell> set_dft_configuration -wrapper enable
```

Enabling Soft Core Wrapping for Non-abutted Flow

Core wrapper insertion supports wrapping the entire design by adding wrapper cells to the design interface. For some of the sub hierarchies, you may want to add wrapper logic inside the hierarchies to isolate those hierarchies. Soft core wrapping such as DFT core wrapping allows you to wrap top-level ports along with soft core hierarchies.

Enable the `dft.test_wrp_soft_core_abutted_flow` application option (off by default). Enabling this application option allows you to skip core wrapping at the top level. Any new logic relevant to wrapper insertion for the soft core is added in the soft core itself and for all ports at the top-level.

Disabling this application option enables non-abutted flow.

Defining Wrapper Shift Signals

The wrapper shift signal enables scan data to shift through wrapper chains, just as a scan-enable signal does for scan chains. You can use any of the following wrapper shift signal configurations.

Using the Default Scan-Enable Signal

By default, Fusion Compiler uses the design's scan-enable signal as the wrapper shift signal:

```
fc_shell> set_dft_signal -view spec -type ScanEnable \  
    -port my_test_se ;# by default, this is used for wrapper cells
```

Unless your design requires otherwise, no further configuration is needed.

Defining a Single Dedicated Wrapper Shift Signal

To define a single dedicated wrapper shift signal, define the signal source as a `wrp_shift` signal:

```
fc_shell> set_dft_signal -view spec -type wrp_shift \  
-port my_wrp_shift
```

Using a Particular Scan-Enable Signal as the Wrapper Shift Signal

If you have multiple scan-enable signals, you can choose one of them to be the wrapper shift signal by also defining it as a `wrp_shift` signal:

```
fc_shell> set_dft_signal -view spec -type ScanEnable \  
-port {my_test_sel my_test_se2}  
fc_shell> set_dft_signal -view spec -type wrp_shift \  
-port my_test_se2
```

Defining Separate Input and Output Wrapper Shift Signals

To use separate wrapper shift signals for input and output wrapper chains, define two signals as `input_wrp_shift` and `output_wrp_shift`, respectively:

```
fc_shell> set_dft_signal -view spec \  
-type input_wrp_shift -port {wrp_ishift}  
fc_shell> set_dft_signal -view spec \  
-type output_wrp_shift -port {wrp_oshift}
```

You can specify only a single signal for each option.

```
fc_shell> # define test clocks  
fc_shell> set_dft_signal -view existing_dft -type ScanClock \  
-timing {45 55} -port {CLK1A CLK1B CLK2}  
fc_shell> # define per-clock-domain wrapper shift signals  
fc_shell> set_dft_signal -view spec -type wrp_shift \  
-port WRP_SHIFT1 -connect_to {CLK1A CLK1B}  
fc_shell> set_dft_signal -view spec -type wrp_shift \  
-port WRP_SHIFT2 -connect_to {CLK2}
```

Configuring Wrapper Clock Signals

Shared wrapper cells reuse an existing, clocked functional register; no clock configuration is needed for them.

Dedicated wrapper cells exist only at ports where shared wrapper cells could not be used. In this case, the tool attempts to identify and use the dominant clock domain associated with each port using the following rules:

- A port's dedicated wrapper cell uses the same clock as the functional registers associated with that port.
- If the port is associated with registers of multiple clock domains, the dominant clock is used.
- If a dominant clock is not found, any wrapper clock, defined using the `set_dft_signal -view spec -type wrp_clock` command, is used.
- If a wrapper clock is not found, any scan clock, defined using the `set_dft_signal -view spec -type ScanClock` command, is used.

Controlling Wrapper Chain Count

In Fusion Compiler, the compressed wrapper chains are architected first, then the other modes are architected using the compressed wrapper chains as segments. These compressed wrapper chain segments cannot be further subdivided.

Wrapper cells and internal scan cells cannot be mixed on the same chain. Also, input and output wrapper cells cannot be mixed on the same chain unless enabled as described in [Mixing Input and Output Wrapper Cells](#).

Inward-Facing Compressed Mode

In this mode, Fusion Compiler always balances the wrapper chains and scan chains together to meet the compressed chain count specification *IC* specified by

```
fc_shell> set_scan_compression_configuration \  
-chain_count IC ;# compressed DFTMAX chain count  
fc_shell> # ...or...  
fc_shell> set_streaming_compression_configuration \  
-chain_count IC ;# compressed DFTMAX Ultra chain count
```

This mode is created only when scan compression is enabled.

Inward-Facing Uncompressed Mode

In this mode, the tool creates wrapper chains by concatenating the smaller compressed wrapper chain segments. The tool balances the compressed wrapper chain segments and scan cells together to meet the overall chain count target *IU* specified by

```
fc_shell> set_scan_configuration \  
-chain_count IU ;# inward-facing uncompressed chain count
```

This mode is created only when an uncompressed scan mode is configured using the preceding command.

Outward-Facing Uncompressed Mode

In this mode, the tool creates wrapper chains using the smaller compressed wrapper chain segments as building blocks. The default wrapper chain count *O* is chosen to match the scan I/O resources of other modes in the design (highest priority first):

- Uncompressed inward-facing mode (Internal_scan)
`(set_scan_configuration -chain_count IU)`
- Compressed inward-facing mode (ScanCompression_mode)
`-outputs Nset_scan_compression_configuration -inputs N`

To specify a particular chain count for the outward-facing mode, use the following command:

```
fc_shell> set_wrapper_configuration \  
          -chain_count W ;# outward-facing mode wrapper chain count
```

This specification does not affect the architecture of the compressed wrapper chains. If there are not enough wrapper chain segments to satisfy this constraint, the tool issues a warning message during DFT preview and insertion:

```
Warning: Only '2' wrapper chains can be architected in wrp_of mode.  
Cannot honor -chain_count specification of '5'.
```

Setting a Reuse Threshold

By default, the tool reuses all registers associated with a port as shared wrapper cells, regardless of number. However, you can set a *reuse threshold* to control this, as described in [Controlling Register Reuse](#).

To set a reuse threshold, use the following command:

```
fc_shell> set_wrapper_configuration -reuse_threshold N
```

The default is no limit (infinity).

Configuring Port-Specific Wrapper Settings

To specify the wrapper cell characteristics of specific ports, you can use the `set_boundary_cell` command.

To prevent the insertion of wrapper cells for a specific list of ports, use the following command:

```
fc_shell> set_boundary_cell -ports port_list -type none
```

This might be needed for special input or output port signals that should not be controlled during test. Since excluding ports from the wrapper chain reduces test coverage, you should use this capability only when necessary.

To specify a dedicated wrapper cell for ports that would otherwise use a shared wrapper cell, specify a WC_D1 wrapper cell with the `set_boundary_cell` command:

```
fc_shell> set_boundary_cell -ports port_list -type WC_D1
```

You cannot use the `set_boundary_cell` command to force a shared wrapper cell type to be used for a port if the I/O register does not meet the requirements for a shared wrapper cell.

Mixing Input and Output Wrapper Cells

By default, input and output wrapper cells cannot be mixed on the same scan chain. To allow this mixing, use the following command:

```
fc_shell> set_wrapper_configuration \  
-mix_cells true
```

Previewing the Wrapper Cells

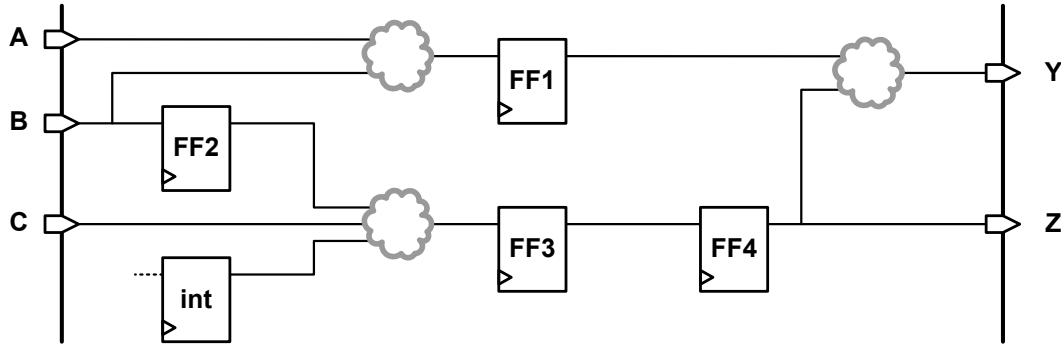
After you have configured your wrapper chain and wrapper cells, use the `preview_dft` command to preview the core wrapping configuration.

For all ports that have wrapper cells, the following information is reported:

- Index number of wrapper cell (used to indicate duplicate input/output wrapper cells and to reference control cells)
- Port name
- Wrapper cell type: dedicated or shared
- Function of wrapper cell: input, output, three-state, or control
- Wrapper cell implementation: swapped-in (SWP) or in-place (INP)
- Safe value
- Wrapper cell clock
- Wrapper cell instance name

Consider the following example design:

Figure 64 Example Design for Core Wrapping



Example 6 shows the wrapper chain preview report for this design.

Example 6 Core Wrapping Preview Report

```
fc_shell> preview_dft
...omitted...

*****Test wrapper plan report
Design : top
*****
Reuse Threshold : 5
Wrapper Length : 5
```

Index	Port	Wrapper Function	Cell Type	Control Cell	Impl	Safe Value	Wrapper Clock	Cell Name
4	C	input	WC_S1	-	INP	0	CLK	FF3_reg
3	A	input	WC_S1	-	INP	0	CLK	FF1_reg (*)
3	B	input	WC_S1	-	INP	0	CLK	FF1_reg (*)
3	Y	output	WC_S1	-	INP	0	CLK	FF1_reg (*)
2	Y	output	WC_S1	-	INP	0	CLK	FF4_reg (d)
2	Z	output	WC_S1	-	INP	0	CLK	FF4_reg (d)
1	B	input	WC_S1	-	INP	0	CLK	FF2_reg (*)
1	C	output	WC_S1	-	INP	0	CLK	FF2_reg (*)
0	C	output	WC_S1	-	INP	0	CLK	int_reg

```

Number of ports not wrapped      : 26

ATECLK
CLK
OCC_bypass
OCC_reset
SE
...

```

The preview report uses annotations to indicate when a shared wrapper cell is associated with multiple ports. Note the following reporting conventions:

- Shared registers that are common to multiple input ports and multiple output ports are shown with the letter d (d) for all subsequent entries after the first register entry; these marked entries do not count against the total wrapped register count.
- Internal registers in the fanin to an input shared register are classified as output registers (because they drive values that are captured during outward-facing test).
- Shared registers that are common to both an input port and an output port but are used only as input wrapper cells are classified as input registers and shown with an asterisk (*) for the output port.
- Shared input registers that are in the fanin to another shared input register are classified as input registers and shown with the annotation (*) (i) for the fanin to the shared input register.

Creating an EXTEST-Only Core Netlist

During core creation, you can create and write out an additional version of the core netlist that contains only the logic needed for operation in outward-facing (EXTEST) test modes. This EXTEST-only core netlist significantly reduces the memory requirements for pattern generation of top-level test modes in TestMAX ATPG and for pattern simulation.

After inserting DFT and writing the full netlist files, protocol files, and other design files, execute the following command:

```
fc_shell> create_dft_netlist -type extest
```

This command removes all logic in the design except for the following:

- Wrapper chains
- Interface logic between wrapper chains and I/O ports
- Wrapper chain control logic
- Test-mode decode logic
- IEEE 1500 controller logic
- Any other logic required for EXTEST mode operation

After removing the unnecessary logic with the `create_dft_netlist` command, write out an EXTEST-only Verilog netlist file using the `write_verilog` command. Because the `create_dft_netlist` command removes logic from the design in memory, these should be the last steps in your core creation script.

You can use this EXTEST-only core netlist in any TestMAX ATPG pattern generation run where the core operates in its outward-facing test mode. All necessary logic is retained so that the core operates properly when exercised by the test protocols.

Note the following limitations:

- The `create_dft_netlist` command can be run only after DFT is inserted.
- The `create_dft_netlist` command uses the EXTEST mode CTL stored in the NDM for preserving the required cells. Any structural design modifications made after DFT insertion are not considered during netlist processing.
- Cores with multiple EXTEST test modes are not supported.

Defining Core Wrapping Test Modes

In Fusion Compiler, when core wrapping is enabled, you must define the list of test modes using the `define_test_mode` command; the wrapper modes are not automatically created.

Table 4 shows the test mode usage types you can specify with the `-usage` option of the `define_test_mode` command. You can define one or more test modes for each usage.

Table 4

Test mode usage	Test mode description	Default test mode name
<code>wrp_if</code>	Inward-facing uncompressed scan mode	<code>Internal_scan</code>
<code>scan_compression</code>	Inward-facing compressed scan mode	<code>ScanCompression_mode</code>
<code>wrp_of</code>	Outward-facing uncompressed scan mode	<code>wrp_of</code>
<code>scan</code>	Unwrapped uncompressed scan mode	

You can configure individual user-defined test modes using the `-test_mode` option of DFT configuration commands. The `-test_mode` option cannot reference a default test mode unless that mode name was explicitly defined with the `define_test_mode` command.

Example 7 defines a set of user-defined core wrapping test modes. The inward-facing compressed scan mode, defined with `-usage scan_compression`, specifies the base mode as the inward-facing uncompressed scan mode, defined with `-usage wrp_if`.

Example 7 User-Defined Core Wrapping Test Modes

```
# define test modes
define_test_mode MY_INTEST      -usage wrp_if
define_test_mode MY_INTEST_COMP -usage scan_compression
```



```
define_test_mode MY_EXTEST          -usage wrp_of

# configure scan modes
set_scan_compression_configuration \
  -test_mode MY_INTEST_COMP \
  -input 4 -output 4 \
  -chain_count 12
set_scan_configuration \
  -test_mode MY_INTEST \
  -chain_count 4
set_wrapper_configuration \
  -test_mode MY_EXTEST \
  -class core_wrapper -chain_count 4
```

Top-Down Flat Testing With Transparent Wrapped Cores

In some cases (such as for IDDQ testing), you might want to make core wrapper chains transparent to perform top-down flat testing of the full chip. This can be done by implementing and using a *transparent mode* in your wrapped cores.

Transparent modes are described in the following topics:

- [Introduction to Transparent Test Modes](#)
- [Defining Core-Level Transparent Test Modes](#)
- [Defining Top-Level Flat Test Modes](#)
- [Limitations](#)

Introduction to Transparent Test Modes

Wrapped cores allow a design to be tested hierarchically. Inward-facing test modes test the logic inside a core, and outward-facing test modes test the logic surrounding a core. In both types of modes, the wrapper chain is actively used to logically separate the core logic from the surrounding logic.

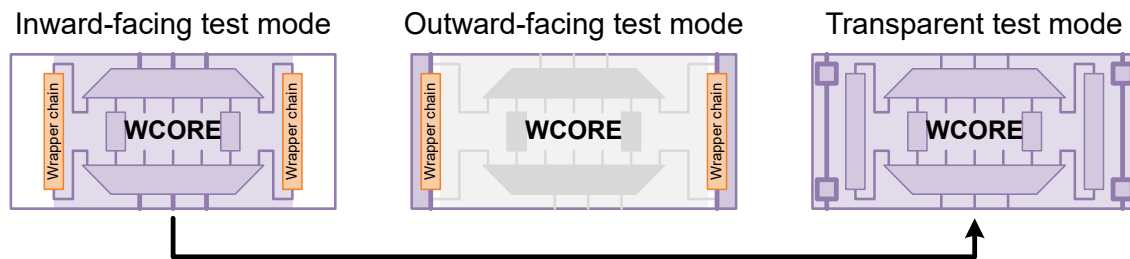
To perform top-down flat testing of a design with wrapped cores, wrapped cores must provide an additional test mode where the wrapper chain is logically transparent. This core-level test mode is called a *transparent* test mode. At the top level, wrapped cores can be placed into their transparent modes to perform top-down flat testing of the entire design. This top-level test mode is called a *flat* test mode.

At the core level, a transparent test mode is defined as an extension of an inward-facing standard scan or compressed scan test mode. A transparent mode is identical to its referenced inward-facing mode, except that

- Wrapper chains are treated as regular scan chains.
- Dedicated wrapper cells are logically transparent, although their flip-flops remain in the wrapper chains and act as observe test points on I/O paths.
- Any scan compression codecs from the referenced inward-facing mode are reused.
- Feedthrough chains drive the outward-facing wrapper scan I/Os in transparent mode.

Figure 65 shows how the transparent mode of a wrapped core relates to the inward-facing and outward-facing test modes. Standard scan modes are not shown.

Figure 65 The Three Test Mode Types of a Wrapped Core



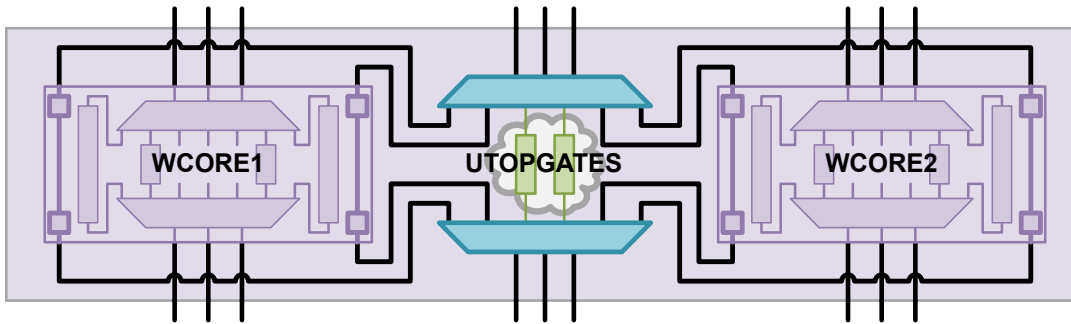
Transparent modes require that dedicated wrapper scan I/Os be used in the core's outward-facing modes. The tool creates and stitches placeholder chains, called *feedthrough* chains, between these outward-facing I/Os in the transparent modes. Each feedthrough chain consists of two scan cells that are clocked by the same head and tail clock and edge as the outward-facing wrapper chains they represent.

At the top level, a flat test mode is defined as an extension of a top-level-only mode, which tests only top-level logic with wrapped cores in outward-facing mode. A flat mode is identical to its underlying top-level-only mode, except that:

- You schedule wrapped cores in transparent mode instead of outward-facing mode.
- Any top-level scan compression codecs from the underlying top-level-only mode are reused.

Figure 66 shows how wrapped cores, placed in their transparent modes, are tested along with the top-level logic in a top-level flat test mode.

Figure 66 Performing Top-Down Flat Testing Using Wrapped Cores in Transparent Mode



The feedthrough chains allow the codec from the top-level-only mode to be reused in the flat test mode. Feedthrough chains are used only in flat test modes where they are compressed by a top-level codec reused from the underlying top-level-only mode.

Feedthrough chains are unused in

- Top-level flat standard scan modes
- Top-level flat compressed scan modes without top-level codecs that compress the core wrapper connections
- Flat modes where the top-level codec participates in codec I/O sharing

See [Limitations](#).

Defining Core-Level Transparent Test Modes

To define a transparent test mode at the core level, use the `-transparent_mode_of` option of the `define_test_mode` command to specify a previously defined inward-facing standard scan or compressed scan test mode as the parent mode. The transparent mode is derived from this parent mode. For example,

```
# define inward-facing modes
define_test_mode INWARD_STD -usage wrp_if
define_test_mode INWARD_COMP -usage scan_compression

# define outward-facing mode
define_test_mode OUTWARD_STD -usage wrp_of

# define transparent modes
define_test_mode TRANS_STD -usage wrp_if \
    -transparent_mode_of INWARD_STD
define_test_mode TRANS_COMP -usage scan_compression \
    -transparent_mode_of INWARD_COMP
```

Specify the usage of a transparent mode to be the same as its parent mode: `wrp_if` for a standard scan mode and `scan_compression` for a compressed scan mode.

To define dedicated wrapper scan I/Os for the core's outward-facing modes, use the `-test_mode` option of the `set_dft_signal` command. For example,

```
# define dedicated scan I/Os for outward-facing mode
set_dft_signal -view spec -type ScanDataIn -port SI[*] \
  -test_mode {INWARD_STD TRANS_STD INWARD_COMP TRANS_COMP}
set_dft_signal -view spec -type ScanDataOut -port SO[*] \
  -test_mode {INWARD_STD TRANS_STD INWARD_COMP TRANS_COMP}

set_dft_signal -view spec -type ScanDataIn -port WSI[*] \
  -test_mode {OUTWARD_STD}
set_dft_signal -view spec -type ScanDataOut -port WSO[*] \
  -test_mode {OUTWARD_STD}
```

For more information, see the man page for this message.

Defining Top-Level Flat Test Modes

To define a flat test mode at the top level, define a test mode that schedules the cores in their transparent mode along with the top-level logic of the current design. For example,

```
# define modes that test only cores
define_test_mode CORES_INWARD_STD -usage scan \
  -target {core1:INWARD_STD core2:INWARD_STD}
define_test_mode CORES_INWARD_COMP -usage scan_compression \
  -target {core1:INWARD_COMP core2:INWARD_COMP}

# define modes that test only top-level logic
define_test_mode TOPONLY_STD -usage scan \
  -target {top}
define_test_mode TOPONLY_COMP -usage scan_compression \
  -target {top}

