Tessent® BoundaryScan User's Manual

Software Version 2019.1

March 2019

Document Revision 12

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: mentor.com
Support Center: support.mentor.com

Send Feedback on Documentation: support.mentor.com/doc_feedback_form

Revision History

Revision	Changes	Status/ Date
12	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Tessent Release Notes* for this product are reflected in this document. Approved by Ron Press.	Released Mar 2019
11	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Tessent Release Notes* for this product are reflected in this document. Approved by Ron Press.	Released Dec 2018
10	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Tessent Release Notes* for this product are reflected in this document. Approved by Ron Press.	Released Aug 2018
9	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the *Tessent Release Notes* for this product are reflected in this document. Approved by Ron Press.	Released May 2018

Author: In-house procedures and working practices require multiple authors for documents. All associated authors for each topic within this document are tracked within the Mentor Graphics Technical Publication's source. For specific topic authors, contact Mentor Graphics Technical Publication department.

Revision History: Released documents maintain a revision history of up to four revisions. For earlier revision history, refer to earlier releases of documentation which are available at the following URL:

http://support.mentor.com

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction to Tessent BoundaryScan

Boundary scan, also referred to as JTAG, is a design for testability (DFT) technique that facilitates the testing of interconnect circuitry on printed circuit boards and in multi-chip modules (MCMs). The devices on those boards can also be tested, to a limited extent, with boundary scan techniques. Boundary scan greatly improves board-level testing, shortening the time devoted to manufacturing test and diagnostics.

Note

JTAG is the Joint Test Action Group, which is the committee that formulated the IEEE 1149.1 standard that describes boundary scan.

The most recent revision to the IEEE standard occurred in 2013 and is known as IEEE 1149.1-2013. Support for differential and capacitively coupled pin testing was introduced with the IEEE 1149.6 specification. Tessent BoundaryScan is fully compliant with the IEEE 1149.1-2001 and IEEE 1149.6 specifications, but does not support the extensions introduced in the IEEE 1149.1-2013 specification.

Benefits of Boundary Scan

In-circuit test is becoming less useful due to the increasing popularity of surface mount devices. Boundary scan provides the same benefits as in-circuit testing without requiring physical access to the electrical network on printed circuit boards. Adding boundary scan logic to your board enables you to detect the vast majority of board manufacturing process faults. These faults include wrong, missing, or incorrectly oriented components; components with stuck, shorted, or open pins; and failed wire bonds.

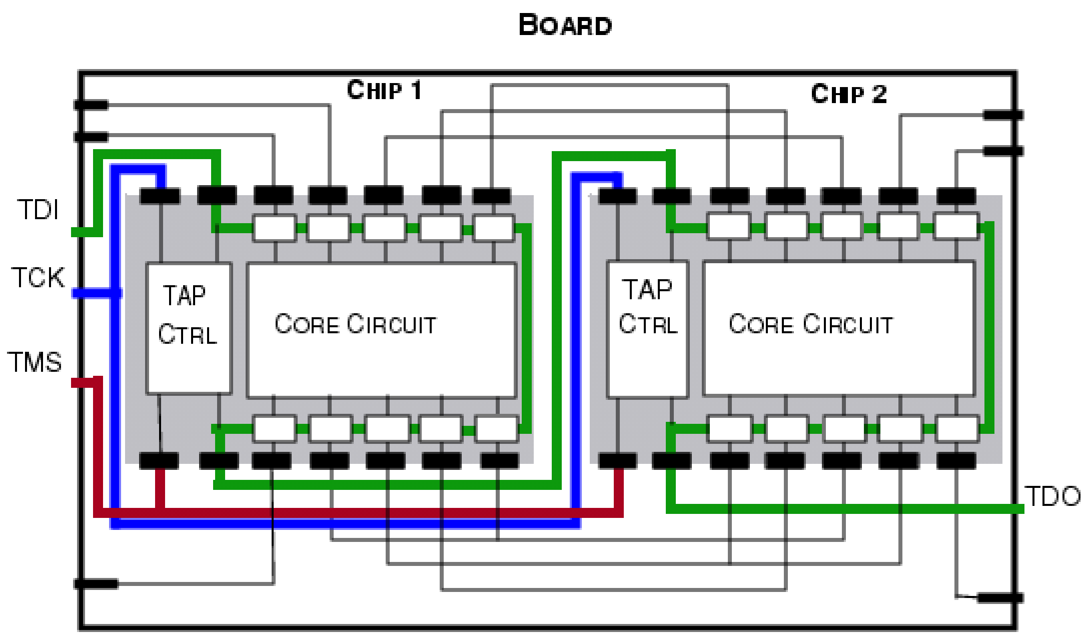
Although your engineering costs may increase slightly because of the additional silicon and ports used for the boundary scan circuitry, implementing the IEEE 1149.1 standard can dramatically reduce design manufacturing costs.