# define flat test modes that test the entire design
define_test_mode TOPFLAT_STD -usage scan \
  -target {core1:TRANS_STD core2:TRANS_STD top}
define_test_mode TOPFLAT_COMP -usage scan_compression \
  -target {core1:TRANS_COMP core2:TRANS_COMP top}
```

The tool automatically identifies the standard scan or compressed scan top-level-only mode associated with each flat mode.

Do not apply any DFT configuration commands specifically to the flat test modes; they are ignored.

Limitations

Note the following limitations of top-down flat testing using transparent core-level test modes:

- DFTMAX Ultra compression is not supported.
- You must define core-level transparent and top-level flat test modes with the `define_test_mode` command; they are not created by default.
- At the core level, to create transparent modes, you must use dedicated wrapper scan I/Os in the outward-facing modes.
- At the top level, you cannot mix transparent and nontransparent core test modes in the same test mode.
- At the top level, you cannot define multiple flat top-level scan compression modes.

SCANDEF Generation for Wrapper Chains

SCANDEF generation is supported for wrapper chains. The SCANDEF information is generated as follows:

- Wrapper cells can be reordered within wrapper chains.
- Wrapper chain cells cannot be repartitioned with regular scan chain cells.
- Input wrapper cells can be repartitioned between input wrapper chains, and output wrapper cells can be repartitioned between output wrapper chains.
- If input and output wrapper cell mixing is enabled, input wrapper chain cells can be repartitioned with output wrapper chain cells.

Input and output wrapper chain mixing is enabled by default in the simple wrapper flow and disabled by default in the maximized reuse flow. You can change this setting with the `-mix_cells` option of the `set_wrapper_configuration` command.

- Each wrapper cell is represented as an ORDERED construct that ensures all logic gates for that wrapper cell are kept together. However, a shared wrapper cell in the maximized reuse flow is simply a scan cell, so it is represented as an individual scan cell instead of an ORDERED construct.

See Also

- [Chapter 12, Generating SCANDEF for Scan Reordering in Layout](#) for more information about generating SCANDEF information

Limitations of Core Wrapping

Note the following limitations related to the behavior of the scan-enable signal in capture mode:

- Preexisting scan-enable control of clock-gating cells

If the design has I/O registers controlled by clock-gating cells that are controlled by the scan-enable port, these I/O registers might not shift during the capture mode operation of the INTEST and EXTEST modes because scan-enable might not be active in capture mode.

- Preexisting scan-enable control of set and reset

If the set or reset pins of the I/O registers are controlled by the scan-enable port, the I/O registers might not shift during the capture mode of operation of the INTEST and EXTEST modes because scan-enable might not be active in capture mode.

10

Clock Gating

To reduce switching power consumption, Fusion Compiler inserts *integrated clock-gating cells* for banks of functional registers that conditionally hold their state. These clock-gating cells have test pins that ensure proper clocking behavior during test.

The following topics provide more information:

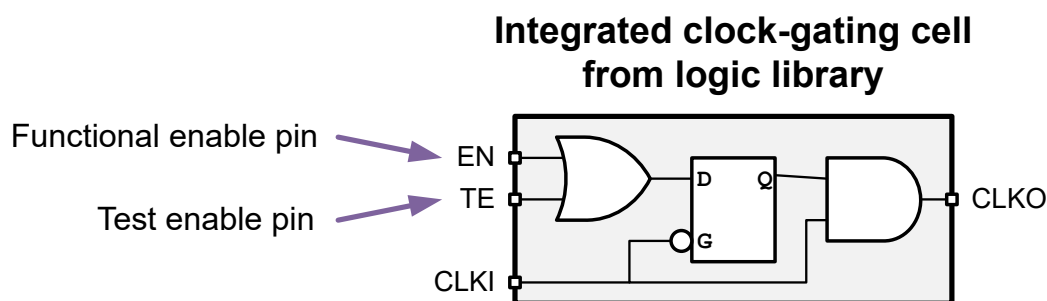
- [Clock Gating Concepts](#)
- [Configuring Clock-Gating for DFT](#)
- [Clock Gating for DFT Wrapper Cells](#)

Clock Gating Concepts

During scan shift, all registers must receive the clock signal. Correspondingly, integrated clock-gating cells have two enable pins:

- The functional enable pin is driven by functional logic; it enables the clock during functional operation.
- The test enable pin is driven by test logic; it enables the clock during test operation.

The signals are combined inside the clock-gating cell so that either one enables the clock:

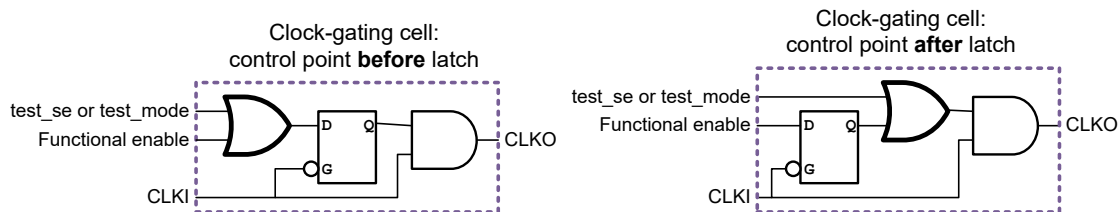


Clock-Gating Test Control Points

In a clock-gating cell, the *control point* is the logic gate that allows the clock to be enabled when its test pin is asserted. There are two degrees of freedom in control point implementation, also shown in [Figure 67](#):

- The control point can be placed before or after the latch element.
- The control signal can be driven by the scan-enable or test-mode signal.

Figure 67 Clock-Gating Control Points Before and After Latch



For most designs, the “before” control point style driven by the scan-enable signal is recommended for the following reasons:









- The “before” control point ensures that no combinational path exists from the control signal input port to the downstream clock pins of the scan cells.
- If the “after” control point is used and both phases of the clock are gated in the design, there is no time at which the control signal can cleanly toggle without truncating an active clock pulse.
- A scan-enable control signal ensures that the gated clocks are always-active during scan shift, but that the functional clock-gating paths can still be tested during scan capture.
- A test-mode control signal prevents the functional clock-gating paths from being tested, requiring additional testability logic to be inserted in the design.

Most integrated clock-gating cells are implemented using control points before the latch.

Choosing a Clock-Gating Control Point Configuration

[Table 5](#) shows the possible combinations of clock gating, clock waveforms, control signals, and control point location you can use.

Table 5 *Latched-Based Clock-Gating Configurations*

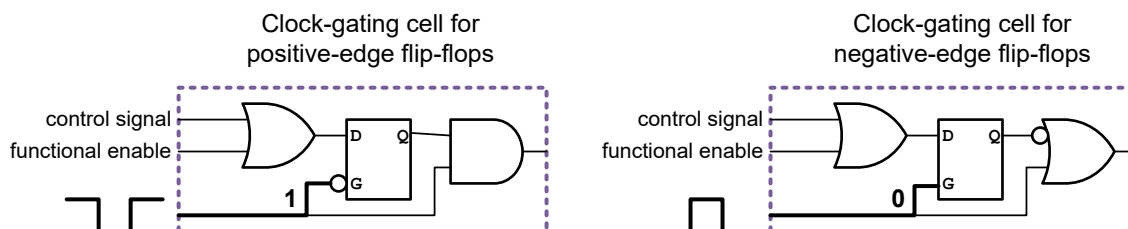
Clock-gating style	Clock waveform	Control signal	Control point location	Gated register can be scanned?
Latch-based gating for positive-edge flip-flops		test_mode	Before latch	Yes
			After latch	Yes
		scan_enable	Before latch	Yes
			After latch	Yes
		test_mode	Before latch	Yes, but requires extra initialization cycles to be manually specified for the test protocol.
			After latch	Yes
		scan_enable	Before latch	Yes, but supports only one level of clock gating.
			After latch	No
Latch-based gating for negative-edge flip-flops		test_mode	Before latch	Yes
			After latch	Yes
		scan_enable	Before latch	Yes
			After latch	Yes
		test_mode	Before latch	Yes, but requires extra initialization cycles to be manually specified for the test protocol.
			After latch	Yes
		scan_enable	Before latch	Yes, but supports only one level of clock gating.
			After latch	No

For each combination, the last column indicates whether the gated register can be included in scan chains. Four special cases, marked with footnotes, require additional consideration as described in the following section.

Initialization for Special Cases of Before-Latch Control Points

In the four special cases marked by footnotes in , the gating latch is inactive at the beginning of the test program (time = 0). This unknown latch state causes an X value to reach the gated flip-flops, which would normally prevent them from being included in scan chains. Figure 68 shows the clock waveform and clock-gating logic for these cases.

Figure 68 Special Clock-Gating Cases With Inactive Latch at Beginning of Test Clock Cycle



If you are using a scan-enable control signal, which asserts and de-asserts during every test pattern, the DRC engine performs an analysis of the clock-gating logic to verify a known state in the latch. This analysis supports only one such level of special-case clock gating, although you can have additional levels of non-special-case clock gating.

If you are using a test-mode control signal, to achieve a known state in the latch, you must add a clock pulse to the `test_setup` section of the test protocol. Use the following `set_dft_drc_configuration` command to update the `test_setup` section with the clock pulse:

```
fc_shell> set_dft_drc_configuration -clock_gating_init_cycles 1
```

If you have multiple cascaded latch-based clock-gating cells and the first latch is loaded with the test-mode signal, use the following `set_dft_drc_configuration` command to update the `test_setup` section with the specified number of clock pulses:

```
fc_shell> set_dft_drc_configuration -clock_gating_init_cycles n
```

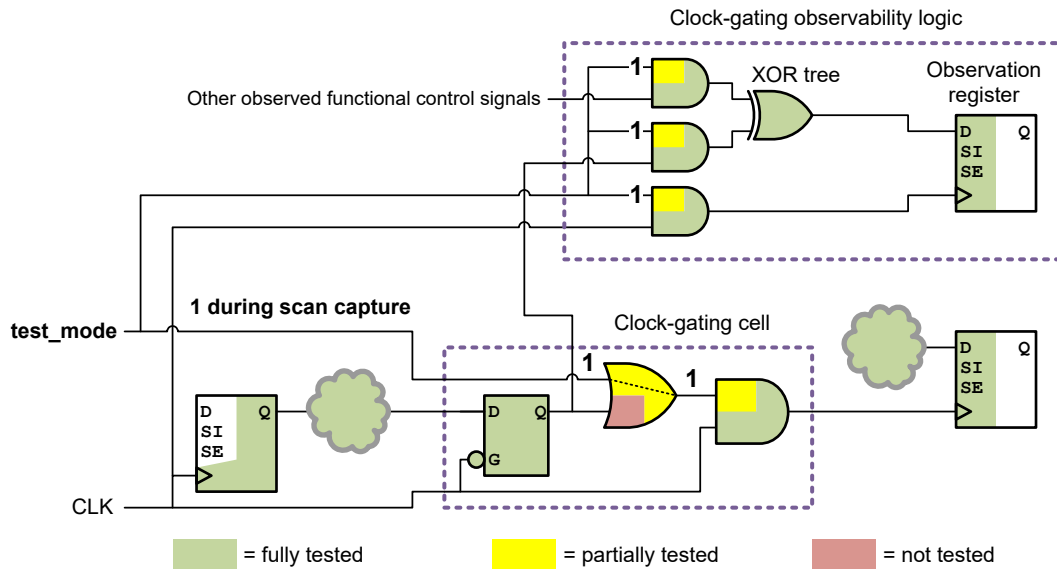
Here, *n* equals the number of clock pulses required to initialize clock-gating latches.

Note the following:

- The set of clock cycles should equal the depth of the chain of clock-gating latches. Make sure this is the case.
- Violations still occur when there are multiple cascaded latches and the scan-enable and control point location are used as before, with mixed active-high and active-low latches.

If your clock-gating cell has an observe pin, Fusion Compiler inserts observability logic to regain some of this coverage. Figure 71 shows an observability logic example.

Figure 71 Observability Logic for Clock-Gating Cells



In test mode, an XOR tree observes the functional gating control signals from one or more clock-gating cells; the output of the XOR tree is captured by a dedicated observability register. This observability register is included in the scan chains, but its output is not otherwise connected functionally in the design. AND or NAND gates (depending on synthesis) prevent the observability logic from consuming power during mission mode.

Note that placing the control point after the latch allows the latch to be tested by the observability logic.

Configuring Clock-Gating for DFT

In Fusion Compiler, clock gating is always enabled. During compile, conditional-capture register banks in the RTL are identified and gated. In addition, the test pins of clock-gating cells (both tool-inserted and instantiated) are connected to the scan-enable signal. No further configuration is needed.

However, the DFT implementation for clock gating can be customized as follows.

Specifying Clock-Gating Control Signals

By default, Fusion Compiler chooses an available ScanEnable signal to use for the clock-gating control signal. If this is sufficient, no further configuration is needed.

However, you can also define particular clock-gating control signals to meet your own design requirements.

Specifying a Particular Global Clock-Gating Control Signal

To specify a particular scan-enable or test-mode signal to use as the clock-gating control signal, define that signal with the `clock_gating` usage:

```
set_dft_signal
  -type ScanEnable | TestMode
  -view spec
  -usage clock_gating
  -port port_list
```

The `-type` option specifies the type of clock-gating control signal you want to define.

Remember that when you define a DFT signal with a usage, the tool is limited to using that signal only for that usage.

Specifying Object-Specific Clock-Gating Control Signals

You can also define clock-gating control signals for specific parts of the design by using the `-connect_to` option of the `set_dft_signal` command:

```
set_dft_signal
  -type ScanEnable | TestMode
  -view spec
  -usage clock_gating
  -port port_list
  [-connect_to object_list]
```

The `-connect_to` option specifies a list of design objects that are to use the specified clock-gating control signal. The supported object types are

- Clock-gating cells (instantiated)
- Hierarchical cells
- Designs
- Test clock ports

This allows you to make clock-domain-based signal connections. It includes clock-gating cells that gate the specified test clocks. The functional clock behavior is not considered.

This specification requires that a functional clock also be defined on the test clock port.

- Scan-enable or test-mode pins of CTL-modeled cores

The following example defines a ScanEnable signal named SE_CG to connect to the test pins of existing clock-gating cells ICG_CLK100 and ICG_CLK200:

```
fc_shell> set_dft_signal \  
-type ScanEnable -view spec -port SE_CG \  
-usage clock_gating -connect_to {ICG_CLK100 ICG_CLK200}
```

Excluding Clock-Gating Cells From Test-Pin Connection

You might have a portion of the design that is excluded from scan testing, and you do not want Fusion Compiler to connect the test pins of those clock-gating cells to test-mode or scan-enable signals. To prevent the tool from connecting the test pins of some clock-gating cells, use the `-exclude_elements` option of the `set_dft_clock_gating_configuration` command:

```
fc_shell> set_dft_clock_gating_configuration  
-exclude_elements object_list
```

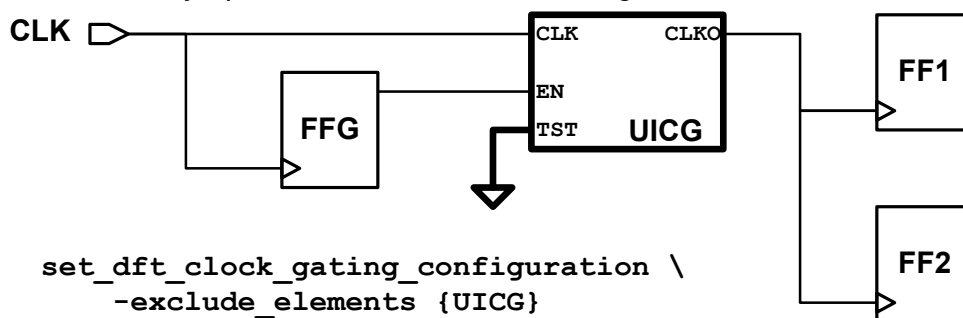
The *object_list* can include the following object types:

- Clock-gating cell leaf instances (instantiated in RTL)
- Hierarchical cell instances – All clock-gating cells within the instances are included in the specification.
- Clock-gating library cell – All instances of that cell are included in the specification.
- Clocks – All clock-gating cells in the clock domains are included in the specification.

For tool-inserted clock-gating cells, the tool ties the test pin to a de-assertion constant. For RTL-instantiated clock-gating cells, the tool leaves the existing RTL connection in place.

Figure 72 shows an example of an excluded clock-gating cell.

Figure 72 Directly Specified Excluded Clock-Gating Cell



Clock Gating for DFT Wrapper Cells

Clock gating enables the clock gating engine to gate wrapper cells and internal cells with separate clock gates. The power consumption in DFT EXTEST mode can be reduced by turning off the clock to the internal cells.

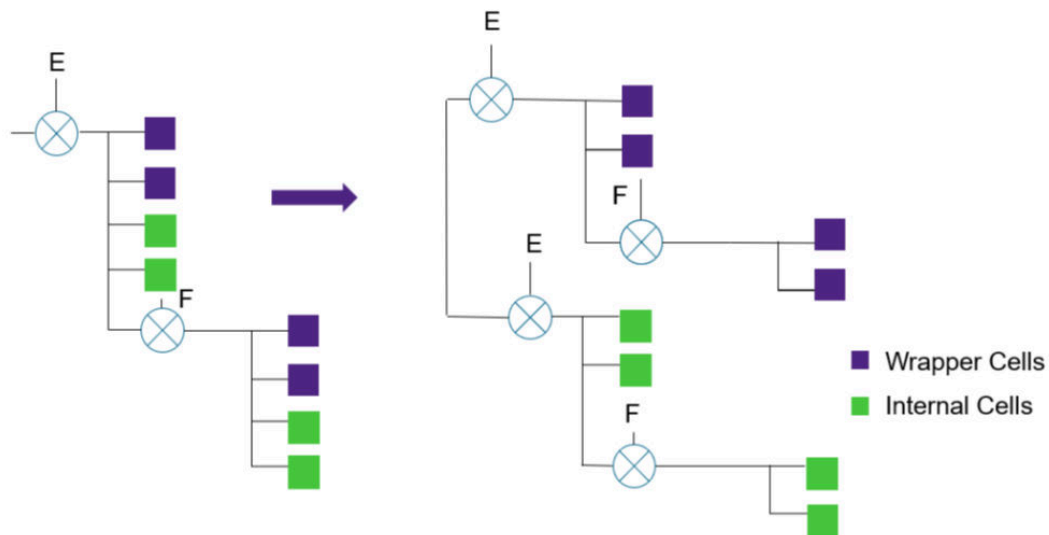
The following topics provide more information on clock gating:

- [Clock Gating](#)
- [Limitations](#)

Clock Gating

When you do not specify the `max_fanout` option, the Fusion Compiler tool inserts one clock gate for every unique enable condition, by default. In DFT wrapper clock gating, the clock gate is split into two. One clock gate is used for the wrapper cells, and the other is used for the internal cells, as shown in the following figure.

Figure 73 DFT wrapper clock gating



When you split the clock gating signal to gate wrapper cells and internal cells separately, the resultant clock gates and the wrapper cells that were split away are marked such that they can be queried independently.

To enable DFT wrapper clock gating, set the `clockgate.enable_dft_wrapper_cell_support` application option to `true`.

Limitations

If the `depth_threshold` option is used, then clock gating might not work after using the `insert_dft` command.

11

The Internal Pins Flow

The internal pins flow allows you to define certain DFT signals on internal pins instead of top-level ports. This allows more complicated test logic to be inserted, but it also comes with some caveats that must be considered.

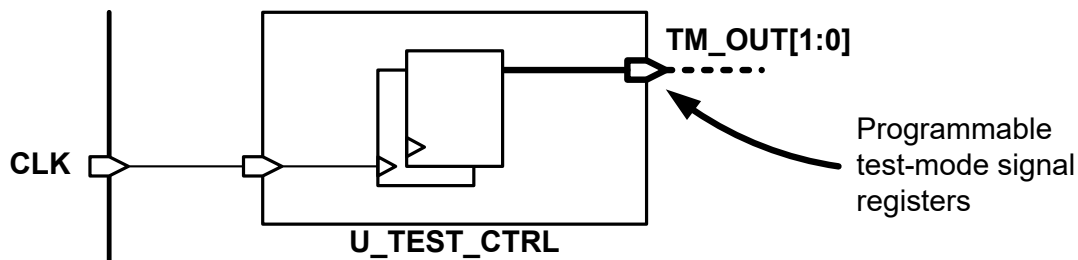
The following topics provide more information:

- [Introduction to the Internal Pins Flow](#)
- [Defining Signals on Internal Pins](#)
- [Writing Out the Test Protocol](#)
- [Limitations](#)

Introduction to the Internal Pins Flow

Normally, DFT signal (such as clocks, resets, scan-ins, and scan-outs) are defined on ports of the current design. Even when defined with a hookup pin, these signals are ultimately connected to a port.

However, advanced DFT architectures might require some DFT signals to connect to an internal pin, with no straightforward path to a port. The following figure shows programmable registers intended to drive TestMode signals:



In these cases, you can use the *internal pins* flow to define such signals. It is called a flow because it requires consideration both before and after DFT insertion, as described below.

Defining Signals on Internal Pins

To define signals on internal pins, do the following:

1. In your global DFT configuration, enable the internal pins flow:

```
set_dft_drc_configuration -internal_pins enable
```

2. When defining your DFT signals, define each internal-pins signal by using the `-hookup_pin` option *without* the `-port` option:

```
set_dft_signal -view spec -type TestMode \  
-hookup_pin U_TEST_CTRL/TM_OUT[1]  
set_dft_signal -view spec -type TestMode \  
-hookup_pin U_TEST_CTRL/TM_OUT[0]
```

The `-hookup_pin` option accepts only a single pin object; use multiple commands for multiple pins.

You can define a mix of port-driven and internal-pins signals as needed.

The following signal types can be defined as internal-pins signals:

Table 6 *Supported Internal-Pins Signal Types*

ScanMasterClock
MasterClock
ScanClock
Reset
Constant
TestMode
ScanEnable
ScanDataIn
ScanDataOut
PLLClock
occ_reset (or pll_reset)
occ_bypass (or pll_bypass)
LOSPipelineEnable

Writing Out the Test Protocol

After DFT insertion, the CTL model for the design contains information about internal pins. This information is understood by Fusion Compiler, but it cannot be used by TestMAX ATPG.

When you write out a test protocol using the `write_test_protocol` command, the resulting SPF references these internal pins. Before using the protocol in TestMAX ATPG, you must remove the internal pins references, then make any modifications needed for the assumptions in the test protocol to be true.

Limitations

Note the following limitations of the internal pins flow:

- Post-DFT DRC is not supported for compressed scan designs with internal-pins signals.
- In most cases, the output protocol must be modified before use in the TestMAX ATPG tool.
- The core wrapper flow does not support internal-pins signals.

See Also

- [SolvNetPlus article 040136](#), “Using the Internal Pins Flow With Internal Test Registers” for an example that drives DFT signals from design registers

12

Generating SCANDEF for Scan Reordering in Layout

Fusion Compiler can generate SCANDEF information that describes how scan cells in the design can be reordered and repartitioned. You can use this SCANDEF information in the Fusion Compiler tool to optimize scan chains and to fix timing violations using physical information. You can also use this information in other place-and-route tools.

This topic covers the following:

- [Introduction to SCANDEF](#)
- [SCANDEF Constructs](#)
- [Writing Out the SCANDEF Information](#)
- [Limitations of SCANDEF Generation](#)

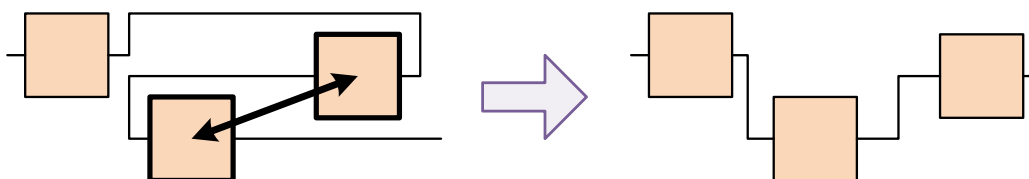
Introduction to SCANDEF

Scan-inserted designs require additional routing compared to nonscan designs. To meet die size and timing requirements, you should reduce the routing overhead as much as possible. One way to do this is to optimize scan chains based on physical information.

There are two types of scan optimization operations:

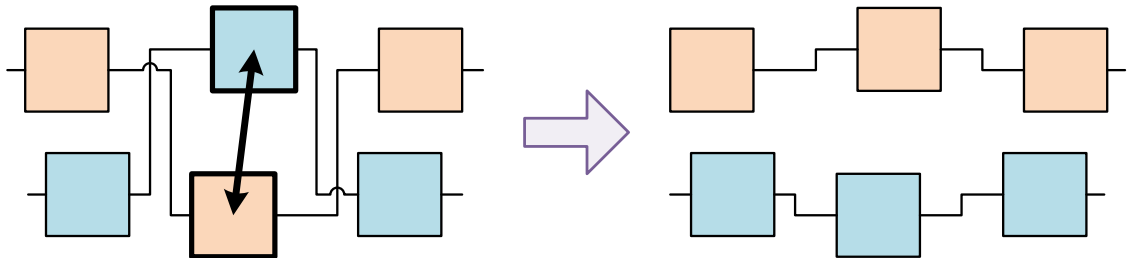
- *Scan reordering* changes the scan chain position of two (or more) scan cells within the same scan chain. [Figure 74](#) shows a scan reordering example.

Figure 74 Scan Reordering of Two Scan Cells



- *Scan repartitioning* swaps two (or more) scan cells between different scan chains, such that the original chain lengths are preserved. [Figure 75](#) shows a scan repartitioning example.

Figure 75 Scan Repartitioning of Two Scan Cells



During scan reordering and repartitioning, the layout tool must honor DFT constraints such as clock mixing, DFT partitions, multivoltage regions, and multiple test modes. However, communicating this information directly to the layout tool would be complex and error-prone.

Instead, SCANDEF communicates what reordering and repartitioning operations are possible given the DFT constraints. As a result, the layout tool does not need to understand DFT constraints; it simply optimizes as allowed by the SCANDEF.

SCANDEF Constructs

A SCANDEF file is a DEF file that uses a set of scan-specific constructs to describe scan chain information. A *stub chain* definition describes a portion of a scan chain that can be reordered within itself. A stub chain consists of a START point, a STOP point, and one or more scan elements between them. [Example 8](#) shows the first three stub chain definitions in a SCANDEF file.

Example 8 SCANDEF Example

```

VERSION 5.5 ;
NAMECASESENSITIVE ON ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN top ;
SCANCHAINS 8 ;
- 1_clk_st_45_55_DEFAULT_POWER_DOMAIN_VDD_internal
+ PARTITION clk_st_45_55_DEFAULT_POWER_DOMAIN_VDD_internal
  MAXBITS 8
+ START U131 Y
+ FLOATING ZN_reg[0] ( IN SI ) ( OUT Q )
              ZN_reg[1] ( IN SI ) ( OUT Q )
              ZN_reg[2] ( IN SI ) ( OUT Q )
              ZN_reg[3] ( IN SI ) ( OUT Q )
              ZN_reg[4] ( IN SI ) ( OUT Q )

```

Chapter 12: Generating SCANDEF for Scan Reordering in Layout

SCANDEF Constructs

```

+ PARTITION CLK_45_45
+ STOP ZN_reg[4] SI ;
- 2_clk_st_45_55_DEFAULT_POWER_DOMAIN_VDD_internal
+ PARTITION clk_st_45_55_DEFAULT_POWER_DOMAIN_VDD_internal
MAXBITS 8
+ START U132 Y
+ FLOATING ZN_reg[5] ( IN SI ) ( OUT Q )
+ ORDERED SR_reg[3] ( IN SI ) ( OUT Q )
          SR_reg[2] ( IN D ) ( OUT Q )
          SR_reg[1] ( IN D ) ( OUT Q )
          SR_reg[0] ( IN D ) ( OUT Q )
+ PARTITION CLK_45_45
+ STOP ZN_reg[4] SI ;
- 3_clk_st_45_55_DEFAULT_POWER_DOMAIN_VDD_internal
+ PARTITION clk_st_45_55_DEFAULT_POWER_DOMAIN_VDD_internal
  MAXBITS 8
+ START test_si2
+ FLOATING IP_inst ( IN test_si1 ) ( OUT test_so1 ) (BITS 5)
          IPglue_logic_cell1 ( IN TI ) ( OUT SO )
          IPglue_logic_cell2 ( IN TI ) ( OUT SO )
+ PARTITION IPCLK_45_45
+ STOP test_so2 ;
...

```

Note the following constructs:

- The START and STOP points specify the stub chain boundaries. They can be a variety of scan chain constructs such as scan I/O ports, codec logic gates, lockup latches, reconfiguration MUXs, or buffer/inverter pins. Therefore, stub chains are not usually identical to scan chains, and the number of stub chains defined in the SCANDEF information does not necessarily match the number of scan chains in the design.
- A FLOATING section is a list of unordered cells that can be reordered freely by the layout tool. Because the cells are described as an unordered list, the order in the SCANDEF file might not match the order in the design.
- An ORDERED section describes a group of scan cells that cannot be reordered within that group, but can be reordered as a group within a stub chain. Common causes of ORDERED sections are shift registers identified by the `compile_fusion` command, scan segments defined with the `set_scan_path -ordered_elements` command, and buffers or inverters between scan cells.
- The BITS attribute indicates a scan element that represent multiple scan bits. This allows complex scan cells, such as DFT-inserted cores, to be represented in abstract form. By default, each individual scan element represents a single scan bit.
- A PARTITION name indicates that the stub chain elements can be repartitioned with those of another stub chain with the same partition name. The tool constructs

partition names so that identical names indicate stub chains that are compatible for repartitioning. Partition names are made unique or omitted for stub chains whose elements cannot be repartitioned.

A stub chain can include zero or more ORDERED sections. However, it can only contain zero or one FLOATING section, as having multiple FLOATING sections within the same stub chain is meaningless.

A SCANDEF file does not necessarily contain all scan cells in the design. It contains information only about scan cells in the design that *can* be reordered or repartitioned. Scan cells or scan segments that cannot be optimized are omitted from the file.

The layout tool can reorder and/or repartition many scan cells at a time. For example, several compatible scan chains in a geographic region can be completely reconstructed, if the SCANDEF information is honored and the original scan chain lengths are preserved.

Writing Out the SCANDEF Information

To generate SCANDEF information, perform the following steps after reading in your design and applying the DFT configuration:

1. Execute the `compile_fusion` command.
2. Generate the SCANDEF information with the `write_scan_def` command:

```
fc_shell> write_scan_def -output filename.scandef
```

This command writes out the SCANDEF information to the specified file name.

You can use the resulting SCANDEF information in your place-and-route tool to optimize scan chain routing order. In the Fusion Compiler tool, use the following command to read in the SCANDEF file:

```
read_def filename.scandef
```

When you read SCANDEF information into the Fusion Compiler tool, it checks the integrity of the information against the design netlist before using it.

The SCANDEF must be read into the Fusion Compiler tool using this command; it is not stored in the NDM database.

Limitations of SCANDEF Generation

Note the following limitations of SCANDEF generation:

- Manual post-DFT modifications to the scan structure or scan element naming are not supported.

Manual post-DFT design modifications that affect the scan architecture or scan element naming are not supported. Examples include:

- Hierarchy uniquification or ungrouping
- ECO modification to scan chains or logic that affects DFT operation
- Modifying IEEE 1801 Standard (UPF) power intent to insert isolation and/or level-shifter cells after DFT insertion
- For stub chains that terminate at a reconfiguration MUX, the STOP pin is the scan-in pin of the last scan cell instead of the input pin of the reconfiguration MUX. This prevents the last scan cell in the stub chain from being reordered or repartitioned.
- `set_scan_group` specifications might not be represented properly.
- Some `set_scan_path` specifications, when applied to a specific test mode other than the first-defined test mode, are not represented properly.

For details, see [SolvNetPlus article 2314593](#), “SCANDEF Generation Limitations for Multiple Test Modes.”

13

On-Chip Clocking (OCC)

On-chip clocking (OCC) support is common to all scan ATPG (basic-scan and fast-sequential) and compressed scan environments. This implementation is intended for designs that require ATPG in the presence of phase-locked loop (PLL) and clock controller circuitry.

OCC support includes phase-locked loops, clock shapers, clock dividers and multipliers and so on. In the scan-ATPG environment, scan chain `load_unload` is controlled through an automatic test equipment (ATE) clock. However, internal clock signals that reach state elements during capture are PLL-related.

OCC flows can use either the user-defined clock controller and clock chains or the DFT-inserted OCC clock controller. If you use an existing user-defined clock controller, you would need a set of user-defined commands to identify the existing clock controller outputs with their corresponding clock chain control bits.

This chapter includes the following topics:

- [Background](#)
- [Clock Type Definitions](#)
- [OCC Controller Structure and Operation](#)
- [Enabling On-Chip Clocking Support](#)
- [Specifying OCC Controllers](#)
- [Reporting Clock Controller Information](#)
- [Controlling the OCC Controller Bypass Path](#)
- [DFT-Inserted OCC Controller Configurations](#)
- [Waveform and Capture Cycle Example](#)
- [Limitations](#)

Background

At-speed testing for deep-submicron defects requires not only more complex fault models for ATPG and fault simulation, such as transition faults and path delay faults, but also requires the accurate application of two high-speed clock pulses to apply the tests for these fault models. The time delay between these two clock pulses, referred to as the launch clock and the capture clock, is the effective cycle time at which the circuit will be tested.

A key benefit of scan-based at-speed testing is that only the launch clock and the capture clock need to operate at the full frequency of the device under test. Scan shift clocks and shift data can operate at a much slower speed, thus reducing the performance requirements of the test equipment. However, complex designs often have many different high-frequency clock domains, and the requirement to deliver a precise launch and capture clock for each of these from the tester can add significant or prohibitive costs to the test equipment. Furthermore, special tuning is often required for properly controlling the clock skew to the device under test.

One common alternative for at-speed testing is to leverage existing on-chip clock generation circuitry. This approach uses the active controller, rather than off-chip clocks from the tester, to generate the high-speed launch and capture clock pulses. This type of approach generally reduces tester requirements and cost and can also provide high-speed clock pulses from the same source as the device in its normal operating mode without additional skews from the test equipment or test fixtures.

When using this approach, additional on-chip controller circuitry is inserted to control the on-chip clocks in test mode. The on-chip clock control is then verified, and at-speed test patterns are generated that apply clocks through proper control sequences to the on-chip clock circuitry and test-mode controls. The Fusion Compiler and TestMAX ATPG tools support a comprehensive set of features to ensure that

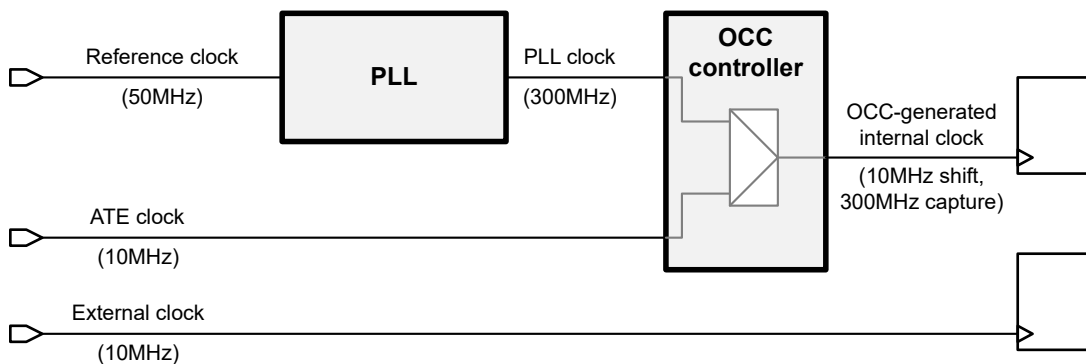
- The test-mode control logic for the OCC controller operates correctly and has been connected properly
- Test-mode clocks from the OCC circuitry can be efficiently used by TestMAX ATPG for at-speed test generation
- OCC circuitry can operate asynchronously to other shift clocks from the tester
- TestMAX ATPG patterns do not require additional modifications to use the OCC and to run properly on the tester

Clock Type Definitions

Note the following clock definitions as they apply to OCC controller clocks in this chapter. [Figure 76](#) shows an example of each clock type.

- **Reference clock** – The frequency reference to the phase-locked loop (PLL). It must be maintained as a constantly pulsing and free-running oscillator, or the circuitry will lose synchronization.
- **PLL clock** – The output of the PLL. It is also a free-running source that runs at a constant frequency that might or might not be the same as the reference clock.
- **ATE clock** – Shifts the scan chain, typically more slowly than a reference clock. This signal must preexist, or you must manually add this signal (that is, port) when inserting the OCC. The period for this clock is determined by the `dft.test_default_period` application option. Usually the ATE clock is not used as a reference clock, but it must be treated as a free-running oscillator so that it does not capture predictable data while the OCC controller generates at-speed clock pulses. The ATE clock is called a dual clock signal when the same port drives both the ATE clock and the reference clock.
- **Internal clock** – The OCC controller is responsible for gating and selecting between the PLL and ATE clocks, thus creating the internal clock signal to satisfy ATPG requirements.
- **External clock** – A primary clock input of a design that directly clocks flip-flops through the combinational logic.

Figure 76 Clock Types Used in the OCC Controller Flow



OCC Controller Structure and Operation

OCC controller types and operation is covered in the following topics:

- [DFT-Inserted and User-Defined OCC Controllers](#)
- [OCC-Controlled Clock Relationships](#)
- [OCC Controller Signal Operation](#)
- [Clock Chain Operation](#)
- [Logic Representation of an OCC Controller and Clock Chain](#)
- [Scan-Enable Signal Requirements for OCC Controller Operation](#)

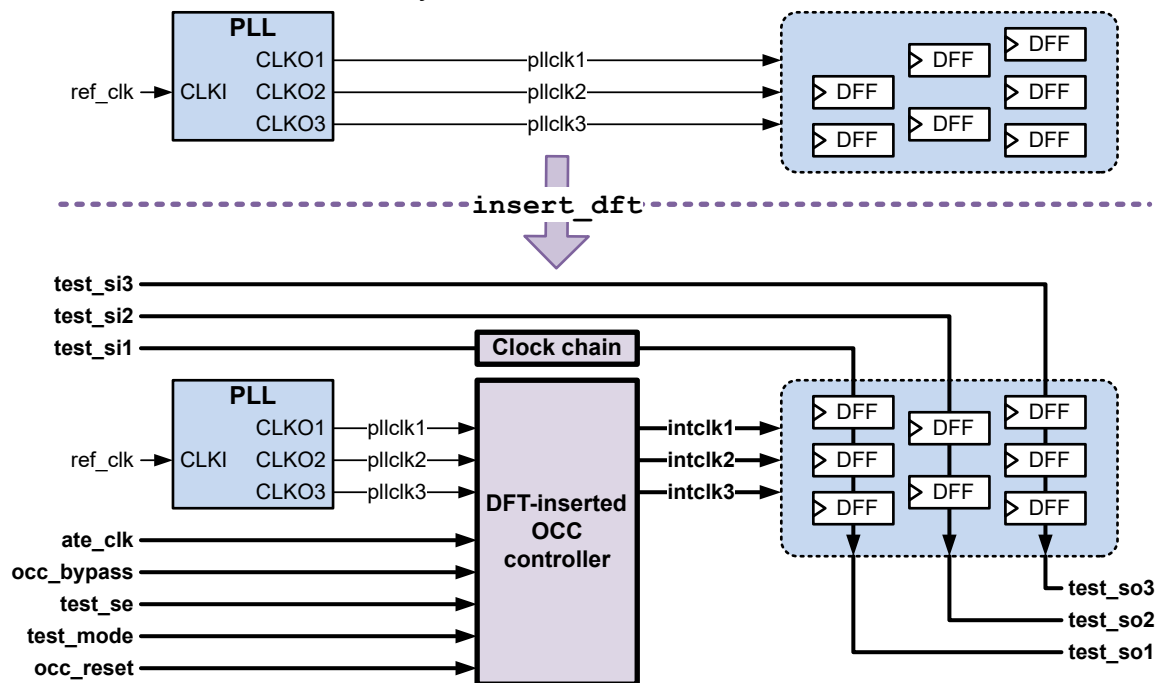
DFT-Inserted and User-Defined OCC Controllers

You can use DFT-inserted or user-defined OCC controllers, as described in the following flows:

- [Specifying DFT-Inserted OCC Controllers](#)

The `insert_dft` command inserts a DFT-inserted OCC controller and clock chain, making control signal connections and modifying the clock signal connections as needed. The OCC controller design is validated and incorporated into the resulting test protocol. This flow is shown in [Figure 77](#).

Figure 77 OCC and Clock Chain Synthesis Insertion Flow

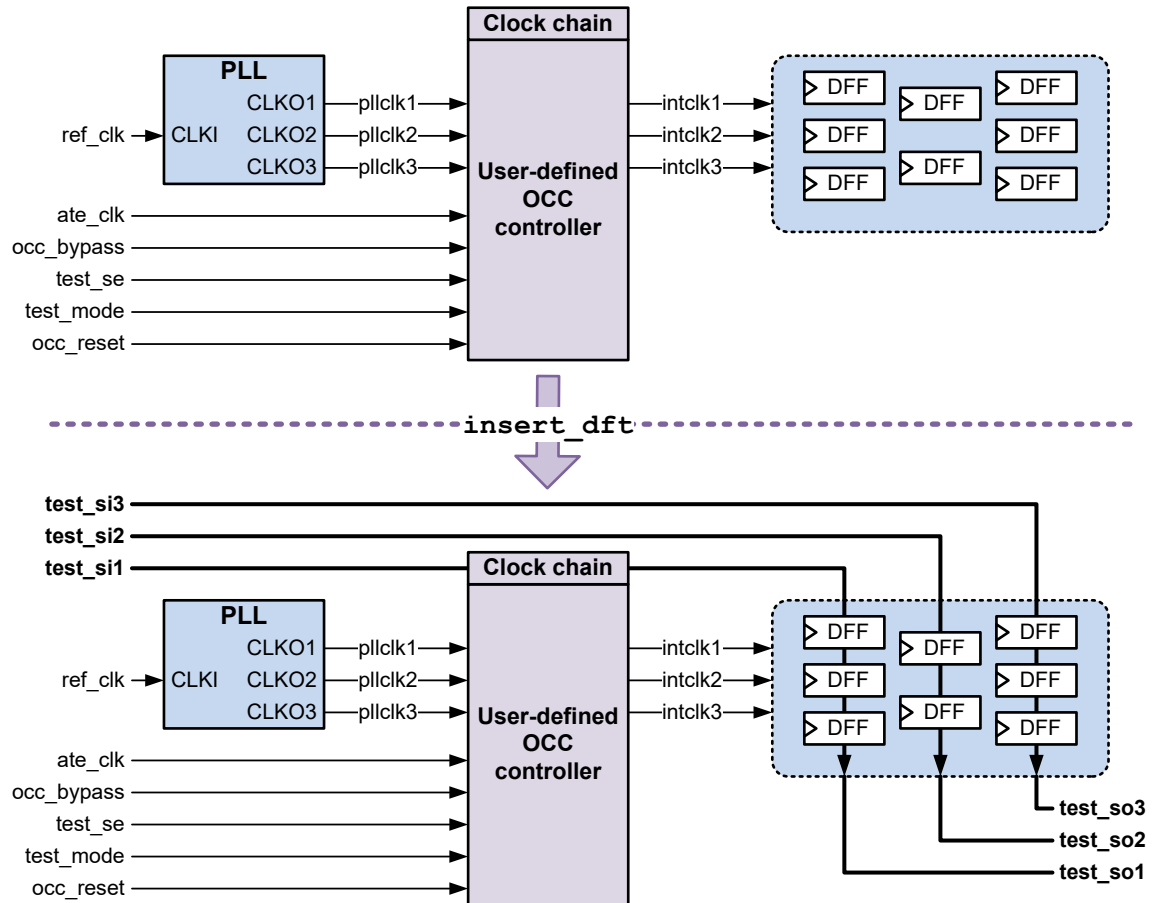


- [Specifying Existing User-Defined OCC Controllers](#)

The RTL contains the OCC controller and clock chain logic, all control signal and clock signal connections, and the connections from the clock chain to the OCC controller. Before DFT insertion, this existing user-defined OCC controller and clock chain logic is described to the tool using the `set_dft_signal` and `set_scan_group` commands. The

OCC controller design is validated and incorporated into the resulting DFT logic and test protocol. This flow is shown in [Figure 78](#).

Figure 78 User-Defined Clock Controller and Clock Chain Flow

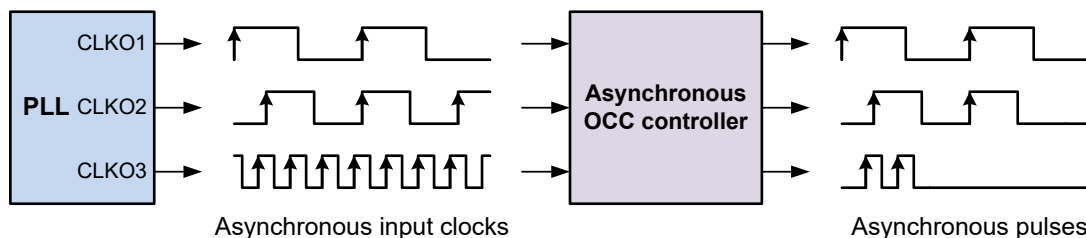


Note that in this flow, the tool does not make any signal connections to the OCC controller or clock chain logic, except for scan data connections to the clock chain segments. You must ensure that all clock and control signal connections exist prior to DFT insertion.

OCC-Controlled Clock Relationships

Each controlled clock uses dedicated clock control logic that generates clock pulses with no regard to alignment with other controlled clocks, as shown in [Figure 79](#). The input clocks being controlled can be synchronous or asynchronous with each other.

Figure 79 Asynchronous OCC Controller Generating Two Clock Pulses



ATPG treats the controlled clock domains as asynchronous; faults between them are not tested. You can use an OCC controller to control synchronous clocks, but their synchronous relationships are lost; there is no guarantee that their pulse sequences will initiate on the same rising edge.

OCC Controller Signal Operation

Note the following:

- The reference clock (refclk) is always free-running. It is used as a test default frequency input to the PLL.
- The PLL clocks (pllclk1, pllclk2, and pllclk3) are free-running clock outputs from the on-chip clock generator; they can be divided, shaped, or multiplied. They are used for the launch and capture of internal scanable elements that become internal clocks.
- The ATE clock (ate_clk) shifts the scan chain per tester specifications. Each PLL might have its own ATE clock.

See [Waveform and Capture Cycle Example](#) for a waveform diagram that demonstrates the relationship between the various clocks.

- The OCC controller serves as an interface between the on-chip clock generator and internal scan chains. This logic typically contains clock multiplexing logic that allows internal clocks to switch from a slow ATE clock during shift to a fast PLL clock during capture.
- Internal clocks (intclk1, intclk2, and intclk3) are outputs of the PLL control logic driving the scan cells. Each internal clock is controlled or enabled by the clock chain and is connected to the sequential elements within the design.
- The OCC bypass signal (occ_bypass) allows the ATE clock signal to connect directly to the internal clock signals, thus bypassing the PLL clocks.
- The ScanEnable signal (test_se) enables switching between the ATE shift clock and output PLL clock signals. ScanEnable must be inactive during every capture procedure, as described in [Scan-Enable Signal Requirements for OCC Controller Operation](#). You can use individual ScanEnable signals for each PLL clock signal.

- The OCC TestMode signal (test_mode) must be asserted in order for the clock controller to operate.
- The OCC reset signal (occ_reset) is asserted during test setup to reset the OCC controller flip-flops to their initial states.

Clock Chain Operation

The clock chain provides a per-pattern clock selection mechanism for ATPG. It is implemented as a scan chain segment of one or more scan cells. Clock selection values are loaded into the clock chain as part of the regular scan load process.

A clock chain operates as follows:

- During scan shift, the clock chain shifts in new values when clocked by a scan clock.

The clock chain can be clocked by either the rising or falling clock edge, depending on what best fits into the overall DFT architecture.

- During scan capture, the clock chain holds its value.

The value scanned into the clock chain must be scanned out, undisturbed, after capture. The clock controller inserted by Fusion Compiler meets this requirement. If you provide your own clock controller, ensure that it meets this requirement.

Note the following scan architecture aspects of clock chains:

- For standard scan designs, the clock chain can be a dedicated scan chain or a segment within a scan chain.
- For DFTMAX compressed scan designs, the clock chain can be an uncompressed (external) scan chain or a special segment within a compressed scan chain. For more information, see [Scan Compression and OCC Controllers](#).
- For DFTMAX Ultra compressed scan designs, the clock chain must be an uncompressed (external) scan chain. For more information, see [Using OCC Controllers With DFTMAX Ultra Compression](#).
- Clock chains of the same type (compressed or external) can be concatenated together.

Compressed clock chains are concatenated into a single chain and placed inside the compressor where a regular single chain would be placed.

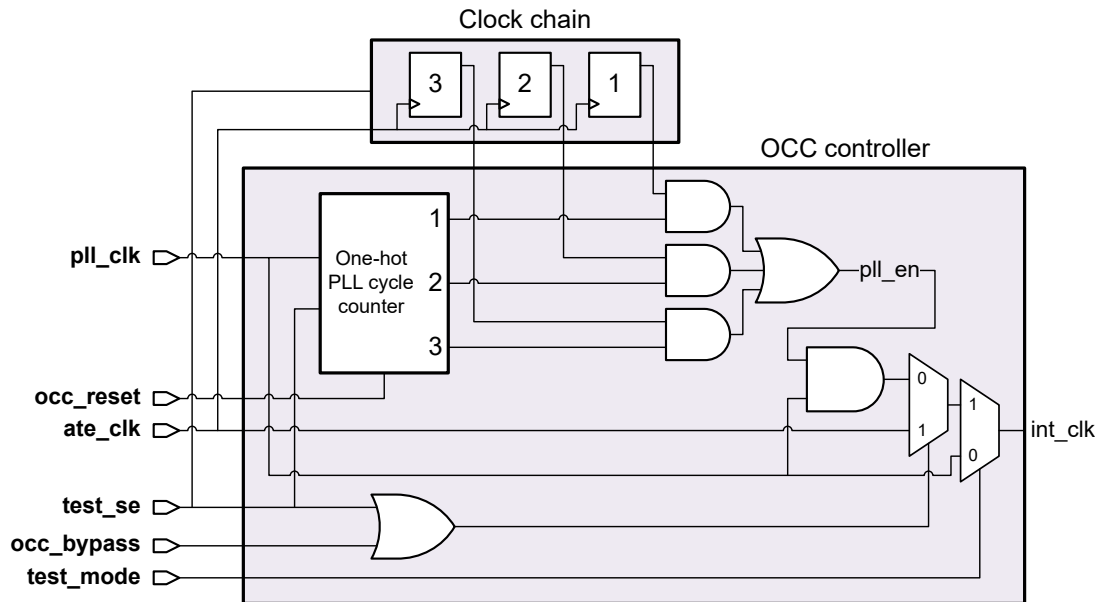
In the DFT-inserted OCC controller flow, the tool inserts a clock chain block that is separate from the OCC controller block.

In the user-defined OCC controller flow, the clock chain can be a part of the OCC controller design or it can be a separate design.

Logic Representation of an OCC Controller and Clock Chain

Figure 80 shows the logic structure of an OCC controller and clock chain.

Figure 80 Logic Representation of an OCC Controller



An OCC controller is designed to deliver up to a user-specified number N of at-speed clock pulse cycles during capture. A PLL cycle counter generates N successive one-hot enable signals, each of which is gated by the output of a clock chain scan cell. This logic structure provides ATPG with the flexibility to control which cycles deliver an at-speed clock pulse. For an OCC controller that handles multiple OCC generators, the clock chain contains a set of N scan cells for each clock.

Note that the figure shows the conceptual operation of a single-clock OCC controller. Implementation details, such as cleanly switching between the PLL clock and ATE clock and providing synchronous or asynchronous control of multiple clocks, are not shown.

See Also

- [SolvNetPlus article 034274, “DFT-inserted OCC Controller Data Sheet”](#) for more information about the logic structure and operation of the DFT-inserted OCC controller

Scan-Enable Signal Requirements for OCC Controller Operation

The scan-enable signal switches the OCC controller between the ATE shift clock and output PLL clock signals. Therefore, for proper operation, *the scan-enable signal must be held in the inactive state in all capture procedures.*

If you use the STIL protocol file created by the tool, the protocol already meets this requirement. The tool constrains all scan-enable signals to the inactive state in the capture procedures, excluding any scan-enable signals defined with the `-usage {clock_gating}` option of the `set_dft_signal` command. (Signals with multiple usages that include `clock_gating` are still constrained.)

If you use a custom STIL protocol file, make sure that all scan-enable signals used by OCC controllers are constrained to the inactive state in all capture procedures.

Enabling On-Chip Clocking Support

To enable OCC support for a design that contains or will contain OCC controllers, use the `-clock_controller` option of the `set_dft_configuration` command:

```
fc_shell> set_dft_configuration -clock_controller enable
```

Specifying OCC Controllers

This topic covers the different methods of specifying OCC controllers in Fusion Compiler:

- [Specifying DFT-Inserted OCC Controllers](#)
- [Specifying Existing User-Defined OCC Controllers](#)
- [Specifying OCC Controllers for External Clock Sources](#)

Specifying DFT-Inserted OCC Controllers

If you have a design that contains an OCC generator, such as a PLL, but not an OCC controller and clock chain, Fusion Compiler can insert both the OCC controller and clock chain. Note that this clock controller design supports only one ATE clock per OCC controller. This topic describes the flow associated with this type of implementation.

The PLL clock is expected to already be connected in the design being run through this flow. Fusion Compiler will disconnect this PLL clock at the hookup location and insert the newly synthesized clock controller at this location.

This topic covers the following:

- [Defining Clocks](#)
- [Defining Global Signals](#)
- [Configuring the OCC Controller](#)
- [Configuring the Clock Chain](#)

- [Configuring the Clock-Gating Logic](#)
- [Performing Timing Analysis](#)
- [Script Example](#)

Defining Clocks

You need to define the reference, PLL, and ATE clocks by using the `set_dft_signal` command. Note that this command does not require you to specify the primary inputs.

This topic covers the following:

- [Reference Clocks](#)
- [PLL-Generated Clocks](#)
- [ATE Clocks](#)

```
fc_shell> set_dft_signal -view existing_dft \  
-type RefClock -port refclk1 \  
-period 10 -timing [list 3 8]
```

Also note the following caveats when defining a reference clock:

- If the reference clock period is an integer divisor of the test default period, then patterns can be written in a variety of formats, including STIL, STIL99, and WGL.
- If the reference clock is not an integer divisor to the test default period, the only format that can be written in a completely correct way is STIL. Other formats, including STIL99, cannot include the reference clock pulses, and a warning is printed, indicating that these pulses must be added back to the patterns manually.
- Do not define a reference clock period or timings with resolution finer than 1 picosecond. The TestMAX ATPG tool cannot work with finer timing resolutions.

```
fc_shell> set_dft_signal -view existing_dft \  
-type PLLClock \  
-hookup_pin PLL/pllclk1  
fc_shell> set_dft_signal -view existing_dft \  
-type PLLClock \  
-hookup_pin PLL/pllclk2  
fc_shell> set_dft_signal -view existing_dft \  
-type PLLClock \  
-hookup_pin PLL/pllclk3  
  
fc_shell> set_dft_signal -view existing_dft \  
-type ATEClock \  
-port ATEclk \  
-timing [list 45 55]
```

By default, Fusion Compiler makes the ATE clock connection at the source port specified in the `-view existing_dft` signal definition. To specify a hookup pin to be used for the clock connection, use the `-hookup_pin` option in a subsequent `-view spec` scan clock signal definition. For example,

```
fc_shell> set_dft_signal -view spec \  
-type ATEClock \  
-port ATEclk \  
-hookup_pin PAD_ateclk/Z
```

You can use the same clock port as both the ATE clock and PLL reference clock. However, caveats apply. For more information, see [SolvNetPlus article 037838, "How Can I Use the Same Clock Port for the ATE and PLL Reference Clocks?"](#)

Reference Clocks

Reference clock signals are always defined in the existing DFT view. The `insert_dft` command does not connect them because they are considered to be functional signals rather than test signals. The only effect of defining them is that they are defined in the test protocol for use by DRC in the TetraMAX tool. For some special cases, a reference clock signal might not be needed.

See [Script Example](#) to see how to define a PLL reference clock.

PLL-Generated Clocks

For Fusion Compiler to correctly insert the OCC, you must define the PLL-generated clocks as well as the point at which they are generated. See [Script Example](#) to see how to define a set of launch and capture clocks for internal scannable elements controlled by the OCC controller.

ATE Clocks

See [Script Example](#) to see how to define the signal behavior of the ATE-provided clock required for shifting scan elements.

Defining Global Signals

You must identify the top-level interface signals that control the OCC controller. This includes the OCC bypass, OCC reset, and ScanEnable signals. You must also define a dedicated TestMode signal that activates the OCC controller logic. In the OCC controller insertion flow, these signals are defined with the `-view spec` option because they will be implemented and connected by the `insert_dft` command.

The following examples show how to define a set of OCC controller interface signals for the design example:

```
fc_shell> set_dft_signal -view spec \  
-type occ_reset \  
-port OCC_reset
```

```
fc_shell> set_dft_signal -view spec \
    -type occ_bypass \
    -port OCC_bypass
fc_shell> set_dft_signal -view spec \
    -type ScanEnable \
    -port SE
fc_shell> set_dft_signal -view spec \
    -type TestMode -usage occ \
    -port TM_OCC
```

The TestMode signal must be a dedicated signal for the OCC controller. It must be active in all test modes and inactive in mission mode. It cannot be shared with TestMode signals used for other purposes, such as test-mode selection.

In the internal pins flow, you can specify internal hookup pins for these OCC control signals by using the `-hookup_pin` option of the `set_dft_signal` command. However, you cannot specify internal hookup pins for ATE clocks or reference clocks.

Configuring the OCC Controller

To specify where to insert a DFT-inserted OCC controller, use the `set_dft_clock_controller` command. Note the following syntax and descriptions:

```
set_dft_clock_controller
    -cell_name cell_name
    -ateclock clock_name
    -test_mode_port port_name
    -pllclks ordered_list
    [-chain_count integer]
    [-cycles_per_clock integer]
```

Table 7 *set_dft_clock_controller Command Syntax*

Option	Description
<code>-cell_name <i>cell_name</i></code>	Specifies the hierarchical name of the clock controller cell.
<code>-ateclock <i>clock_name</i></code>	Specifies the ATE clock (port) you want to connect to the OCC controller. Note: You cannot specify multiple clocks per controller.
<code>-test_mode_port <i>port_name</i></code>	Specifies the test-mode port used to enable the clock controller. Use this option if you have multiple test-mode ports and you want to use a specific port to enable the clock controller. The specified port must be defined as a TestMode signal using the <code>set_dft_signal</code> command.
<code>-pllclks <i>ordered_list</i></code>	For asynchronous OCC controllers, specifies the ordered list of PLL output clock pins to control.

Table 7 *set_dft_clock_controller Command Syntax (Continued)*

Option	Description
<code>-chain_count integer</code>	Specifies the number of clock chains. The default number of clock chains is one.
<code>-cycles_per_clock integer</code>	Specifies the maximum number of capture cycles per clock. You should specify a value of two or greater. Capture cycles are cycles during capture when capture clocks are pulsed. Typically, for at-speed transition testing, there are two capture cycles: one is used for launching a transition and the other for capturing the effect of that transition.

Use the `-pllclocks` option to specify the list of clock source pins to be controlled.

The following example inserts an OCC controller that controls three clocks:

```
fc_shell> set_dft_clock_controller \
-cell_name occ_int \
-ateclock { ATEclk } \
-test_mode_port { occ_test_mode } \
-pllclocks { pll/pllclk1 pll/pllclk2 pll/pllclk3 } \
-cycles_per_clock 2
```

To insert multiple OCC controllers, use multiple `set_dft_clock_controller` commands.

Configuring the Clock Chain

By default, the clock chain is constructed as follows:

- In scan compression modes, the clock chain registers exist on their own compressed chain.
- In standard scan modes, the clock chains can be combined with other compressed chains to form scan chains.

You can use the `set_scan_path` command with the `-class occ` option to define a particular structure. Specify the cell name of an OCC controller to reference its clock chain.

For example, to create an external clock chain for two OCC controllers,

```
set_dft_clock_controller -cell_name BLK1/OCC1 ...
set_dft_clock_controller -cell_name BLK2/OCC2 ...
set_scan_path OCC_CHAIN -class occ \
-include_elements {BLK1/OCC1 BLK2/OCC2} \
-scan_data_in SI_OCC -scan_data_out SO_OCC
```

```
set_dft_clock_controller -cell_name BIG_OCC -chain_count 2 \
  -pll_clocks {(many clocks)} ...
```

In `set_scan_path` specifications, the OCC controller cell name references all clock chains for that OCC controller. To reference individual clock chains, append an index number using the colon (:) character:

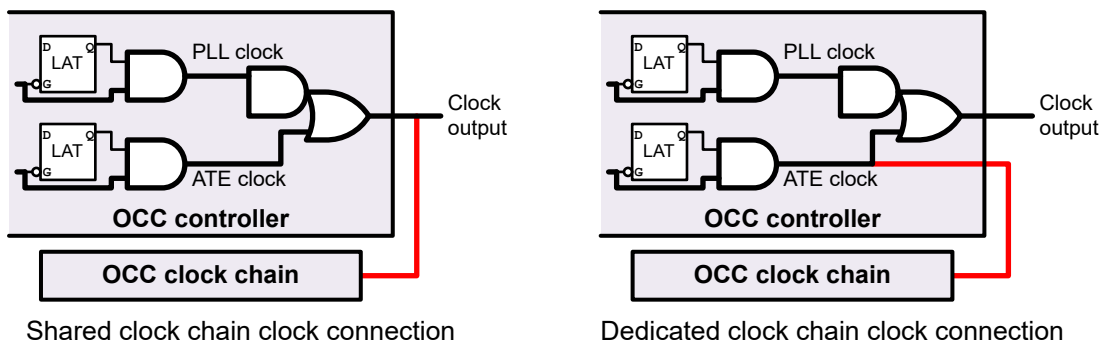
```
set_dft_clock_controller -cell_name BIG_OCC -chain_count 2 \
  -pll_clocks {(many clocks)} ...
set_scan_path OCC_CHAIN1 -class occ \
  -include_elements {BIG_OCC:0} \
  -scan_data_in SI_OCC1 -scan_data_out SO_OCC1
set_scan_path OCC_CHAIN2 -class occ \
  -include_elements {BIG_OCC:1} \
  -scan_data_in SI_OCC2 -scan_data_out SO_OCC2

fc_shell> set_app_options -name dft.test_dedicated_clock_chain_clock
  -value true
```

This creates a dedicated OCC controller clock output for the clock chain that places it in only the ATE clock path, which prevents it from affecting the high-speed PLL clock path.

Figure 81 shows both types of clock-chain clock connections.

Figure 81 Shared and Dedicated Clock Chain Clock Connections



In both cases, you should consider how the clock-chain clock connection interacts with clock tree synthesis (CTS).

Handling Long Clock Chains

The default is to create a single clock chain. However, if you have many clocks to control, the clock chain might become longer than other scan chains.

To split the clock control bits up across multiple clock chains, use the `-chain_count` option.

Configuring the Clock-Chain Clock Connection

By default, the clock-chain clock connection shares the first functional clock output of the OCC controller. This places the clock chain in both the PLL and ATE clock paths.

To use a dedicated clock-chain clock connection from the OCC controller design, set the following application option:

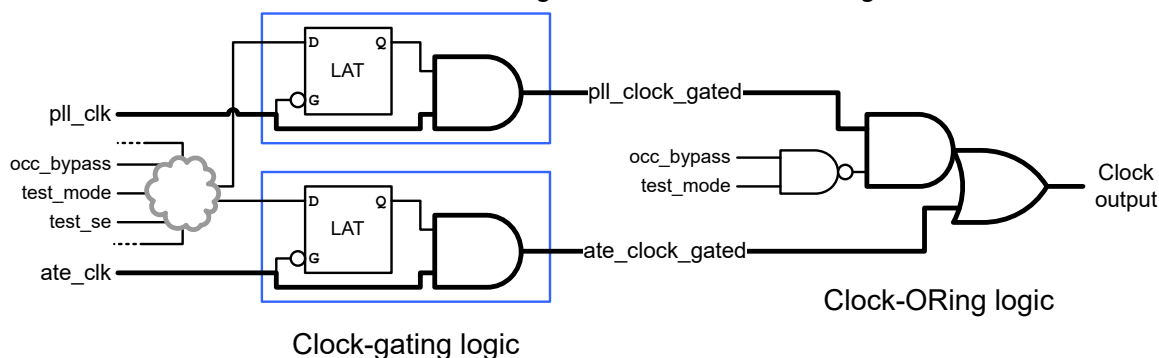
```
fc_shell> set_app_options -name dft.test_dedicated_clock_chain_clock
-value false
```

Configuring the Clock-Gating Logic

The DFT-inserted OCC controller uses latch-based clock-gating logic for glitch-free selection between the fast and slow clocks. For the fast and slow clocks, the combinational gating-enable signals are combined and latched for each clock; the latched enable signal is then used to gate the clock.

By default, the tool builds the clock-gating logic using discrete latch and combinational library cells, as shown in [Figure 82](#). Clock paths are shown in bold, and rectangles indicate hierarchy created inside the OCC controller block by DFT insertion.

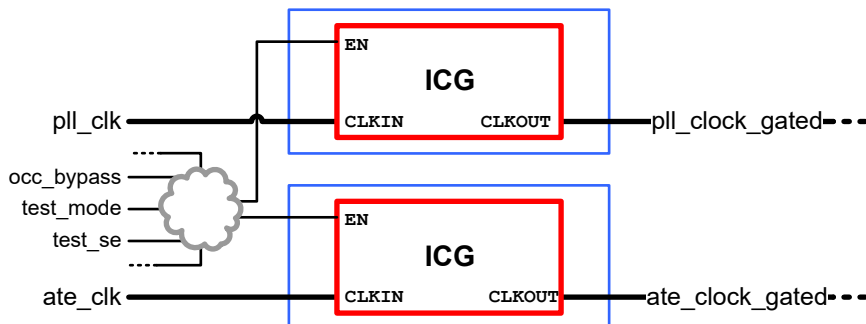
Figure 82 Default Latch-Based Clock-Gating and Clock Selection Logic Structure



To use an integrated clock-gating cell instead, as shown in [Figure 83](#), set the targeted library cell reference (without the library name) as follows:

Fusion Compiler

Figure 83 Specifying an Integrated Clock-Gating Cell for the Clock-Gating Logic



For more information about this application option, see the man page.

See Also

- [SolvNetPlus article 034274, “DFT-inserted OCC Controller Data Sheet”](#) for more information about the logic structure and operation of the DFT-inserted OCC controller

Performing Timing Analysis

After DFT insertion completes, you must ensure that the OCC controller logic is properly constrained for timing analysis.

See Also

- [SolvNetPlus article 022490, “Static Timing Analysis Constraints for On-Chip Clocking Support”](#) for more information about performing timing analysis in a DFT-inserted OCC controller flow
- [SolvNetPlus article 034274, “DFT-Inserted OCC Controller Data Sheet”](#) for information about synthesizing a DFT-inserted OCC controller (two sections titled “Special Considerations”)

Script Example

[Example 9](#) shows a script that performs pre-DFT DRC, scan chain stitching, and post-DFT DRC. The STIL protocol file generated at the end of the DFT insertion process contains PLL clock details suitable for the TestMAX ATPG tool.

Example 9 Flow Example for DFT-Inserted OCC Controller and Clock Chain

```
read_verilog mydesign.v
current_design mydesign
link

# Define the PLL reference clock
# top level free running clock
set_dft_signal -view existing_dft -type RefClock \
```

```
-port refclk1 -period 100 -timing {45 55}

# Define the ATE clock
# the ATE-provided clock for shift of scan elements
set_dft_signal -view existing_dft -type ATEClock \
  -port ATEclk -timing {45 55}

# Define the PLL generated clocks --
# these are the launch/capture clocks for internal scannable
# elements and are controlled by occ controller
set_dft_signal -view existing_dft -type PLLClock \
  -hookup_pin pll/pllclk1
set_dft_signal -view existing_dft -type PLLClock \
  -hookup_pin pll/pllclk2
set_dft_signal -view existing_dft -type PLLClock \
  -hookup_pin pll/pllclk3

# Define the OCC TestMode signal
set_dft_signal -type TestMode -usage occ -port TM_OCC -view existing_dft

# Enable on-chip clocking control
set_dft_configuration -clock_controller enable

# The following command specifies the OCC controller
# design to be instantiated
set_dft_clock_controller \
  -cell_name snps_occ_controller \
  -ateclock { ATEclk } \
  -test_mode_port { TM_OCC } \
  -pllclocks { pll/pllclk1 pll/pllclk2 pll/pllclk3 } \
  -cycles_per_clock 2
set_scan_configuration -chain_count 30 -clock_mixing no_mix
report_dft -clock_controller
create_test_protocol
preview_dft
insert_dft
compile_fusion

# Run DRC with external clocks enabled during capture
# (PLL bypassed)
set_dft_drc_configuration -pll_bypass enable
dft_drc
change_names -rules verilog -hierarchy
write -format verilog -hierarchy -output top.scan.v

# Write out combined PLL enabled/bypassed test protocol:
write_test_protocol -output scan_pll.stil
```

Specifying Existing User-Defined OCC Controllers

If you have a design that contains an OCC generator and it already instantiates an existing user-defined OCC controller and clock chain, Fusion Compiler can analyze the OCC controller, validate the functionality, and incorporate it into the test protocol.

This topic covers the following:

- [Defining Clocks](#)
- [Defining Global Signals](#)
- [Specifying Clock Chains](#)
- [Script Example](#)

Defining Clocks

Use the `set_dft_signal` command to define the following clock signals:

- [Reference Clocks](#)
- [PLL-Generated Clocks](#)
- [ATE Clocks](#)
- [Clock Chain Configuration and Control-Per-Pattern Information](#)

```
fc_shell> set_dft_signal -view existing_dft \  
                  -type RefClock -port refclk1 \  
                  -period 10 -timing [list 3 8]
```

Also note the following caveats associated when defining a reference clock:

- If the reference clock period is an integer divisor of the test default period, then patterns can be written in a variety of formats, including STIL, STIL99, and WGL.
- If the reference clock is not an integer divisor of the test default period, the only format that can be written in a completely correct way is STIL. Other formats (including STIL99) cannot include the reference clock pulses. The tool issues a warning to indicate that these pulses must be added back to the patterns manually.
- Do not define a reference clock period or timings with resolution finer than 1 picosecond. The TestMAX ATPG tool cannot work with finer timing resolutions.

```
fc_shell> set_dft_signal -view existing_dft \  
                  -type PLLClock \  
                  -hookup_pin PLL/pllclk1  
fc_shell> set_dft_signal -view existing_dft \  
                  -type PLLClock \  
                  -hookup_pin PLL/pllclk2
```

```
fc_shell> set_dft_signal -view existing_dft \  
-type PLLClock \  
-hookup_pin PLL/pllclk3  
  
fc_shell> set_dft_signal -view existing_dft \  
-type ATEClock \  
-port ATEclk \  
-timing [list 45 55]
```

You can use the same clock port as both the ATE clock and PLL reference clock. However, caveats apply. For more information, see [SolvNetPlus article 037838](#), “How Can I Use the Same Clock Port for the ATE and PLL Reference Clocks?”

```
fc_shell> set_dft_signal -view existing_dft \  
-type MasterClock -usage occ \  
-hookup_pin occ_int/intclk1 \  
-ate_clock ATEclk \  
-pll_clock PLL/pllclk1 \  
-ctrl_bits [list 0 occ_int/FF_cyc1/Q 1 \  
                1 occ_int/FF_cyc2/Q 1 \  
                2 occ_int/FF_cyc3/Q 1]
```

The `-ctrl_bits` option is used to provide a list of triplets that specify the sequence of bits needed to enable the propagation of the clock generator outputs. The first element of each triplet is the cycle number (integer) indicating the cycle where the clock signal will be propagated. The second element is the pin name (a valid design hierarchical pin name) of the clock chain control bit. The third element is the active state (0 or 1) of the control bit signal. For more information about this option, see the `set_dft_signal` man page.

The `-view existing_dft` option is used because connections already exist between the referenced port and the clock controller.

Reference Clocks

A reference clock definition is used primarily as an informational device. The only effect of defining them is that they are defined in the test protocol for use by DRC. For some special cases, a reference clock signal might not be needed.

The following example shows how to define a PLL reference clock:

PLL-Generated Clocks

PLL clocks are the output of the PLL. This output is a free-running source that also runs at a constant frequency, which might not be the same as the reference clock's. This information is forwarded to the TestMAX ATPG tool through the protocol file to allow the verification of the clock controller logic.

The following commands show how to define the PLL clocks for the design example:

ATE Clocks

An ATE clock signal can be pulsed several times before and after scan shift (scan-enable signal inactive) to synchronize the clock controller logic in the capture phase and back into the shift phase.

The following commands show how to define ATE clocks for the design example:

Clock Chain Configuration and Control-Per-Pattern Information

You must specify the correlation between the internal clock signals driven from the clock controller outputs, the signals driven from the clock generator (PLL) outputs, and the signals provided by the user-defined clock chain. This information indicates how the clock signal is enabled by the clock chain control bits in each clock cycle. The correspondence between the controlled internal clock signals and clock chain control bits is identified in the protocol file for TestMAX ATPG to generate patterns.

You must specify

- All user-defined clock controller outputs referencing the internal clocks
- A corresponding set of clock chain control bits, ATE clock, and clock generator output (PLL) for each clock controller output

The following example specifies a user-defined OCC controller with a three-bit clock chain:

Defining Global Signals

You must identify the top-level interface signals that control the OCC controller. This includes the OCC bypass, OCC reset, and ScanEnable signals. You must also define a dedicated TestMode signal that activates the OCC controller logic. In the user-defined OCC controller flow, these signals are defined with the `-view existing_dft` option because they already exist and must be described to the tool.

The following examples show how to define a set of OCC controller interface signals for the design example:

```
fc_shell> set_dft_signal -view existing_dft \  
-type occ_reset \  
-port OCC_reset  
fc_shell> set_dft_signal -view existing_dft \  
-type occ_bypass \  
-port OCC_bypass  
fc_shell> set_dft_signal -view existing_dft \  
-type ScanEnable \  
-port SE  
fc_shell> set_dft_signal -view existing_dft \  
-type TestMode -usage occ \  
-port TM_OCC
```

The TestMode signal must be a dedicated signal for the OCC controller. It must be active in all test modes and inactive in mission mode. It cannot be shared with TestMode signals used for other purposes, such as AutoFix or multiple test-mode selection.

You can specify an internal hookup pin for any of these OCC controller interface signals by using the `-hookup_pin` option of the `set_dft_signal` command. You cannot specify internal hookup pins for ATE clocks or reference clocks.

Specifying Clock Chains

The clock chain is a shift register with an enable, typically clocked on the rising edge. It is always defined with the `set_scan_path` command, and it must be defined with the `-class occ` option to mark it as the clock chain.

The commands used depend on whether your clock chain is in RTL (inferred registers) or gates (scan-stitched register cells).