Boundary Scan Overview

The primary use for boundary scan circuitry is in board-level testing, but it can also control circuit-level test structures such as BIST or internal scan. Create a standard interface for accessing and testing chips at the board level by adding boundary scan circuitry to your design.

Figure 1-1 shows a board containing two chips with boundary scan circuitry.

Figure 1-1. Boundary Scan Chips on a Board



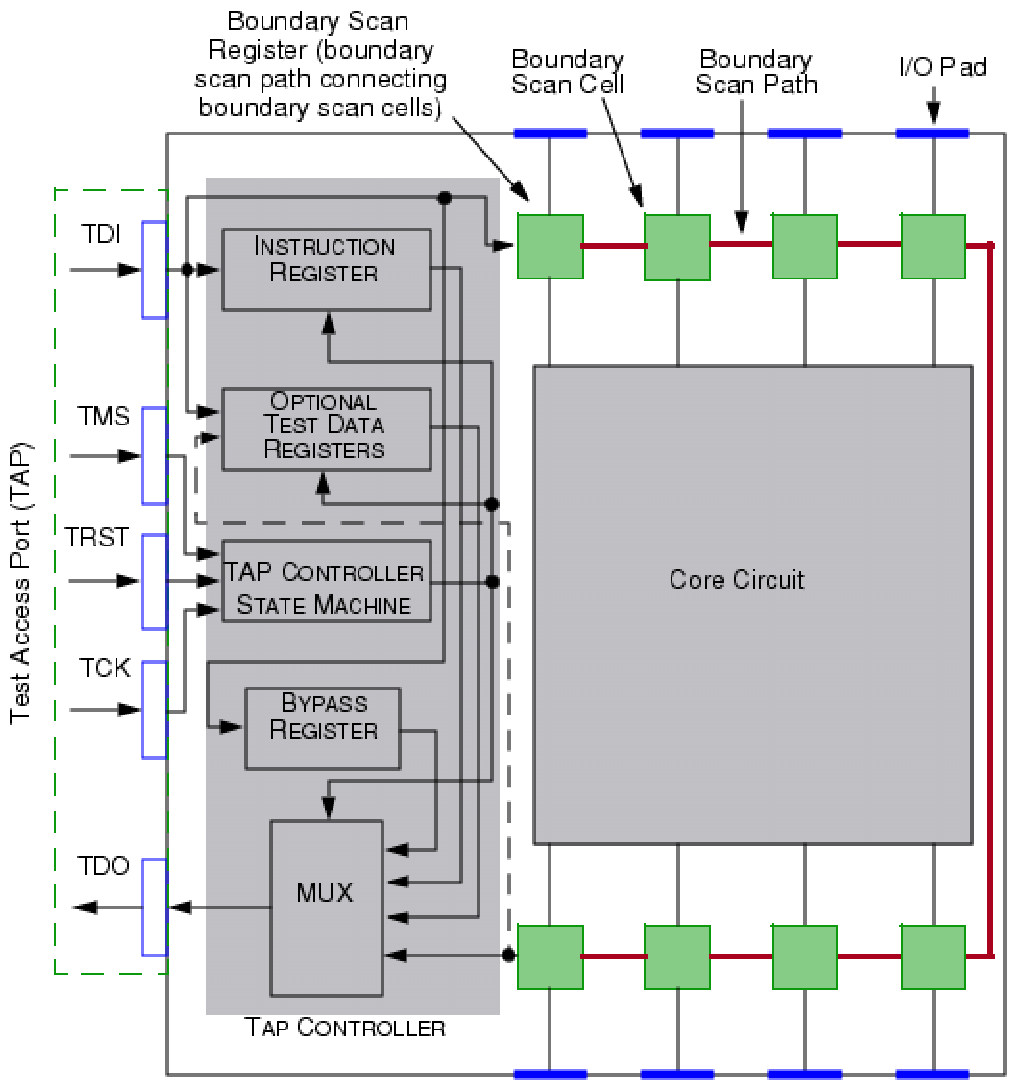
When used on a board, boundary scan provides access to the input and output ports of the chips by linking them together into a long scan path. Data shifts along the scan path, starting at the board's test data in (TDI) port and ending at the board's test data out (TDO) port. In between, the scan path connects all the devices on the board that contain boundary scan circuitry. The TDO of one chip feeds the TDI of the next, all the way around the board.

The test clock (TCK) and test mode select (TMS) inputs connect globally to each boundary scan device in the scan path of the board. With this configuration you can test board interconnections, perform a snapshot of normal system data, or test individual chips. The test access port (TAP) controller is a state machine that controls the operation of the boundary scan circuitry.

Boundary Scan Architecture

This figure shows the general architecture of a chip after the addition of boundary scan logic.

Figure 1-2. Boundary Scan Architecture



This simple boundary scan architecture contains the following components:

- **Core circuit** — This is the application logic of the original design before boundary scan logic is added. This logic might already contain internal scan circuitry (or, at least, internal scan ports so the boundary scan path can be built).
- **Boundary scan cells** — These contain memory elements for capturing data from the circuit, loading data into the circuit, or serially shifting data to the next scan cell in the

path. The tool places boundary scan cells between the core logic and each input, bidirectional, or two or three-state output pin. Boundary scan cells collectively comprise a parallel-in, parallel-out shift register that runs along the periphery, or boundary, of the original design.

- **Test Access Port (TAP)** — This is a set of signals that control the boundary scan operation. The TAP consists of a minimum of four pins for the four signals of the test bus. These signals include the test clock (TCK), the test data input (TDI), the test data output (TDO), and the test mode selector (TMS). Also shown is an optional, active low, asynchronous test reset (TRST).

- **TAP controller state machine** — This is a finite state machine that controls the operation of the instruction and test data registers. The TAP controller's state depends on the value of the TMS line at each clock pulse and the controller's current state.

- **Boundary scan register** — This is the main test data register. The boundary scan register is a virtual shift register consisting of the individual boundary scan cells that can, either serially or in parallel, load and unload input and output data for the circuit.

- **Bypass register** — This is a register that shortens the serial path between TDI and TDO to one cell when there is no requirement to test a particular device. This shortened path in effect bypasses the chip, allowing more efficient test data shifting to other devices in the chain.

- **Optional Test Data registers**:

    - **Device Identification register** — This register contains a device identification code or programming code used to check that the board is populated with the correct chips.

    - **Data-Specific registers** — These registers allow access to the chip's test support features, such as BIST and internal scan paths.

- **Instruction Register** — This register controls the boundary scan circuitry by connecting a specific test data register between the TDI and TDO pins. It controls the operation affecting the data in that register, using a predefined set of instructions. Some instructions are mandatory, and others are optional.

    Mandatory instructions:

    - **EXTEST** — This instruction tests circuitry external to the devices themselves, such as board interconnect. EXTEST is the main test instruction for boundary scan testing.

    - **SAMPLE/PRELOAD** — This instruction takes data from the chips's I/O pads and latches it into the boundary scan register (during normal board operation).

        Note

        The IEEE 1149.1-2001 Standard has redefined this instruction as two separate instructions (SAMPLE and PRELOAD).

- **SAMPLE** — This instruction takes data from the chip's I/O pads and latches it into the boundary scan register (during normal board operation).
- **PRELOAD** — This instruction loads test data into the boundary scan register before selecting another instruction.
- **BYPASS** — This instruction enables bypassing of chips not being tested. For example, if you only want to test one chip in a board with 20 chips in the boundary scan chain, all bypassed chips will contribute a single scan flop to the scan chain and the chip under test would contribute all of its boundary scan registers to the scan chain. Consequently, shifted data needs to go through only one extra shift to pass through each of the chips not being tested (as opposed to the entire boundary scan register).
- **EXTEST_PULSE** — This instruction tests circuitry external to the chips themselves, for designs that have AC pins. This instruction is mandatory only when using AC cell types.