```
module my_clock_chain (clk, si, se, qmask);
  input clk, si, se;
  output [0:2] qmask;
  reg [0:2] qmask;
  always @(posedge clk)
    if (se == 1'b1)
      qmask[0:2] <= {si, qmask[0:1]};
endmodule
```

In your Tcl script, use the `set_scan_path` command to define the elaborated shift register cells, from earliest clock bit to latest:

```
fc_shell> set_scan_path my_path1 -class occ \
  -include { \
    my_occ/my_clock_chain/qmask_reg[0] \
    my_occ/my_clock_chain/qmask_reg[1] \
    my_occ/my_clock_chain/qmask_reg[2] }

module my_clock_chain (clk, si, se, qmask, so);
  input clk, si, se;
  output [2:0] qmask;
  reg [2:0] qmask;
  output so;
  wire gclk;
  CCICG ICGX1 (.EN(shift), .TE(1'b0), .CLKI(clk), .CLKO(gclk));
  FF0 DFFX1 (.D(si), .CK(gclk), .q(qmask[0]));
  FF1 DFFX1 (.D(qmask[0]), .CK(gclk), .q(qmask[1]));
  FF2 DFFX1 (.D(qmask[1]), .CK(gclk), .q(qmask[2]));
  assign so = qmask[2];
endmodule
```

In your OCC controller netlist, use the `set_scan_group` command to define the cell order and access pins of the shift register:

```
fc_shell> set_scan_group my_cc_group \  
-include_elements { \  
    my_occ/my_clock_chain/qmask_reg[0] \  
    my_occ/my_clock_chain/qmask_reg[1] \  
    my_occ/my_clock_chain/qmask_reg[2]} \  
-access { \  
    ScanDataIn my_occ/my_clock_chain/si \  
    ScanDataOut my_occ/my_clock_chain/so \  
    ScanEnable my_occ/my_clock_chain/se} \  
-serial_routed true
```

Then, use the `set_scan_path` command with the `-class occ` option to mark this shift group as a clock chain:

```
fc_shell> set_scan_path my_clock_chain -class occ \  
-include_elements {my_cc_group}
```

RTL Clock Chains

In your OCC controller RTL, describe the clock chain as a simple shift register with a shift enable. For example,

Gate-Level Clock Chains

In your OCC controller netlist, describe the clock chain as a simple shift register with a shift enable. For example,

Script Example

When you run a design that contains a user-defined OCC controller and clock chains, a STIL protocol file is generated, as shown in [Example 10](#).

Example 10 Flow Example for Existing OCC Controller and Clock Chain

```
read_verilog mydesign.v  
current_design mydesign  
link  
  
# Define the PLL reference clock  
# (top-level free running clock)  
set_dft_signal -view existing_dft -type RefClock \  
    -port refclk1 -period 100 -timing [list 45 55]  
  
# Define the ATE clock  
# (the ATE-provided clock for shift of scan elements)  
set_dft_signal -view existing_dft -type ATEClock \  
    -port ATEclk -timing [list 45 55]  
  
# Define the PLL generated clocks
```

```
set_dft_signal -view existing_dft -type PLLClock \
    -hookup_pin pll/pllclk1
set_dft_signal -view existing_dft -type PLLClock \
    -hookup_pin pll/pllclk2
set_dft_signal -view existing_dft -type PLLClock \
    -hookup_pin pll/pllclk3

# Enable PLL capability
set_dft_configuration -clock_controller enable

# Specify clock controller output and control-per-pattern information
set_dft_signal -type MasterClock -usage occ \
    -hookup_pin occ_int/intclk1 \
    -ate_clock ATEclk -pll_clock PLL/pllclk1 \
    -ctrl_bits [list 0 occ_int/FF_1/Q 1 \
        1 occ_int/FF_2/Q 1] \
    -view existing_dft

# Define the existing clock chain segments
set_scan_group my_cc_group \
    -include_elements [list occ_int/FF_1 \
        occ_int/FF_2] \
    -access [ list ScanDataIn occ_int/si \
        ScanDataOut occ_int/so \
        ScanEnable occ_int/se] \
    -serial_routed true
set_scan_path my_clock_chain -class occ \
    -include_elements {my_cc_group}

# Specify global controller signals
set_dft_signal -type occ_reset -port OCC_reset -view existing_dft
set_dft_signal -type occ_bypass -port OCC_bypass -view existing_dft
set_dft_signal -type ScanEnable -port SE -view existing_dft

# Define the OCC TestMode signal
set_dft_signal -type TestMode -usage occ -port TM_OCC -view existing_dft

# Registers inside the OCC controller must be nonscan or else the
# internal clock will not be controlled correctly. However, the
# clock chain must be scanned. Use set_scan_element false on the
# former but not on the latter.
set_scan_element false occ_int/clock_controller
set_scan_configuration -chain_count 30 -clock_mixing no_mix
create_test_protocol
dft_drc
report_dft -clock_controller
preview_dft -show all
insert_dft

# Run DRC with external clocks enabled during capture
# (PLL bypassed)
set_dft_drc_configuration -pll_bypass enable
dft_drc
```



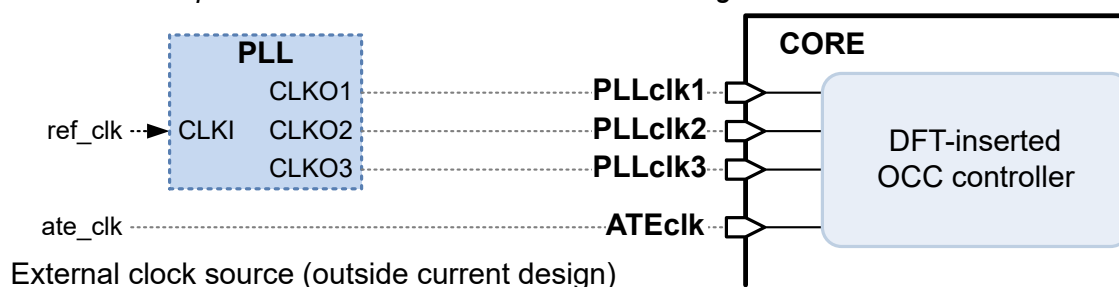
```
change_names -rules verilog -hierarchy
write -format verilog -hierarchy -output top.scan.v

# Write out combined PLL enabled/bypassed test protocol:
write_test_protocol -output scan_pll.stil
```

Specifying OCC Controllers for External Clock Sources

In some cases, the on-chip clocking source might be external to the current design, as shown in [Figure 84](#), so that the PLL clocks enter the design through input ports.

Figure 84 On-Chip Clock Source External to Current Design



Define such external clock sources as follows:

- Define the port-driven PLL-generated clocks at the ports, and define them as `ScanClock` instead of `Oscillator`:

```
fc_shell> set_dft_signal -view existing_dft \
  -type PLLClock \
  -port {PLLclk1 PLLclk2 PLLclk3} \
  -timing {45 55}
```

Specify typical scan clock timing, even though the clock is a PLL-generated clock instead of a tester-driven clock. This timing discrepancy does not affect OCC architecture or operation.

- If the reference clock does not enter the design, you do not need to define it.
- Include the PLL-generated input ports in the PLL clock sources list specified with the `-pllclocks` option:

```
fc_shell> set_dft_clock_controller \
  -cell_name occ_int \
  -test_mode_port TM_OCC \
  -pllclocks {PLLclk1 PLLclk2 PLLclk3} \
  -ateclock {ATEclk} \
  -cycles_per_clock 2
```

The resulting SPF does not pulse the clock ports during capture, which is incorrect. As a workaround, define the clock as a reference clock in the TestMAX ATPG tool. For example,

```
DRC-T> add_clock 0 {PLLclk1 PLLclk2 PLLclk3} -refclock
```

External clock sources are not supported for existing user-defined OCC controllers.

Reporting Clock Controller Information

Use the `report_dft -clock_controller` command to generate reports.

DFT-Inserted OCC Controller Flow

For DFT-inserted OCC controllers and clock chains, use the `report_dft -clock_controller` command to output a report. This report displays the options that you set for the `set_dft_clock_controller` command.

[Example 11](#) shows a clock controller report for the DFT-inserted OCC controller flow.

Example 11 Report Example from the report_dft_clock_controller -view spec Command

```
*****
Report : OCC Controllers Specification
Design : des_chip
Version: 2.75.0
Date   : Thu Jun 15 10:57:09 2017
*****
=====
TEST MODE: all_dft
VIEW      : Specification
=====
Cell name:                pll_controller_0
Chain count:              1
Cycle count:              2
Fast clock pins:          u_pll/clko1 u_pll/clko2
Slow clock ports/pins:    atecclk
```

Existing User-Defined OCC Controller Flow

For existing user-defined OCC controllers and clock chains, after you define the internal clock signals and the corresponding control-per-pattern information, use the `report_dft_clock_controller -view existing_dft` command to report what you have specified. In [Example 12](#), the report shows information about the clock generator signal, the ATE clock, the OCC controller output, and the clock chain control bits.

Example 12 Report Example from the report_dft_clock_controller Command

```
*****
Report : OCC Controllers Specification
Design : des_chip
Version: 2.75.0
Date   : Thu Jun 15 10:59:48 2017
*****
Clock controller: ctrl_0
=====
Number of bits per clock: 4
Controlled clock output pin: duto/clk
=====
Clock generator signal: duto/PLLCLK
ATE clock signal: i_ateclk
Control pins:
      cycle 0  duto/snps_clk_chain_0/FF_0/Q  1
      cycle 1  duto/snps_clk_chain_0/FF_1/Q  1
      cycle 2  duto/snps_clk_chain_0/FF_2/Q  1
      cycle 3  duto/snps_clk_chain_0/FF_3/Q  1
=====
=====
```

Controlling the OCC Controller Bypass Path

Use the `set_dft_drc_configuration` and `write_test_protocol` commands to enable the OCC controller bypass configuration for design rule checking. The `-pll_bypass` option of the `set_dft_drc_configuration` command configures post-DFT DRC to place the OCC clock controller in bypass configuration.

The syntax is as follows:

```
set_dft_drc_configuration -pll_bypass enable | disable
```

The default setting is `disable`.

The following example performs DRC of both bypass configurations for a scan compression mode:

```
insert_dft
compile_fusion
set_dft_drc_configuration -pll_bypass disable ;# already the default
dft_drc -test_mode ScanCompression_mode
set_dft_drc_configuration -pll_bypass enable
dft_drc -test_mode ScanCompression_mode
write_test_protocol -test_mode Scan_Compression_mode -output
my_design.spf
```

The test protocol written by the `write_test_protocol` command contains information for PLL bypass as well as for PLL enabled. In the TestMAX ATPG tool, use the `run_drc -patternexec` command to select the operating mode to use.

DFT-Inserted OCC Controller Configurations

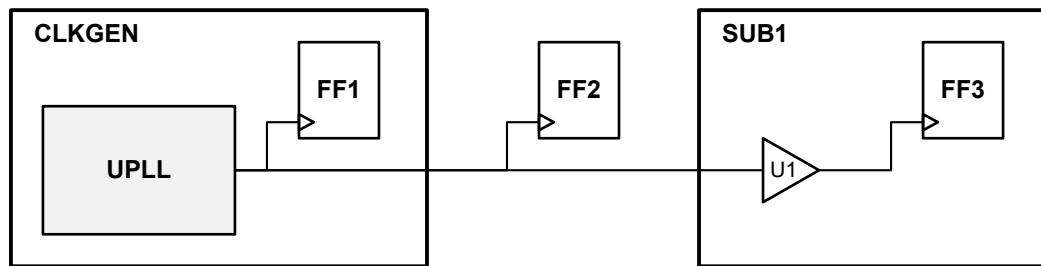
This topic shows DFT-inserted OCC controller configurations and the associated configuration commands, as described in the following topics:

- [Single OCC Controller Configurations](#)
- [Multiple DFT-Inserted OCC Controller Configurations](#)

Single OCC Controller Configurations

This topic shows the results of using various configurations of the `set_dft_clock_controller` command on the design example shown in [Figure 85](#).

Figure 85 Design Example for Single OCC Controller Insertion



The following configuration examples are applied to this design:

- [Example 1](#) – Controller inserted at the output of UPLL, within the CLKGEN block.
- [Example 2](#) – Controller inserted at the output of the CLKGEN block.
- [Example 3](#) – Controller inserted at the output of the buffer.

Example 1

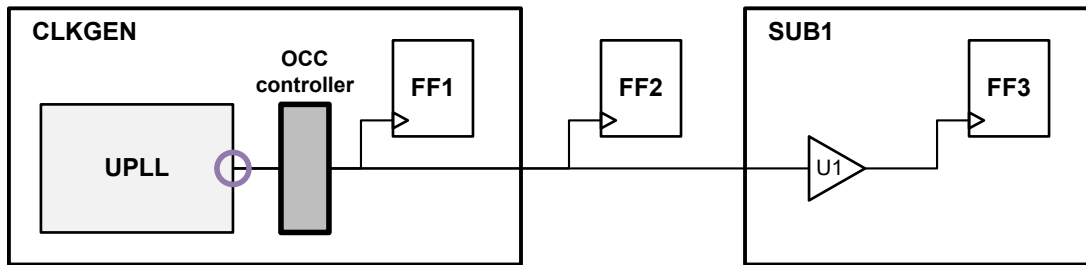
The first example, shown in [Figure 86](#), uses the following configuration:

```
fc_shell> set_dft_clock_controller \
  -pllclocks {CLKGEN/UPLL/clkout}
```

In this case, the following occurs:

- The controller is inserted at the output of PLL, within the clkgen1 block.
- The clocks of all flip-flops are controllable.

Figure 86 Controller Inserted at Output of PLL Within CLKGEN Block



Example 2

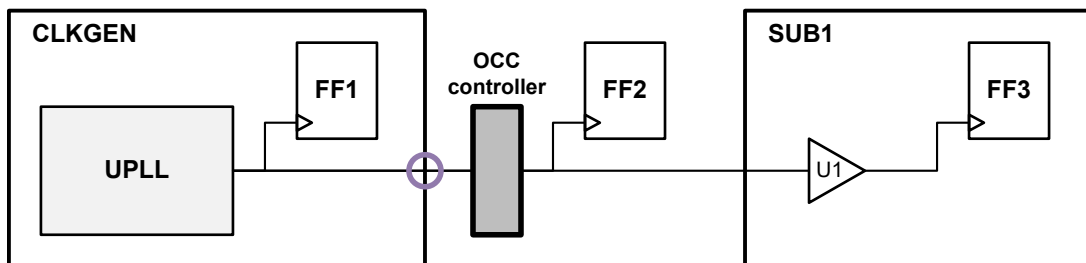
The second example, shown in Figure 87, uses the following configuration:

```
fc_shell> set_dft_clock_controller \
  -pllclocks {CLKGEN/clkout}
```

In this case, the following occurs:

- The controller is inserted at the output of the CLKGEN block.
- The FF1 clock remains uncontrollable.

Figure 87 Controller Inserted at Output of CLKGEN Block



Example 3

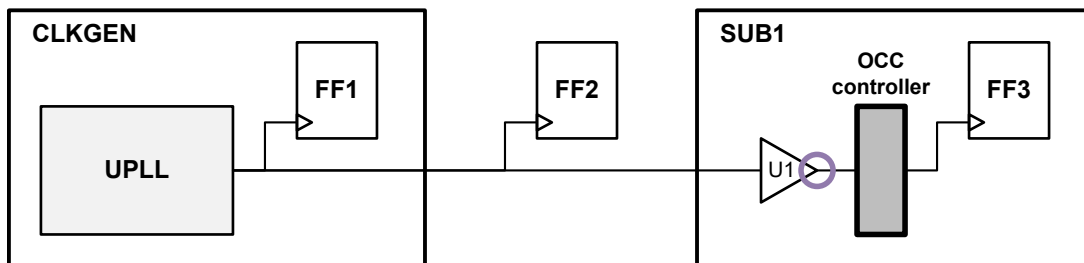
The third example, shown in Figure 88, uses the following configuration:

```
fc_shell> set_dft_clock_controller \
  -pllclocks {SUB1/U1/Z}
```

In this case, the following occurs:

- The controller is inserted at the output of buffer U1.
- The FF1 and FF2 clocks remain uncontrollable.

Figure 88 Controller Inserted at Output of the Buffer



Multiple DFT-Inserted OCC Controller Configurations

This topic shows the results of configuring multiple DFT-inserted OCC controllers for the design example shown in [Figure 85](#).

When multiple PLLs exist in a design, the reference clock input to each PLL cell must be a free-running clock in test mode. Care must be taken to insure that an OCC controller is not inserted at a location that would block a free-running clock to a downstream PLL cell.

In this design example, the primary PLL named UPLL1 receives the incoming reference clock and generates a PLL output clock. This PLL output clock then feeds either a second PLL or clock divider cell, creating a second cascaded PLL output clock.

The following configuration examples are applied to this design:

- [Example 1](#) – Controller incorrectly inserted, at the output of UPLL1.
- [Example 2](#) – Controller correctly inserted, at the output of a buffer driven by UPLL1.

Example 1

The first example, shown in [Figure 89](#), uses the following configuration:

```

fc_shell> set_dft_clock_controller \
    -pllclocks {UPLL1/clkout}
fc_shell> set_dft_clock_controller \
    -pllclocks {UPLL2/clkout}

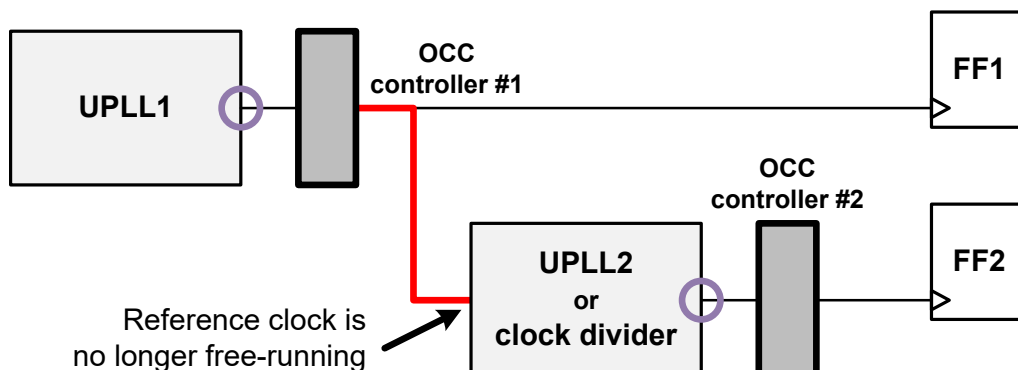
```

In this case, the following occurs:

- The controller is inserted at the output of UPLL1.
- As a result, the free-running clock to UPLL2 is blocked by the first OCC controller, causing incorrect operation of UPLL2.

The incorrect operation of UPLL2 might only be detectable during Verilog simulation of the resulting netlist.

Figure 89 Free-Running Clock Blocked to UPLL2



Example 2

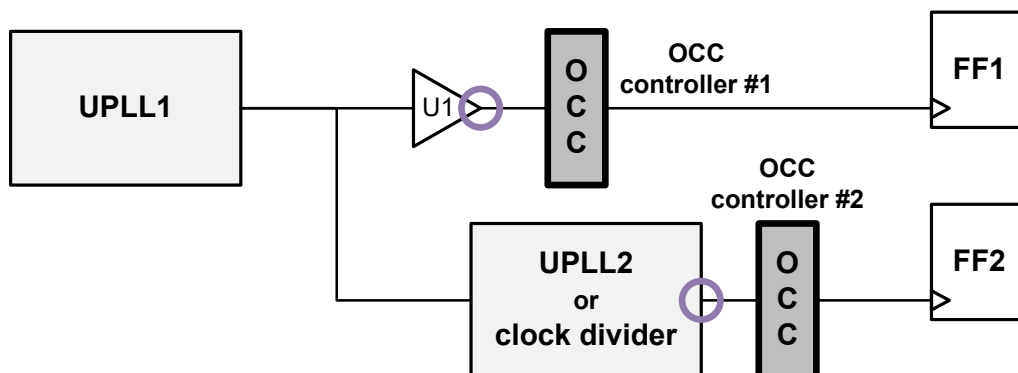
The second example, shown in Figure 90, uses the following configuration:

```
fc_shell> set_dft_clock_controller \
  -pllclks {U1/Z}
fc_shell> set_dft_clock_controller \
  -pllclks {UPLL2/clkout}
```

This example uses a buffer to isolate the downstream fanout that the first OCC controller should drive. In this case, the following occurs:

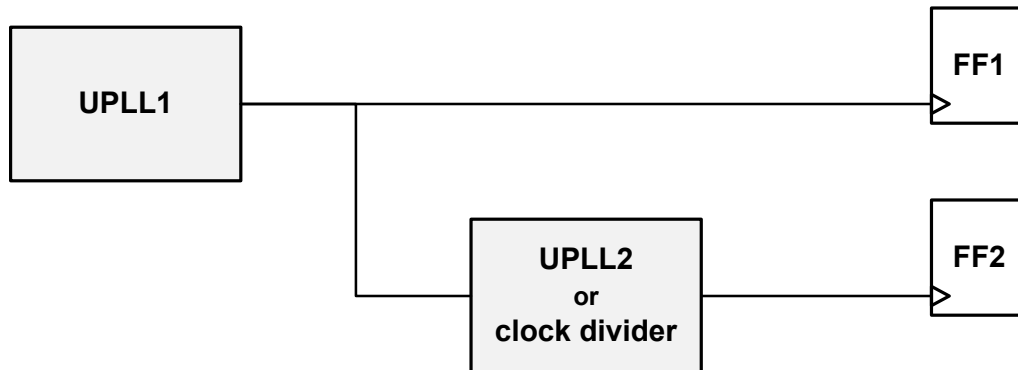
- The controller is inserted at the output of buffer U1 driven by UPLL1.
- As a result, the free-running clock from UPLL1 propagates to UPLL2, allowing correct operation of UPLL2.

Figure 90 Free-Running Clock Propagates to UPLL2



You must ensure that the isolation buffer is not optimized away by applying a `set_dont_touch` command, applying a `set_size_only` command, or using hierarchy. After the clock controller is inserted, the resulting test protocol references the specified pin as a PLL pin.

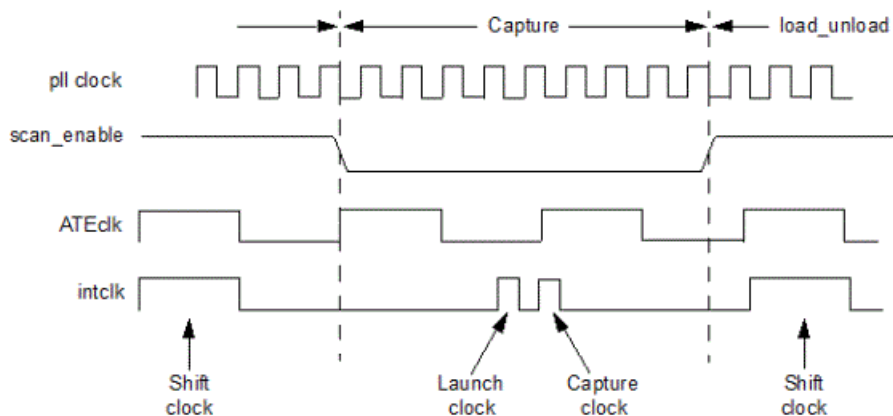
Figure 91 Design Example for Multiple OCC Controller Insertion



Waveform and Capture Cycle Example

Figure 92 shows an example of the relationship between various clocks when the design contains an OCC generator and an OCC controller.

Figure 92 Desired Clock Launch Waveform Example



For information about `pll clock`, `ATEclk`, and `intclk`, see [Clock Type Definitions](#).

Limitations

Note the following limitations:

- In Fusion Compiler,
 - Clock chains must be defined with the `set_scan_path -class occ` command; they are not automatically created.
- Clock chains are not reported by the `report_scan_path` or `report_dft -scan_path` commands (although the `preview_dft` command reports them).
- Inferencing internal PLL or any reference clocks is not supported. For pre-DFT DRC, you must explicitly define your reference clock, ATE clock, and PLL clocks.
- You cannot mix the DFT-inserted and user-defined OCC controller types in the same DFT insertion run. However, this restriction does not apply to cores that already contain OCC controllers.
- The only supported scan style is multiplexed flip-flop.
- External (port-driven) clocks are not yet supported for OCC control.
- Fast-sequential patterns with OCC support cannot measure the primary outputs between system pulses. The measure primary output is placed before the first system pulse and measures only Xs. You have to use pre-clock-measure, with the strobe being placed before the clock.
- External clocks, which have a direct connection to scan flip-flops, cannot serve as ATE clocks for the OCC controller.
- End-of-cycle measures cannot be used when an OCC controller is used to control the clock.
- When a pipelined scan-enable signal is used with OCC flows, the `insert_dft` command fails to make some connections properly. To use these features together, you must check and correct the connections so that the following requirements are met:
 - The scan-enable connections to the OCC controller and clock chain must use the unpipelined scan-enable signal. That is, use the input to the scan-enable pipeline register instead of its output.
 - The clock connection to the scan-enable pipeline register in OCC controller clock domains must be connected to the internal clock output of the OCC controller block.
- External (port-driven) clock sources are not supported for existing user-defined OCC controllers.

14

Inserting Test Points

Test points are points in the design where Fusion Compiler inserts logic to improve the testability of the design. The tool can automatically determine where to insert test points to improve test coverage and reduce pattern count. You can also manually define where test points are to be inserted.

The test point capabilities are described in the following topics:

- [Test Point Types](#)
- [Test Point Structures](#)
- [Automatically Inserted Test Points](#)
- [Inserting the Test Point Logic](#)
- [Inserting Power-Aware Test Points](#)

Test Point Types

The available test point types are:

- [Force Test Points](#)
- [Control Test Points](#)
- [Observe Test Points](#)
- [Multicycle Test Points](#)

The test point schematics in these topics show the functional operation of the test points. During synthesis, constant logic is simplified, and the test point logic might be optimized into the surrounding logic.

Force Test Points

Force test points allow a signal to be overridden (always force) throughout the entire test program. They are typically used to block some other value (such as an X value) from propagating.

The following force test point types are available:

- `force_0`
- `force_1`
- `force_01`

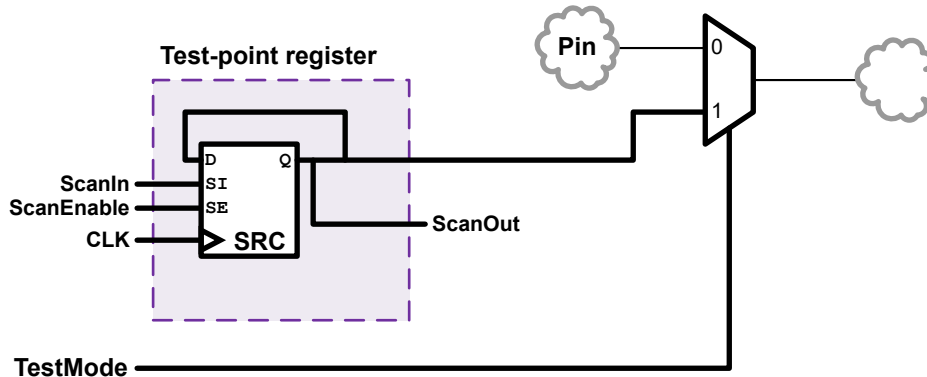
The `force_0` and `force_1` test point types allow a signal to be replaced with a constant 0 or constant 1 value throughout the entire test session. These test point types are useful when a particular signal must be forced to a known value for testability purposes. A logic gate is used to replace the original signal with a fixed constant 0 or 1 value when the TestMode signal is asserted. See [Figure 93](#).

Figure 93 Example of a `force_0` or `force_1` Test Point



The `force_01` test point type allows a signal to be replaced with a scan-selected value throughout the entire test session. A multiplexer is used to replace the original signal with the output of this scan register when the TestMode signal is asserted. See [Figure 94](#).

Figure 94 Example of a `force_01` Test Point



The forced value can vary per-pattern (as the scan register reloads with each pattern), but it remains constant for all capture cycles within a given pattern.

Control Test Points

Control test points allow a hard-to-control signal to be controllable (selectively forced) for some test vectors but not others. They are typically inserted to increase the fault coverage

of the design. They provide some control while still allowing some observation of upstream logic.

The following control test point types are available:

- `control_0`
- `control_1`
- `control_01`

A `control_0` or `control_1` test point is built with a controlling logic gate, an enabling AND gate, and a source scan register. When TestMode is not asserted, the signal always retains its original value. When TestMode is asserted, the signal is forced with a fixed constant 0 or 1 value only when the output of the scan register selects the constant value. This allows the test program to select either the original signal behavior or the constant-forced behavior on a per-pattern basis. See [Figure 95](#) and [Figure 96](#).

Figure 95 Example of a `control_1` Test Point

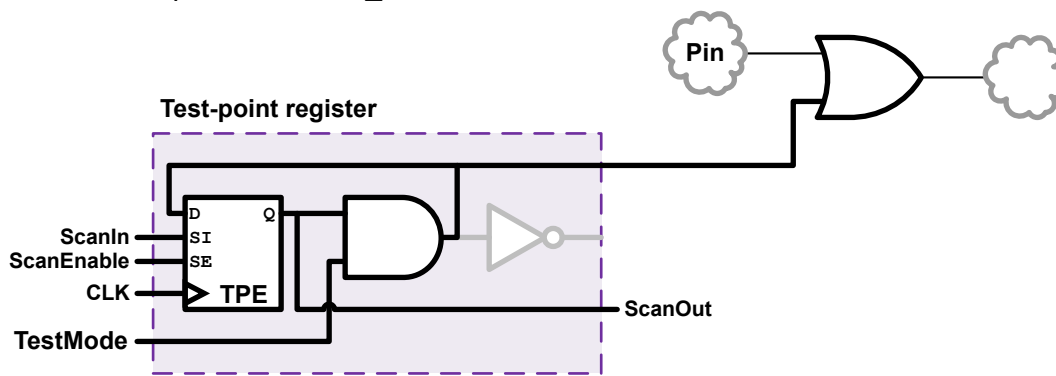
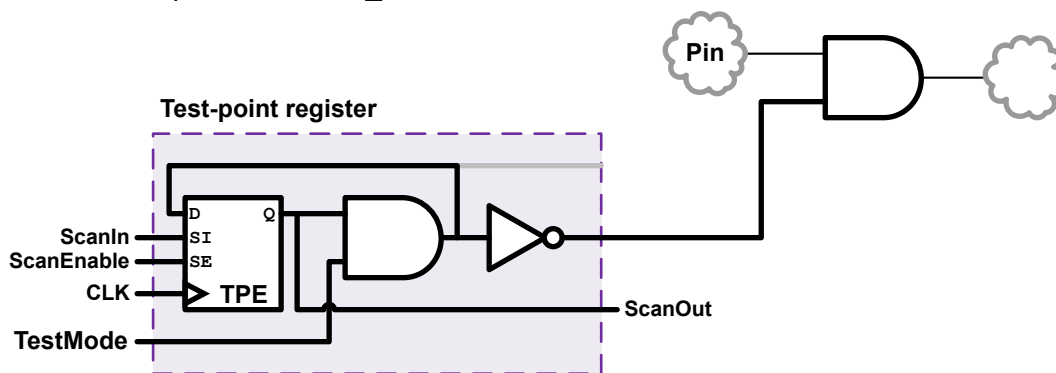


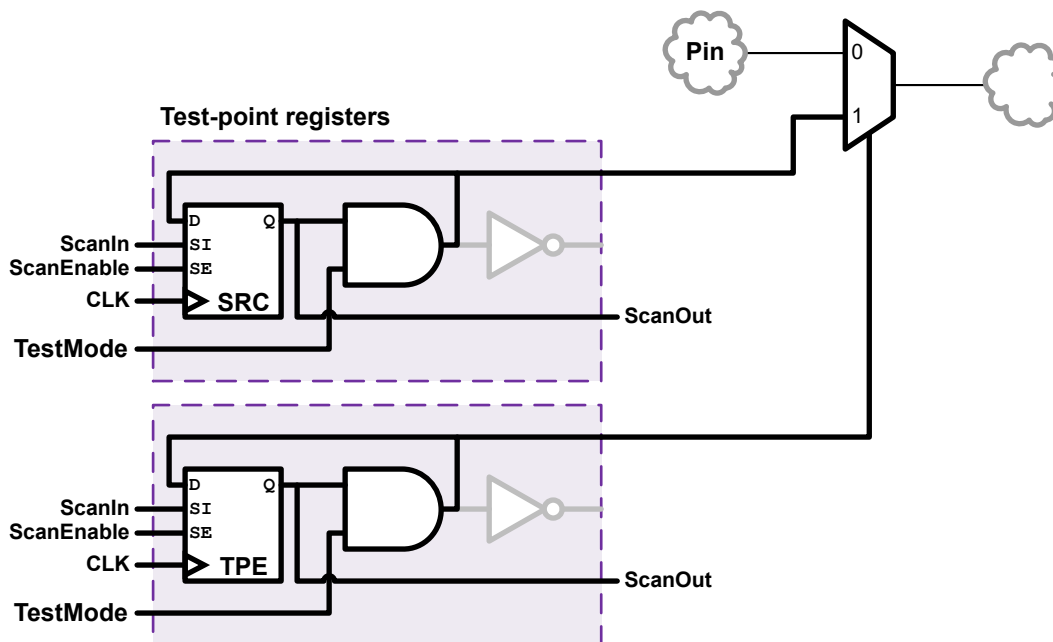
Figure 96 Example of a `control_0` Test Point



A `control_01` test point is similar to the `control_0` and `control_1` test point types, except that a scan-selected source signal value from a scan register is selectively driven onto the net on a vector-by-vector basis. As a result, the `control_01` test point requires

two scan cells per control point, one for the source signal value and one for the enable register that specifies that the source signal should be driven. See [Figure 97](#).

Figure 97 Example of a control_01 Test Point



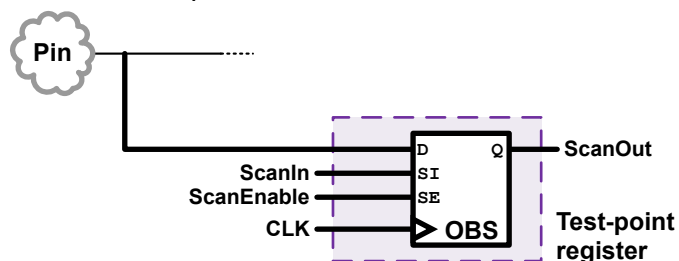
A control point can be enabled or disabled per-pattern, but its assertion behavior remains constant across capture cycles within a given pattern.

Observe Test Points

The `observe` test point type is typically inserted at hard-to-observe signals in a design to reduce test data volume or to increase coverage.

An `observe` test point is a scan register with its data input connected to the signal to be observed. See [Figure 98](#).

Figure 98 Example of an observe Test Point



Multicycle Test Points

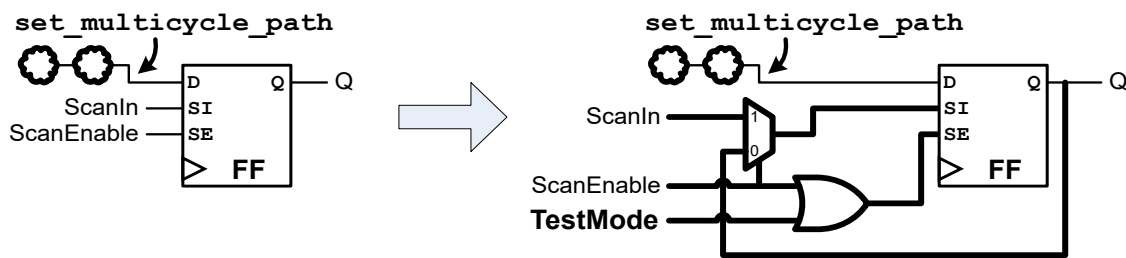
During ATPG, when a scan cell captures a value from a logic path constrained by a multicycle path exception, it captures a dynamic X value because the multicycle logic might not be stable by the capturing clock edge. These captured X values can affect other captured values in compressed designs, and they can propagate through the scan cell into other areas of the design when fast sequential ATPG is used.

The multicycle test point prevents these X values from being captured. It implements the following scan capture behavior:

- When the TestMode signal is asserted, the register holds state instead of capturing.
- When the TestMode signal is deasserted, the register captures normally.

Scan shift operation and functional operation are unaffected. This test point uses reconfigured scan path logic to avoid inserting logic along the functional path. See [Figure 99](#).

Figure 99 Example of a Multicycle Test Point



The multicycle test point does not provide coverage for the blocked capture path. However, it does prevent multicycle X values from propagating into scan compression logic or self-test logic.

Multicycle test points can only be inserted using automatic test point insertion.

Test Point Structures

The following topics describe how test point logic is structured:

- [Test Point Components](#)
- [Test Point Register Clocks](#)
- [Sharing Test Point Registers](#)

Test Point Components

Test points are constructed from (up to) three primary components:

- **Pin**

The functional pin where the test point is inserted to assert its behavior. This is the pin where the test point is located. Every test point has a corresponding insertion pin.

- **Register**

A scan-controllable register that provides source (driving) or sink (capturing) capability to the test point logic. A single register can be shared by multiple test points. Some test points do not use a register.

- **Control signal**

The TestMode or IbistEnable signal that activates the test point logic.

[Table 8](#) summarizes which components are used by each type of test point.

Table 8 *Components Used by Each Type of Test Point*

Test point type	Source register?	Sink register?	Control signal?
force_0, force_1			X
force_01	X		X
control_0, control_1	X		X
control_01	X (two registers)		X
observe		X	X

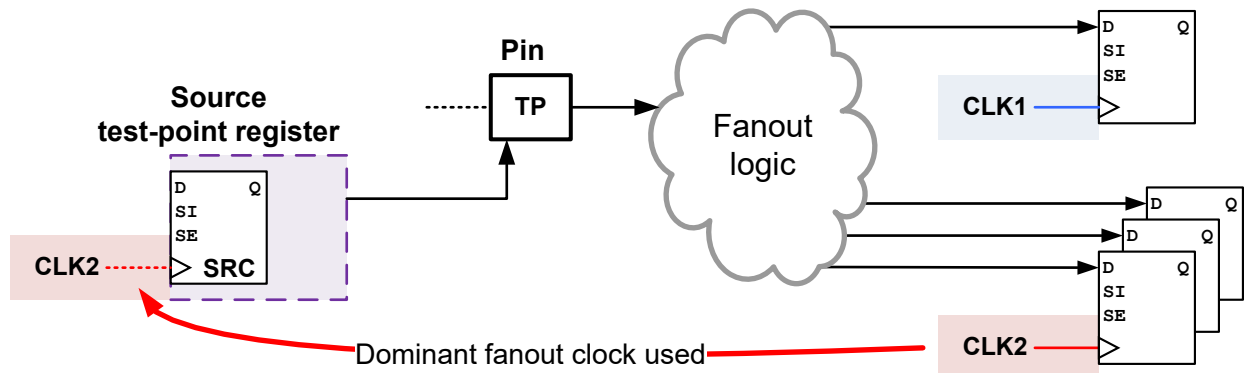
Test point registers do not have a reset signal; their value is set by scan shift when used and blocked by the control signal when not used.

Test Point Register Clocks

As described in [Test Point Components](#), some test point types use a register to drive (source) or capture (sink) data. By default, the tool clocks this register with the same clock as the surrounding logic.

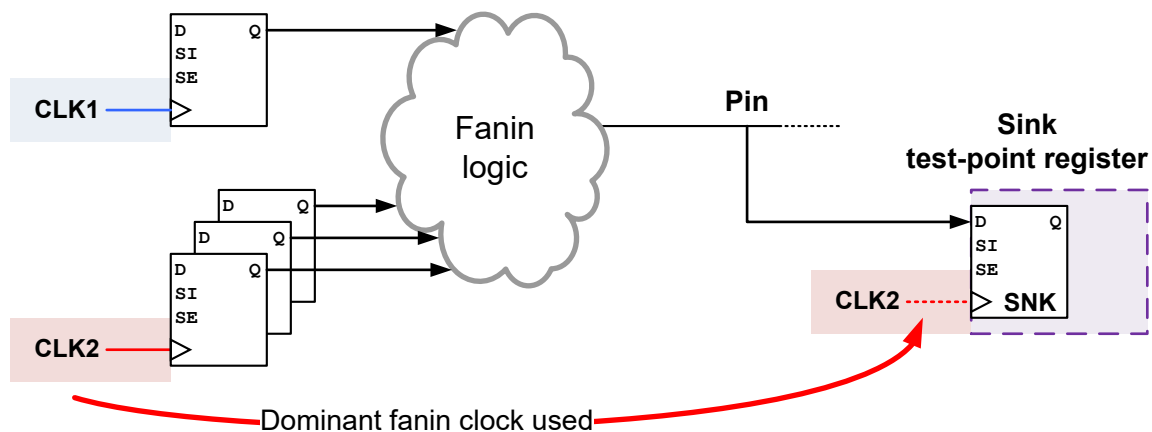
For source registers, the tool uses the dominant clock of the fanout registers, which capture data propagating from the test point:

Figure 100 Using Dominant Fanout Clock



For sink registers, the tool uses the dominant clock of the fanin registers, which drive data that propagates to the test point:

Figure 101 Using Dominant Fanin Clock



- You can specify the name of a scan clock signal, defined as a `ScanClock` signal type with the `set_dft_signal` command.
- In a DFT-inserted OCC controller flow, you can specify the name of a PLL output pin. In this case, the tool maps the test point clock to the output pin of the corresponding OCC controller during DFT insertion.
- In a user-defined OCC controller flow, you can directly specify the name of an output pin of an existing OCC controller.

For multivoltage designs, the register is associated with the power domain of the block in which the test point is inserted.

By default, to reduce the area overhead of test point logic from TestMAX Advisor analysis, Fusion Compiler shares each test point register with multiple test points.

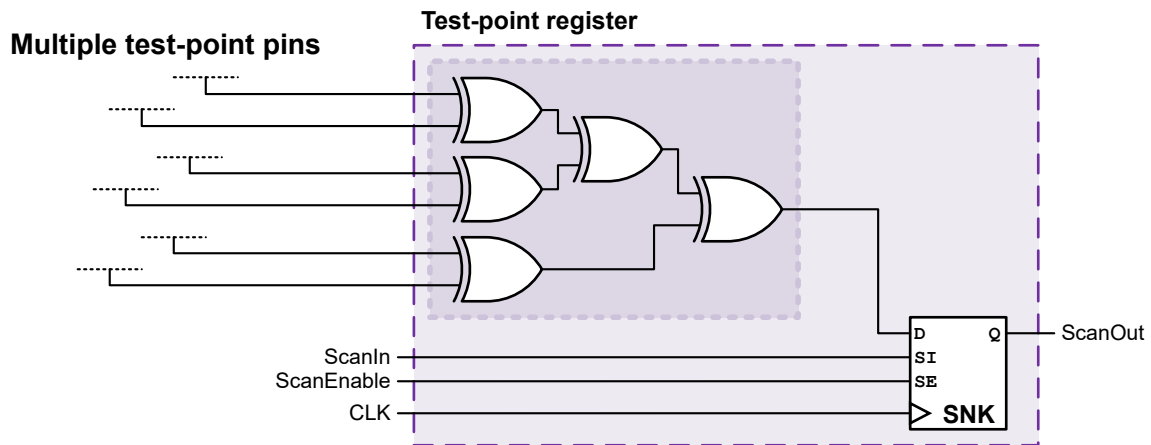
A source or enable test point register can be shared with multiple force or control test point pins. No additional logic gates are required; the register outputs are tied to multiple test point logic gates. [Figure 102](#) shows the logic for multiple `control_0` and `control_1` test points that share the same enable register.

[illegible]

Sharing Sink Registers

A sink test point register can be shared with multiple observe test point pins. The tool builds an XOR reduction tree which collapses multiple observed signals down to a single sink signal connected to the data input of the sink register. See [Figure 103](#).

Figure 103 Shared Sink Register For Multiple observe Test Points



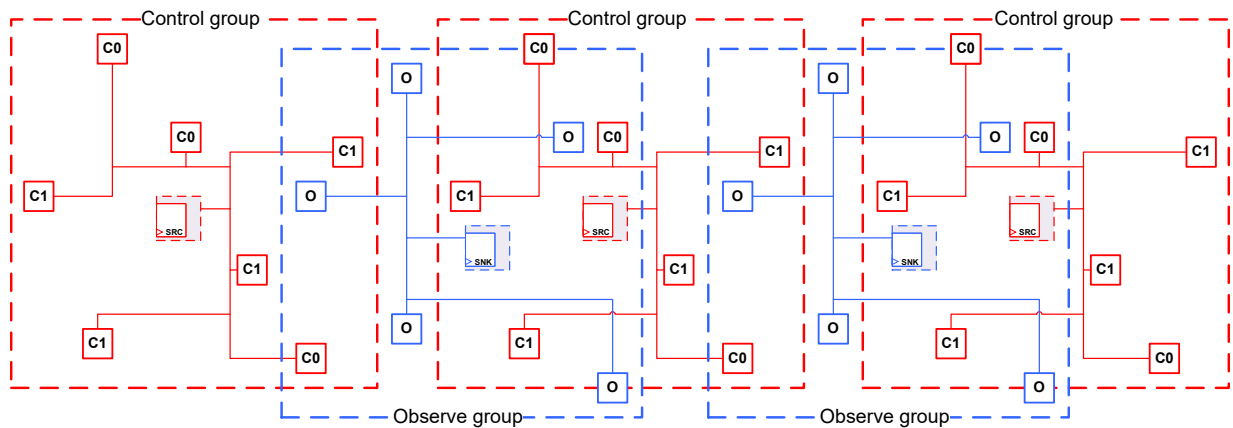
Sharing Rules

The test point pins within each sharing group share the same test point type, clock domain, and power domain. The pins are chosen to be in close physical or logical proximity. The maximum number of pins in a group is set by the `-test_points_per_scan_cell` option of the relevant test point configuration command. The register is inserted in the lowest hierarchy level common to all pins.

A register cannot be shared as both a source (data) and enable (control) register.

In Fusion Compiler, the tool groups pins that are in close physical proximity, then creates the test point register within the group. Test point groups of differing type, clock domain, or power domain are created independently and thus might overlap.

Figure 104 Shared Source Registers (Red) and Sink Registers (Blue)



Automatically Inserted Test Points

You can automatically insert test points in your design to improve its testability. With this feature, Fusion Compiler calls the TestMAX Advisor tool to analyze the design and determine an optimal set of test points, then it implements them during synthesis.

You can use one or more test point *targets*, each focusing on a different aspect of testability:

- `random_resistant`

This target inserts test points that improve random-pattern coverage. This improves the coverage for a given pattern count. It can also improve the maximum coverage obtainable for the design.

- `untestable_logic`

This target inserts test points that make untestable logic testable. This improves the maximum coverage obtainable for a design. It also improves the coverage for a given pattern count.

- `x_blocking`

This target inserts test points to block X values at their sources so they cannot propagate into downstream logic and be captured.

- `multicycle_paths`

This target inserts test points to prevent multicycle-path-constrained logic from being captured by scan cells (which ATPG treats as an X value).

- `shadow_wrapper`

This target inserts *shadow wrappers* around untestable blocks or macrocells so that surrounding logic can be tested. The outputs are forced to known values and the inputs are observed to ensure coverage.

- `core_wrapper`

This target inserts a test-point-based wrapper chain at the data I/O ports of the current design. Only inward-facing (INTEST) functionality is provided, but test point register sharing ensures very low overhead.

- `user`

This target inserts test points whose types and locations are provided by the user.

The following topics describe how to configure automatic test point insertion:

- [Enabling Automatic Test Point Insertion](#)
- [Configuring Global Test Point Insertion Settings](#)
- [Configuring the Random-Resistant Test Point Target](#)
- [Configuring the Untestable Logic Test Point Target](#)
- [Configuring the X-Blocking Test Point Target](#)
- [Configuring the Multicycle Path Test Point Target](#)
- [Configuring the Shadow Wrapper Test Point Target](#)
- [Configuring the Core Wrapper Test Point Target](#)
- [Configuring the User-Defined Test Point Target](#)
- [Enabling Multiple Targets in a Single Command](#)
- [Implementing Test Points From an External File](#)
- [Customizing the Test Point Analysis](#)
- [Running Test Point Analysis](#)
- [Automatic Test Point Insertion Example Script](#)
- [Limitations](#)

Enabling Automatic Test Point Insertion

To enable automatic test point insertion, issue the following command before pre-DFT DRC:

```
fc_shell> set_dft_configuration -testability enable
```

Note:

A TestMAX Advisor or Spyglass® DFT ADV license is required to use the automatic test point insertion feature.

Then, use the `set_testability_configuration` command to configure one or more automatic test point targets:

- To configure global settings, which are shared by all targets, omit the `-target` option.
- To enable a particular target (and to optionally configure any target-specific options it supports), specify that target with the `-target` option.

Configuring Global Test Point Insertion Settings

To configure global aspects of automatic test point insertion, use the `set_testability_configuration` command without the `-target` option:

```
set_testability_configuration
[-clock_signal clock_name]
[-control_signal control_name]
[-test_points_per_scan_cell n]
[-user_options file_name]
[-max_test_points n]
```

Table 9 shows the global configuration options.

Table 9 Global `set_testability_configuration` Options

To do this	Use this option
Use a single dedicated clock for all test point registers in the design	<code>-clock_signal <i>clock_name</i></code> (default is the dominant clock)
Use a particular TestMode signal to enable the test points	<code>-control_signal <i>control_name</i></code> (default is the first available TestMode signal)
Specify the number of force, control, or observe points that can share a test point register	<code>-test_points_per_scan_cell <i>n</i></code> (default is 8)
Customize the test point analysis with user-provided TestMAX Advisor Tcl commands	<code>-user_options <i>file_name</i></code>

Table 9 *Global set_testability_configuration Options (Continued)*

To do this	Use this option
Set the maximum number of <code>random_resistant</code> and <code>untestable_logic</code> test points (combined) that can be inserted	<code>-max_test_points n</code> (default is 5% of the total register count)

You can also use the `-clock_signal`, `-control_signal`, and `-test_points_per_scan_cell` options on a per-target basis to override the global values.

The `-max_test_points` option applies to the `random_resistant` and `untestable_logic` targets. When both targets are enabled, the analysis automatically allocates the global limit across them. Any per-target `-max_test_points` limits apply in addition to, not in place of, the global limit. For details, see [SolvNetPlus article 3021459](#), “How Does The `-max_test_points` Option Work?”

Configuring the Random-Resistant Test Point Target

You can use the `random_resistant` test point target to improve the testability of hard-to-test logic in your design. This improves the coverage for a given pattern count. It can also improve the maximum coverage obtainable and ATPG effectiveness for the design.

The random-resistant target invokes a TestMAX Advisor algorithm that determines an optimal set of test points that improves random-pattern test coverage for a given pattern count. It provides the following benefits:

- High capacity—No random pattern simulation is used; instead, the pattern count is a parameter in a mathematical probabilistic fault-detection analysis.
- Easy to use—The algorithm finds the optimum set of `control_0`, `control_1`, and observe test points; there is no need to guess at per-type limits.

To enable and configure the random-resistant target, use the `set_testability_configuration -target random_resistant` command. The remaining options, which are described in [Table 10](#), can be specified to override their defaults.

Table 10 *Random-Resistant set_testability_configuration Options*

To do this	Use this option
Override the corresponding global value of a parameter	<code>-clock_signal clock_name</code> <code>-control_signal control_name</code> <code>-test_points_per_scan_cell n</code>

Table 10 *Random-Resistant set_testability_configuration Options (Continued)*

To do this	Use this option
Control the runtime-versus-accuracy tradeoff of the analysis	<code>-effort low medium high</code> (default is <code>medium</code>)
Set the number of random-resistant test points at which the analysis completes. Random resistant analysis stops when either the test point or test coverage limit is met.	<code>-max_test_points n</code> (default is no target-specific limit)
Set the random-pattern test coverage value at which the analysis completes. Random resistant analysis stops when either the test point or test coverage limit is met.	<code>-target_test_coverage coverage_value</code> (default is <code>100</code>)
Specifies the random pattern count used for the analysis	<code>-random_pattern_count n</code> (default is <code>64000</code>)
Restrict test point insertion to only particular hierarchical cells or logic cones	<code>-include_elements cell_list</code> <code>-include_fanin_cone pin_port_list</code> <code>-include_fanout_cone pin_port_list</code> (default is to consider the entire design)
Exclude particular hierarchical cells or logic cones from test point insertion	<code>-exclude_elements cell_list</code> <code>-exclude_fanin_cone pin_port_list</code> <code>-exclude_fanout_cone pin_port_list</code> (default is not to exclude anything)

If you are inserting LogicBIST self-test, for best results, set the `-random_pattern_count` option to the number of self-test patterns.

See Also

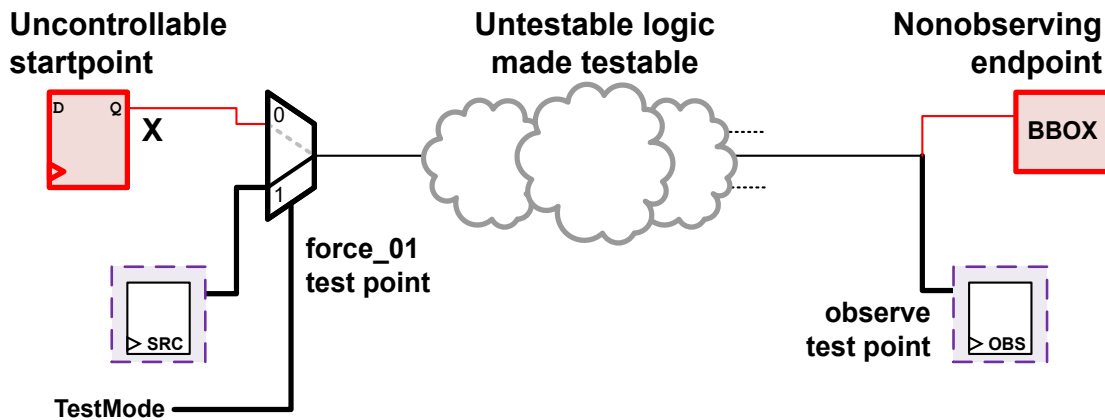
- [Info_random_resistance rule](#) documentation in the TestMAX Advisor documentation for details on the random-resistant analysis algorithm and its parameters

Configuring the Untestable Logic Test Point Target

You can use the `untestable_logic` test point target to make untestable logic testable. This improves the maximum coverage obtainable for a design. It also improves the coverage for a given pattern count.

The untestable logic target invokes a TestMAX Advisor algorithm that determines an optimal set of force_01 and observe test points to control uncontrollable nets and observe unobservable logic, respectively.

Figure 105 Untestable Logic Made Testable



The untestable logic target differs from the X-blocking target as follows:

- It inserts observe test points as well as force_01 test points.
- It performs gain-based analysis, which prefers test points with the highest testability improvement first.
- It considers the `-max_test_points` option.

To enable and configure the untestable logic target, use the `set_testability_configuration -target untestable_logic` command. The remaining options, which are described in [Table 11](#), can be specified to override their defaults.

Table 11 Untestable Logic `set_testability_configuration` Options

To do this	Use this option
Override the corresponding global value of a parameter	<code>-clock_signal clock_name</code> <code>-control_signal control_name</code> <code>-test_points_per_scan_cell n</code>
Set the number of untestable logic test points at which the analysis completes	<code>-max_test_points n</code> (default is no target-specific limit)
Restrict test point insertion to only particular hierarchical cells or logic cones	<code>-include_elements cell_list</code> <code>-include_fanin_cone pin_port_list</code> <code>-include_fanout_cone pin_port_list</code> (default is to consider the entire design)

Table 11 Untestable Logic `set_testability_configuration` Options (Continued)

To do this	Use this option
Exclude particular hierarchical cells or logic cones from test point insertion	<code>-exclude_elements cell_list</code> <code>-exclude_fanin_cone pin_port_list</code> <code>-exclude_fanout_cone pin_port_list</code> (default is not to exclude anything)

This target is highly recommended for designs with LogicBIST self-test.

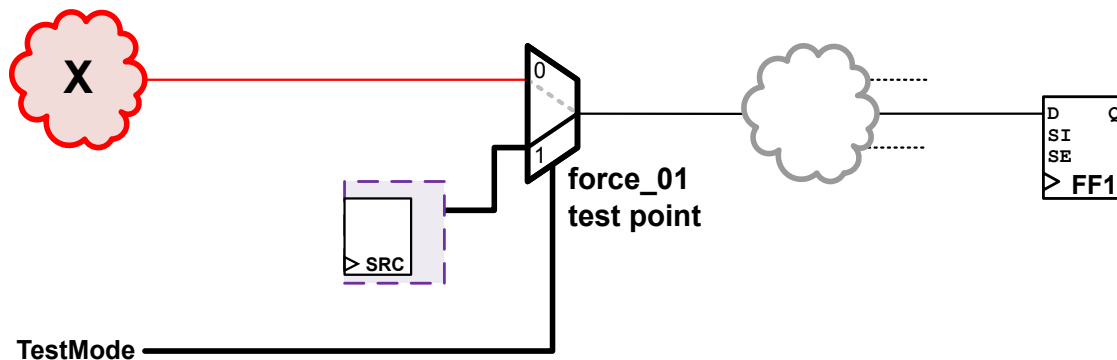
See Also

- [TA_10 rule](#) documentation in the TestMAX Advisor documentation for details on the untestable logic analysis algorithm

Configuring the X-Blocking Test Point Target

You can use the `x_blocking` test point target to identify and block X-value sources from black-box cells. This target inserts `force_01` test points at the sources to force scan-controllable values in place of the X values.

Figure 106 X-Value Source Blocked by `force_01` Test Point



The X-blocking target differs from the untestable logic target as follows:

- It blocks all X sources in the design regardless of gain improvement, which is important for designs with LogicBIST self-test.
- It does not consider the `-max_test_points` option.

To enable the X-blocking target, use the `set_testability_configuration -target x_blocking` command. The remaining options, which are described in [Table 12](#), can be specified to override their defaults.

Table 12 X-Blocking `set_testability_configuration` Options

To do this	Use this option
Override the corresponding global value of a parameter	<code>-clock_signal clock_name</code> <code>-control_signal control_name</code> <code>-test_points_per_scan_cell n</code>

This target blocks X values from black-box cells. It does not block values from registers with uncontrolled clock or asynchronous set/reset signals.

This target is highly recommended for designs with LogicBIST self-test.

Configuring the Multicycle Path Test Point Target

You can use the `multicycle_paths` test point target to prevent multicycle-path-constrained logic from being captured by scan cells (which ATPG treats as an X value). This target inserts multicycle path test points at multicycle-constrained data input pins of scan cells.

To enable the multicycle paths target, use the `set_testability_configuration -target multicycle_paths` command. The remaining options, which are described in [Table 13](#), can be specified to override their defaults.

Table 13 Multicycle Paths `set_testability_configuration` Options

To do this	Use this option
Override the corresponding global value of the parameter	<code>-control_signal control_name</code>

You must apply all multicycle constraints before test point analysis is run.

See Also

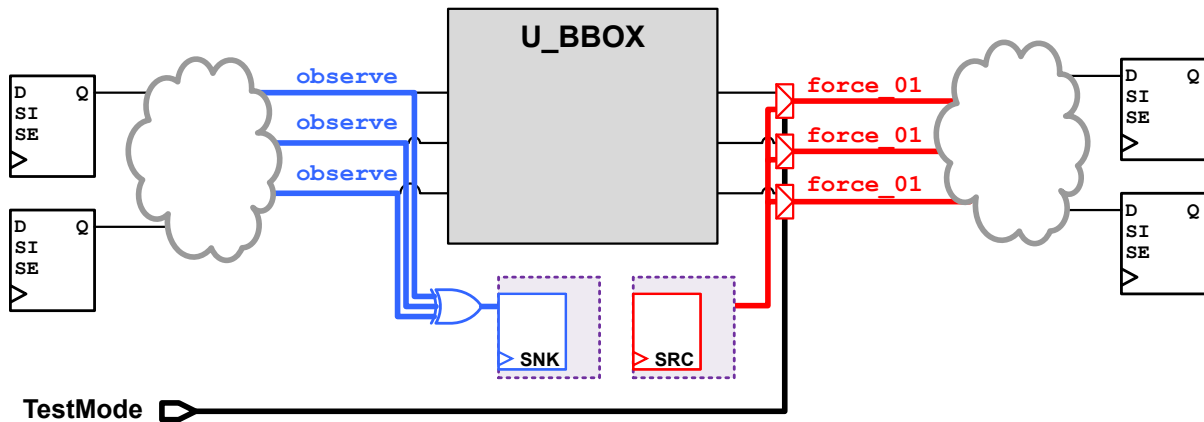
- [Atspeed_05 rule](#) documentation in the TestMAX Advisor documentation for details on the multicycle paths analysis rule
- [Multicycle Test Points](#) for information on the multicycle path test point

Configuring the Shadow Wrapper Test Point Target

You can use the `shadow_wrapper` test point target to allow logic around untestable blocks or macrocells to be tested.

This target inserts `force_01` test points at data output pins to drive known values, and it inserts `observe` test points at data input pins to ensure coverage. If the functional logic around a pin already allows for testability, the analysis detects this and does not insert a test point.

Figure 107 Shadow Wrapper Around a Black-Box Cell



To enable the shadow wrapper target, use the `set_testability_configuration -target shadow_wrapper -isolate_elements cell_list` command.

The `-target shadow_wrapper` option enables the shadow wrapper target and the `-isolate_elements` option specifies the list of cells to isolate; both are required. The remaining options, described in [Table 14](#), can be specified to override their defaults.

Table 14 Shadow Wrapper `set_testability_configuration` Options

To do this	Use this option
Override the corresponding global value of a parameter	<code>-clock_signal clock_name</code> <code>-control_signal control_name</code> <code>-test_points_per_scan_cell n</code>

Only hierarchical, black-box, and CTL-modeled cells can be isolated.

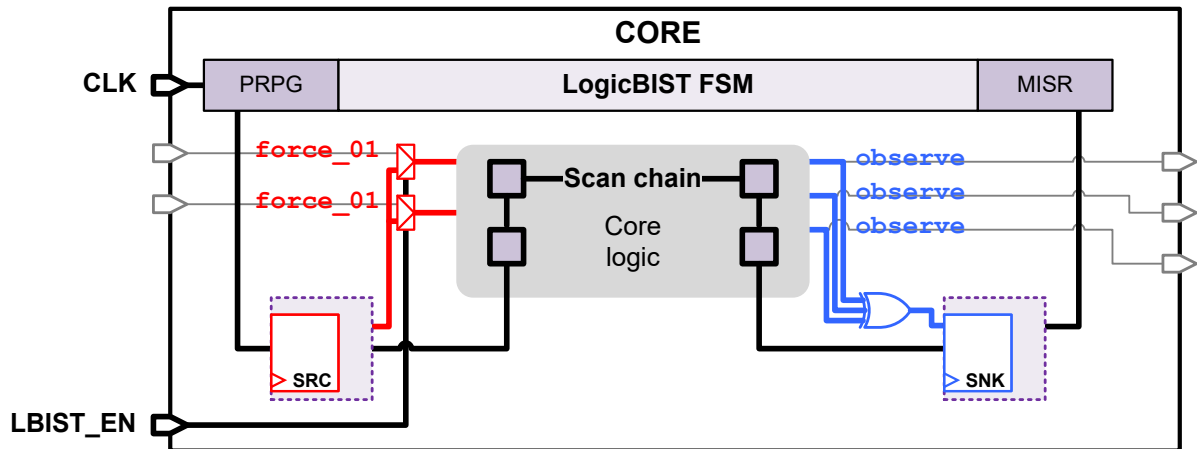
Configuring the Core Wrapper Test Point Target

You can use the `core_wrapper` test point target to insert an inward-facing-only wrapper chain, constructed using test points, at the data I/O ports of the current design.

This target inserts `force_01` test points at data input ports to drive known values, and it inserts observe test points at data output ports to ensure coverage. If the functional logic around a port already allows for testability, the analysis detects this and reuses the functional registers instead of inserting a test point.

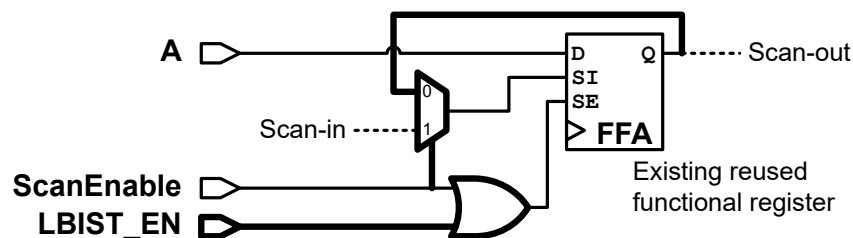
This is not a replacement for full core wrapping in a hierarchical test flow. Instead, it provides a lightweight core isolation capability suitable for unwrapped designs with built-in self-test (BIST). This isolation method is area-efficient, especially when reused registers are used.

Figure 108 Test-Point-Based Wrapper Chain and LogicBIST Self-Test Logic



If an input port is directly registered, its functional register is reused to implement the `force_01` test point by modifying the scan-enable connection to hold state during capture, as shown in Figure 109. Similarly, functional output registers can take the place of observe test points. Small amounts of logic can be permitted between ports and their registers by adjusting fanout and depth threshold values.

Figure 109 Functional Input Register Reused as State-Holding `force_01` Test Point



Just as with the full core-wrapping DFT client, clock, test, and asynchronous set/reset ports are not wrapped. See [Wrapper Cells and Wrapper Chains](#) for details.

To enable this lightweight core wrapper target, use the `-set_testability_configuration -target core_wrapper` command. The remaining options, which are described in [Table 15](#), can be specified to override their defaults. They work the same as described in [Setting a Reuse Threshold](#).

Table 15 Core Wrapper `set_testability_configuration` Options

To do this	Use this option
Override the corresponding global value of a parameter	<code>-clock_signal clock_name</code> <code>-control_signal control_name</code> <code>-test_points_per_scan_cell n</code>
Specify the maximum number of functional I/O registers allowed for reuse before adding a test point	<code>-reuse_threshold threshold_value</code> (default is 0)
Set the maximum number of combinational logic levels (including buffers and inverters) allowed for reuse before adding a test point	<code>-depth_threshold threshold_value</code> (default is 1)
Exclude particular ports from being wrapped	<code>-exclude_elements port_list</code> (default is not to exclude ports)

Configuring the User-Defined Test Point Target

The `user` test point target controls the implementation of user-defined test points.

This target applies to the following:

- User-defined test points specified by the `-test_point_file` option
Test points not from TestMAX Advisor in an external file are assigned to the `user` test point target. User-defined test points from an external file are not implemented unless the `user` target is enabled.
- The `set_test_point_element` command
User-defined test points defined with the `set_test_point_element` command are always implemented, whether the `user` target is enabled or not. However, the `user` target allows you to specify implementation defaults for them.

To enable the user-defined target, use the `set_testability_configuration -target user` command.

The `-target user` option is required to use test points from an external file. The remaining options, which are described in [Table 16](#), can be specified to override their defaults.

Table 16 *User-Defined set_testability_configuration Options*

To do this	Use this option
Override the corresponding global value of a parameter	<code>-clock_signal clock_name</code> <code>-control_signal control_name</code> <code>-test_points_per_scan_cell n</code>

Note:

If you specify the tool to insert user-defined test points on some registers, but then change the names of those registers using the `change_names` command, then the tool does not insert the user-defined test points.

See Also

- [Implementing Test Points From an External File](#) for details on TestMAX Advisor and user-defined test point files

Enabling Multiple Targets in a Single Command

You can enable multiple targets in a single `set_testability_configuration` command by specifying a list with the `-target` option:

```
fc_shell> set_testability_configuration \
           -target {random_resistant x_blocking}
Information: Creating testability configuration for target
'random_resistant'.
Information: Creating testability configuration for target 'x_blocking'.
Accepted testability configuration specification for design 'top'.
1
```

If you specify an option that pertains to some targets but not others, the tool warns of unsupported target/option combinations (but accepts the valid combinations):

```
fc_shell> set_testability_configuration \
           -target {random_resistant x_blocking} -max_test_points 100
Warning: The '-max_test_points' option does not apply to the
'x_blocking' target. (UIT-1810)
Information: Creating testability configuration for target
'random_resistant'.
```

```
Information: Creating testability configuration for target 'x_blocking'.  
Accepted testability configuration specification for design 'top'.  
1
```

For suggestions on resolving the UIT-1810 warning message, see its man page.

Implementing Test Points From an External File

You can implement test points defined in an external file by using the `-test_point_file` option of the `set_testability_configuration` command:

```
set_testability_configuration  
  -test_point_file file_name
```

The `-test_point_file` option is a global option that specifies the name of the test point file to implement.

The test point file must be formatted as follows:

- A test point is described by a line containing three fields, separated by whitespace:
 - A user-defined label (not used by the tool; can be any string)
 - The test point type keyword
 - The net where the test point should be inserted
- For net names, the hierarchy separator character can be “/” (used by the synthesis tools) or “.” (used by the TestMAX Advisor tool).
- Text after “#” is a comment and is ignored.
- Blank lines are ignored.

Note that file-based test points are not implemented unless their targets are enabled (just as with analysis-based test points). Test points are assigned to targets as described in the following sections.

Using User-Defined Test Point Files

Test points from user-defined test point files are assigned to the `user` test point target.

[Example 13](#) shows an example user-defined test point file.

Example 13 Example User-Defined Test Point File

```
# Put these memories into test mode  
* force_1 MEM0_0/TSTMODE  
* force_1 MEM0_1/TSTMODE  
* force_1 MEM1_0/TSTMODE  
* force_1 MEM1_1/TSTMODE
```

```
# these test points come from our in-house Perl script
TP1 control_1 core0/U37641/NET1 # my_algorithm.pl result: Q=478 V=479
TP2 control_0 core4/U73087/NET2 # my_algorithm.pl result: Q=861 V=22
TP3 control_0 core0/U64599/NET3 # my_algorithm.pl result: Q=227 V=964
TP4 control_1 core4/U99749/NETN # my_algorithm.pl result: Q=841 V=9
```

Using TestMAX Advisor Test Point Files

Test points from a TestMAX Advisor test point file (with in-line comments) are routed to their corresponding targets. [Example 14](#) shows an example test point file generated by the TestMAX Advisor tool that contains `random_resistant` and `x_blocking` test points.

Example 14 Example TestMAX Advisor Test Point File

```
# Moment : "TP analysis start " # Time : 2017- 1-30 9:52:19
# Design : "top"
# Initial Random Pattern Test Coverage : 80.68602
# Stuck At Test Coverage : 99.44180
# Random Pattern Count : 10
# Effort Level : medium
# Requested Test Points : 1
# Cut-Off Gain : 0.00000, (cumulative gain of last '1' test points)
# Thread Count : 8

# Index Test_Point_Type Net Comment
1 observe sub3.N50 #Gain : 0.04860 Cov : 80.73462 S@TC : 99.4
# Search completed : 1 (dft_rrf_tp_count(1 - 1)) test points identified.
# BENCHMARK : "COMPLETE TP ANALYSIS " # Time : 5.7086
# Moment : "TP analysis End " # Time : 2017- 1-30 9:52:24

# X Source Section
# Index force_01 net
* force_01 sub2.nZ[0] # -xsource -cell_name sub2.BBOX
* force_01 sub2.nZ[1] # -xsource -cell_name sub2.BBOX
* force_01 sub2.nZ[2] # -xsource -cell_name sub2.BBOX
* force_01 sub2.nZ[3] # -xsource -cell_name sub2.BBOX
```

Using TestMAX Advisor Test Point Files Without Rerunning Analysis

By default, analysis is performed for all enabled targets. However, when implementing test points from a TestMAX Advisor file, you can prevent analysis from rerunning for its targets by specifying the `-only_from_file true` option for those targets. Analysis is still performed for any targets enabled without this option.

The following command implements `random_resistant` and `x_blocking` test points from a TestMAX Advisor file, then also implements shadow-wrapper test points that are not described in the file:

```
# global options - specify test point file
set_testability_configuration -test_point_file my_rrf_xblocking_TPs.txt
```

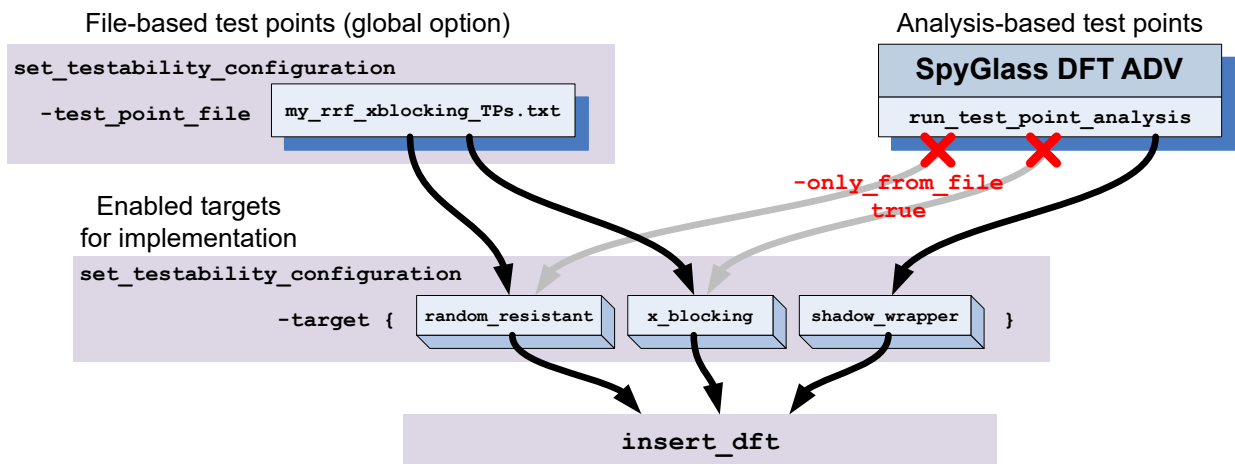


```
# enable targets for implementation
## implement these from the existing TestMAX Advisor file:
set_testability_configuration -target {random_resistant} \
    -only_from_file true
set_testability_configuration -target {x_blocking} \
    -only_from_file true

## derive these in the current run (during run_test_point_analysis)
set_testability_configuration -target {shadow_wrapper} \
    -isolate_elements {IP_BLOCK1 IP_BLOCK2}
```

Figure 110 shows how the test points are populated in each target for this example.

Figure 110 Combining File-Based and Analysis-Based Test Points



If you implement TestMAX Advisor test points from a file by enabling its target without specifying the `-only_from_file true` option, both the file-based and analysis-based test points are implemented for that target.

Customizing the Test Point Analysis

To customize the test point analysis, you can provide a user file containing one or more TestMAX Advisor Tcl commands to be included:

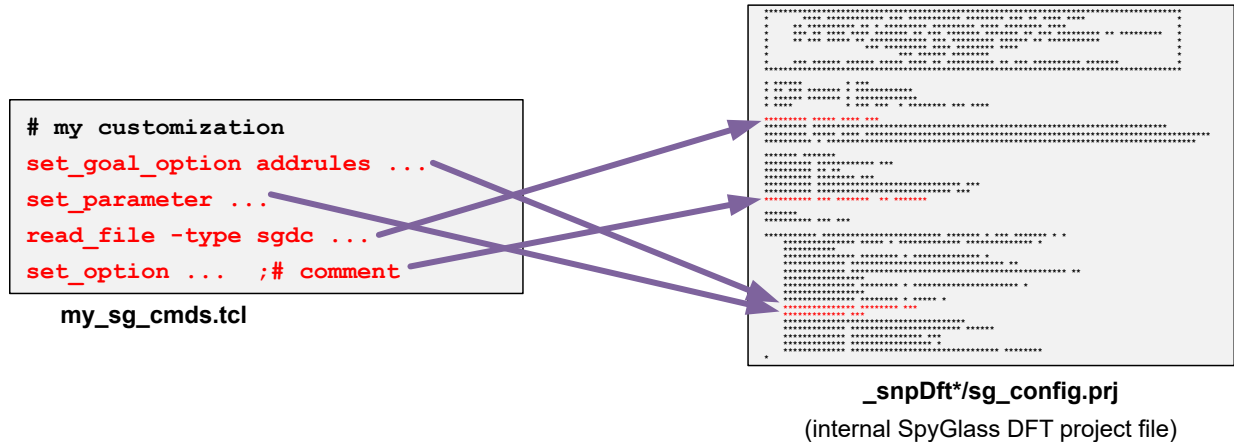
```
fc_shell> set_testability_configuration -sg_command_file my_sg_cmds.tcl
```

This user file is an unordered list of one or more of the following commands:

```
read_file
set_parameter
set_option
set_goal_option
```

The TestMAX Advisor analysis automatically places the commands at the appropriate points in the internal analysis script (also known as the TestMAX Advisor project file):

Figure 111 Including User Commands in the TestMAX Advisor Test Point Analysis



Note:

TestMAX Advisor design constraint (*.sgdc) commands cannot be placed directly in the user file, but they can be placed in a separate file and applied by a `read_file -type sgdc` command in the user file.

Comments after commands are preserved; full-line comments are not.

After the TestMAX Advisor analysis completes, you can examine the TestMAX Advisor project file by looking at the `sg_config.prj` file in the newly created `_snpDft*` directory.

Running Test Point Analysis

If you have configured testability analysis features that require TestMAX Advisor analysis, Fusion Compiler automatically invokes SpyGlass and performs the analysis during the `compile_fusion` command.

The `SPYGLASS_HOME` environmental variable (not Tcl application variable) must be set to find the `spyglass` binary. You can check this as follows:

```
fc_shell> echo $env(SPYGLASS_HOME)
/global/apps/spyglass_2016.06-SP2
```

Automatic Test Point Insertion Example Script

The following script inserts several kinds of testability test points, using a test-mode control signal named `TM_TESTPOINTS`.

Chapter 14: Inserting Test Points

Automatically Inserted Test Points

```
# define DFT signals
set_dft_signal -view existing_dft -type ScanClock \
  -port CLK -timing [list 45 55]
set_dft_signal -view spec -type TestMode \
  -port TM_TESTPOINTS

# enable automatic test point insertion
set_dft_configuration -testability enable

# configure global automatic test point insertion settings
# (shared by all targets)
set_testability_configuration \
  -control_signal TM_TESTPOINTS \
  -test_points_per_scan_cell 16

# enable and configure test point targets
set_testability_configuration \
  -target random_resistant \
  -random_pattern_count 1024

# insert DFT IP (if needed)
create_test_protocol
preview_dft
insert_dft

# perform synthesis *and* test point analysis/insertion
compile_fusion
```

Limitations

Automatic test point insertion has the following limitations:

- The `-control_signal` option *must* be specified, otherwise test point insertion is skipped.
- Test point registers are always positive-edge, even if the dominant clocking around a test point is negative-edge.
- When automatic test point insertion is enabled, do not run the `change_names` command between the `run_test_point_analysis` and the `preview_dft` commands, the `run_test_point_analysis` and `insert_dft` commands, or the `preview_dft` and `insert_dft` commands.

Inserting the Test Point Logic

When the test point feature is enabled, the `compile_fusion` command does the following:

- Runs TestMAX Advisor analysis (if needed)
- Inserts test point logic in design
- Creates a test point report file

Test point scan registers are placed in the lowest level of hierarchy common to all test points for that register.

The test point report file, `testpoint.rpt`, shows the implemented test points, test point registers, and register sharing characteristics:

```
***** Test Point Plan Report *****
Test point register attributes:
  d - dedicated (DFT-inserted) test point register
  f - reused (functional) test point register
  tpe - test point enable signal
  src - test point source signal
  snk - test point sink signal

Test point pin attributes:
  r - random_resistant test point pin
  x - X-blocking test point pin
  m - multicycle path test point pin
  w - core wrapper test point pin
  s - shadow wrapper test point pin
  g - self-gating test point pin
  a - AutoFix test point pin
  u - user-defined test point pin

Index  Test Point Register  Test Point Type  Pins
-----
1      sub3/mult_60/U_dft_tp_sdtc_ip_2/dtc_reg ( d, tpe ) ( CLK3 )
      control_0          sub3/mult_60/U1057/Z (r)
      control_1          sub3/mult_60/U1156/Z (r)
      control_1          sub3/mult_60/U1262/Z (r)
      control_0          sub3/mult_60/U1343/Z (r)
2      sub3/mult_60/U_dft_tp_sdtc_ip_3/dtc_reg ( d, snk ) ( CLK3 )
      observe            sub3/mult_60/U4479/Z (r)
      observe            sub3/mult_60/U4483/Z (r)
      observe            sub3/mult_60/U4487/Z (r)
      observe            sub3/mult_60/U4596/Z (r)

***** Test Point Summary *****
Number of testability control_0 test points:      2
Number of testability control_1 test points:      2
Number of testability observe test points:        4
Total number of test points:                      8
```

```
Total number of DFT-inserted test point registers: 2
*****
```

Inserting Power-Aware Test Points

Test points are points in the design where the Fusion Compiler tool inserts logic to improve the testability of the design.

This feature enables UPF checks for the test point location and test point controller logic.

To enable power-aware test point insertion, use the `set_app_options -name dft.tp_enable_mv_checks -value true` command. UPF checks are done on the nets between the existing driver and its loads. The test points are rejected if the UPF check fails on the nets.

15

Previewing, Inserting, and Checking DFT Logic

This chapter describes how to preview and insert your DFT IP, insert optimized scan structures during synthesis, and check the resulting DFT logic for correct operation.

This chapter includes the following topics:

- [Previewing the DFT IP](#)
- [Inserting the DFT IP](#)
- [Pre-DFT DRC](#)
- [Inserting Scan Structures](#)
- [Post-DFT DRC](#)
- [Customizing the DFT DRC Analysis](#)

Previewing the DFT IP

If you use any of the following elaborated-RTL DFT features, you can use the `preview_dft` command to preview the DFT IP before inserting it with the `insert_dft` command:

- On-chip clocking (OCC) controllers
- Scan compression

The `preview_dft` command also shows the configuration for the following DFT features (although these alone do not enable the `preview_dft` command):

- Core wrapping
- Pipelined scan data
- DFT partitions
- Test modes

For example,

```
fc_shell> preview_dft
- Partition: 'default_partition'
Information: Architecting DFT Compressor IP with '2' inputs / '2' outputs
/ '4' internal chains in mode 'ScanCompression_mode'
Information: Compressor will have 100% x-tolerance
Warning: Architecting unload compressor with no direct diagnosis support
ScanDataIn:
    "SI" -> U_DFT_TOP_IP_0/si[0]
    "SI2" -> U_DFT_TOP_IP_0/si[1]
ScanDataOut:
    "SO" -> U_DFT_TOP_IP_0/so[0]
    "SO2" -> U_DFT_TOP_IP_0/so[1]
ScanEnable:
    "SE" -> U_DFT_TOP_IP_0/test_se
    "SE" -> myOCC/test_se
OCC Bypass:
    "OCC_bypass" -> myOCC/p11_bypass
OCC Reset:
    "OCC_reset" -> myOCC/p11_reset
OCC ATE Clocks:
    "ATECLK" -> myOCC/slow_clk_0
OCC TestMode:
    "TM_OCC" -> myOCC/test_mode_0
OCC PLL Clocks:
    "CLK" -> myOCC/fast_clk_0[0] -> myOCC/clk_0[0]
OCC PLL Chains:
    "SI_OCC" -> myOCC/cc_si_0[0]
    myOCC/cc_so_0[0] -> "SO_OCC"
Clock Controller 0 : myOCC
    Number of bits per clock: 4
    Clock chain count: 1
    Fast clock pins: "CLK"
    Slow clock: "ATECLK"
    Test Mode: "TM_OCC"
Scan path: OCC_CHAIN (o)
    "SI_OCC" -> "SO_OCC" { myOCC:0 }
1
```

If no DFT features are enabled that require elaborated-RTL insertion, the command issues the following message:

```
fc_shell> preview_dft
Warning: There is no DFT IP specification to be reported (DFT-2003)
1
```

Inserting the DFT IP

If you use any of the following DFT features, you must use the `insert_dft` command before compile to insert the DFT IP in elaborated-RTL form:

- Scan compression
- On-chip clocking (OCC) controllers

If no DFT features are enabled that require elaborated-RTL insertion, the command issues the following message:

```
fc_shell> insert_dft
Warning: There is no DFT IP specification to be reported (DFT-2003)
1
```

Pre-DFT DRC

Pre-DFT DRC (design rule checking) is the process of checking the suitability of the design for test operation before the scan structures are inserted.

In the Fusion Compiler tool, pre-DFT DRC is performed automatically during the initial optimization stage of the `compile_fusion` command. For example,

```
fc_shell> compile_fusion -to initial_opto
...
-----
Begin Pre-DFT violations...
Warning: Clock input ZC_reg[3]/CLK of DFF ZC_reg[3] was not controlled. (D1-1)
Information: There is 3 other cell(s) with the same violation. (DFT-3019)
Warning: Clock input ZD_reg[7]/CLK of DFF ZD_reg[7] cannot capture data. (D17-1)
Information: There is 3 other cell(s) with the same violation. (DFT-3019)
Warning: ZC_reg[7]/RSTB input of DFF ZC_reg[7] was not controlled. (D3-1)
Information: There is 3 other cell(s) with the same violation. (DFT-3019)
Warning: Clock input ZD_reg[7]/CLK of DFF ZD_reg[7] not active when clocks are
set on. (D9-1)
Information: There is 3 other cell(s) with the same violation. (DFT-3019)
Pre-DFT violations completed...
-----
Violation Report Summary
Total Violations: 17
-----
1 VECTOR VIOLATIONS
  1 missing state (V14)
16 PRE-DFT VIOLATIONS
  4 DFF clock line not controlled (D1)
  4 Clock_port not capable of capturing data (D17)
  4 DFF reset line not controlled (D3)
  4 Clock_port not active when clocks set to on (D9)

Warning: Violation(s) occurred during test design rule checking. (DFT-2031)
-----
-----
```


Sequential Cell Report:	Sequential Cells	Core Segments	Core Segment Cells
Sequential Elements Detected:	34	0	0
Clock Gating Cells :	1		
Synchronizing Cells :	0		
Non-Scan Elements :	0		
Excluded Scan Elements :	0	0	0
Violated Scan Elements :	12	0	0
(Traced) Scan Elements :	22 (58.8%)	0 (0.0%)	0 (0.0%)

```

-----
Sequential Cell Report
Total Sequential Cells: 34
-----
12 SEQUENTIAL CELLS WITH VIOLATIONS
    12 cells have test design rule violations
22 SEQUENTIAL CELLS WITHOUT VIOLATIONS
    20 cells are valid scan cells
    1 cell is clock gating cell
    1 cell is transparent latch
-----

Information: Test design rule checking completed. (DFT-3018)
Information: Test DRC reports 12 violating cells. (DFT-1086)
...

```

By default, the pre-DFT DRC report summarizes DRC violations by reporting the first instance of each type. To report every violation instance, enable the following option:

```
set_dft_drc_configuration \
    -pre_dft_drc_verbose_report_in_compile enable
```

To write the pre-DFT DRC report to a file instead of the output log, use the following option:

```
set_dft_drc_configuration \
    -pre_dft_drc_report_file_in_compile pre_dft_drc.rpt
```

Inserting Scan Structures

In the Fusion Compiler tool, the scan structures are inserted automatically during the initial optimization stage of the `compile_fusion` command.

The scan structures are inserted into the placed and mapped design to minimize the disturbance. For example,

- Scan chains are routed to be physically compact.
- Existing functional multivoltage cells are used where possible.
- Buffering for test-mode and scan-enable signals is able to utilize free space around functional logic cells.

Scan Structure Insertion in UPF Designs

When the Fusion Compiler tool inserts scan structures in a design that contains UPF information, it considers the multivoltage characteristics of the design when creating new nets and adding connections to existing nets.

In particular, it considers the following:

- The supply of the *source* (driver) port or leaf pin
- The supplies of the *sink* (load) ports or leaf pins
- The domain crossings between the source and sink(s)

When all sinks of a net have the same supply, the net has a *homogeneous* fanout. When the sinks have multiple supplies, the net has a *heterogeneous* fanout.

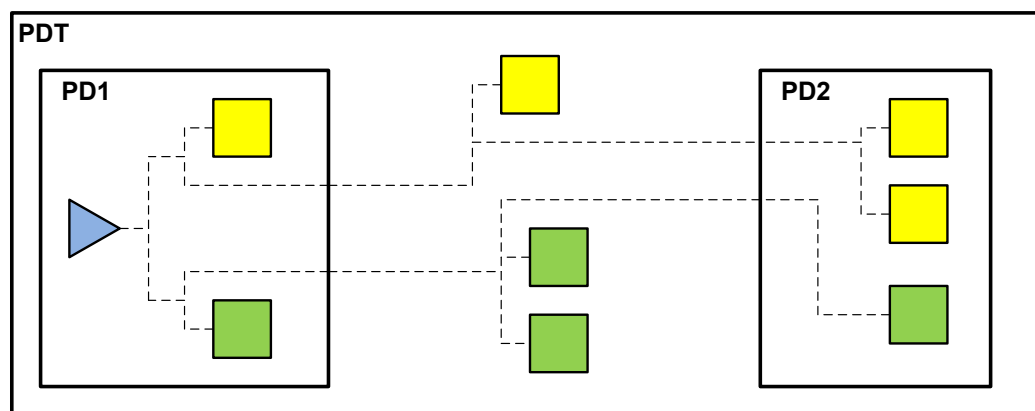
Note:

The tool considers sinks to have the same supply if their supplies are either connected or defined as equivalent with the `set_equivalent` command, even if their power domains are different.

Creating New Nets

When the tool creates a new net,

- Connections to sinks with the *same* supply are branched as close to the sinks as possible, which minimizes port-punching.
- Connections to sinks with *different* supplies are branched as close to the source as possible, which minimizes the potential for unfixable multivoltage violations to the sinks.



Connections to Existing Nets

When the tool adds connections to an existing net, it uses the same rules as described in [Creating New Nets](#). In addition, it applies the following rules:

- The analysis can cross existing isolation and level-shifter cells to find a connection point.
 - However, connections cannot be added on the net segment between an isolation or level-shifter cell and the power domain boundary its strategy was applied to.
- Connections to existing net segments with heterogeneous fanout are not allowed.

Dedicated Wrapper Cell Insertion

When the tool inserts dedicated wrapper cells in a design that contains UPF information, it prefers the highest possible hierarchical nets (closest to the ports) that do not result in multivoltage violations.

For details on the algorithm, see [Dedicated Wrapper Cell Insertion in UPF Designs](#).

Post-DFT DRC

Post-DFT DRC (design rule checking) is the process of checking the scan structures in a design for proper operation.

In the Fusion Compiler tool, post-DFT DRC must be explicitly run after compile by running the `dft_drc` command. For example,

```
fc_shell> dft_drc
Information: Running Post-DFT DRC for Internal_scan mode
-----
Begin scan design rules checking...
-----
Begin reading test protocol file /prj/dft/tmax/top_31512.3293_0.spf...
End parsing STIL file /prj/dft/tmax/top_31512.3293_0.spf with 0 errors.
Test protocol file reading completed, CPU time=0.00 sec.
-----
Begin simulating test protocol procedures...
Nonscan cell constant value results: #constant0 = 1, #constant1 = 0
Nonscan cell load value results      : #load0 = 1, #load1 = 0
Test protocol simulation completed, CPU time=0.00 sec.
-----
Begin scan chain operation checking...
Chain 1 successfully traced with 20 scan_cells.
Warning: Rule S19 (nonscan cell disturb) was violated 8 times.
Scan chain operation checking completed, CPU time=0.00 sec.
-----
Begin clock-gating analysis...
1 ATPG controllable clock-gating cells were found
Setting CTRL3(id=4) to 1 allows 20.00% of all scan cells to toggle.
Setting SE(id=39) to 1 allows 20.00% of all scan cells to toggle.
Clock-gating analysis completed, CPU time=0.00 sec.
-----
```

```

Begin clock rules checking...
Warning: Rule C2 (unstable nonscan DFF when clocks off) was violated 8 times.
Warning: Rule C16 (nonscan cell port unable to capture) was violated 4 times.
Warning: Rule C24 (nonclock PI connected to unstable cell clock input) was
violated 2 times.
Clock rules checking completed, CPU time=0.00 sec.
Clock grouping results: #pairs=0, #groups=0, #serial_pairs=0,
#disturbed_pairs=1, CPU time=0.00 sec.
-----
Begin nonscan rules checking...
Nonscan cell summary: #DFF=12 #DLAT=2 #RAM_outs=0 tla_usage_type=hot_clock_tla
Nonscan behavior: #CU=8 #C0=1 #TLA=1 #LE=4
Nonscan rules checking completed, CPU time=0.00 sec.
-----
Begin DRC dependent learning...
Fast-sequential depth results: control=1(226), observe=1(74), detect=1(226),
CPU time=0.00 sec
DRC dependent learning completed, CPU time=0.00 sec.
-----
DRC Summary Report
-----
Warning: Rule S19 (nonscan cell disturb) was violated 8 times.
Warning: Rule C2 (unstable nonscan DFF when clocks off) was violated 8 times.
Warning: Rule C16 (nonscan cell port unable to capture) was violated 4 times.
Warning: Rule C24 (nonclock PI connected to unstable cell clock input) was
violated 2 times.
There were 22 violations that occurred during DRC process.
Design rules checking was successful, total CPU time=0.00 sec.
-----
1

```

If you have multiple test modes, use the `-test_mode` option to specify which mode to analyze:

```

fc_shell> dft_drc -test_mode Internal_scan
...
fc_shell> dft_drc -test_mode ScanCompression_mode
...

```

You can use the `list_test_modes` command to list the test modes in the current design.

Customizing the DFT DRC Analysis

This topic describes how to customize the DFT DRC (design rule checking) analysis. It applies to both pre-DFT DRC (before scan insertion) and post-DFT DRC (after scan insertion).

Saving the DFT DRC Temporary Files

DFT DRC uses the TestMAX ATPG DRC engine to perform its checking. By default, the log files and other temporary files from this analysis are deleted. To preserve these files, use the `-log_dir` option of the `set_dft_drc_configuration` command.

For example,

```
file delete -force pre
file mkdir pre
set_dft_drc_configuration -log_dir pre
compile_fusion -to initial_opto

file delete -force post
file mkdir post
set_dft_drc_configuration -log_dir post
dft_drc
```

The directory contains the DRC log file, along with the Tcl script, netlist, and test protocol files used to generate it.

Providing DRC Simulation Models

In some cases, the synthesis libraries used by the tool might model a cell as a black box, while the simulation libraries used by the TestMAX ATPG analysis might provide functional information for the same cell.

To improve the accuracy of the analysis, you can provide the TestMAX ATPG simulation models for these black-box cells to DFT DRC:

```
fc_shell> set_dft_drc_configuration -libs {mem32x4.v mem16x8.v}
```

Note:

You should use this option only to replace leaf cells that do not have functional models. Do not use it to replace any arbitrary module in the design.