Optional instructions:

- **INTEST** — This instruction tests a chip's internal circuitry by applying a test vector to, and capturing the output response from, the application logic.
- **IDCODE** — This instruction connects the device identification register between TDI and TDO. The device identification register contains the device ID code, which is typically used to determine if the chip belongs on the board.
- **USERCODE** — This instruction also connects the identification register between TDI and TDO, but the information placed in that register is user-defined and is meant to expand on the IDCODE information.
- **CLAMP** — This instruction forces static 1s or 0s on selected nodes in order to block interfering signals or create a testable situation.
- **HIGHZ** — This instruction forces a chip's output and bidirectional pins into a high-impedance state. In this condition, an in-circuit tester can test it without risking overdrive damage.
- **RUNBIST** — This instruction executes the circuit's internal BIST procedure.
- **EXTEST_TRAIN** — This instruction, which is available when using AC cell types, operates similarly to EXTEST_PULSE but can use multiple TCK cycles. For more details, refer to the IEEE 1149.6 Standard.

Embedded Boundary Scan

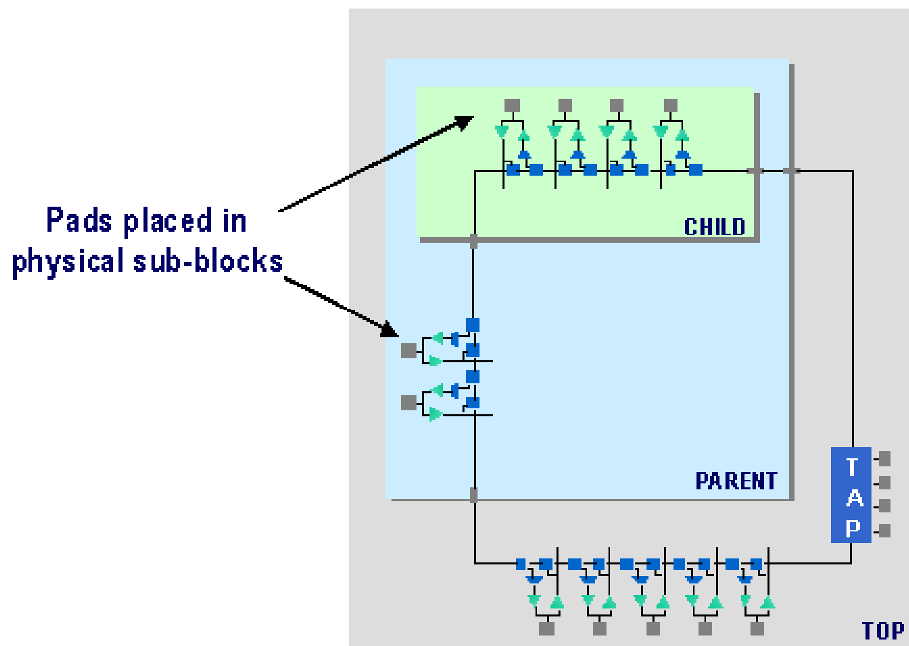
Increasingly, chip I/O cells are being placed directly into cores to be closer to the logic they service. This approach typically results in significant physical design benefits, including reduced signal routing and improved timing. The Mentor embedded boundary scan feature allows boundary scan cells to be integrated alongside their associated I/O cells within the core rather than at the top level of the chip.

Boundary scan cells can be added to any core at any level within a design. A three -level example is shown in Figure 1-3.

The embedded boundary scan capability of Tessent BoundaryScan automates both the integration of the boundary scan cells and verification of the resulting boundary scan segment within the core.

This approach not only maintains and extends the physical design benefits of placing I/O cells directly into cores, but also enables a more efficient core reuse methodology, because all design and DFT sign-off activity can take place fully at the core level.

Figure 1-3. Embedded Boundary Scan Implementation

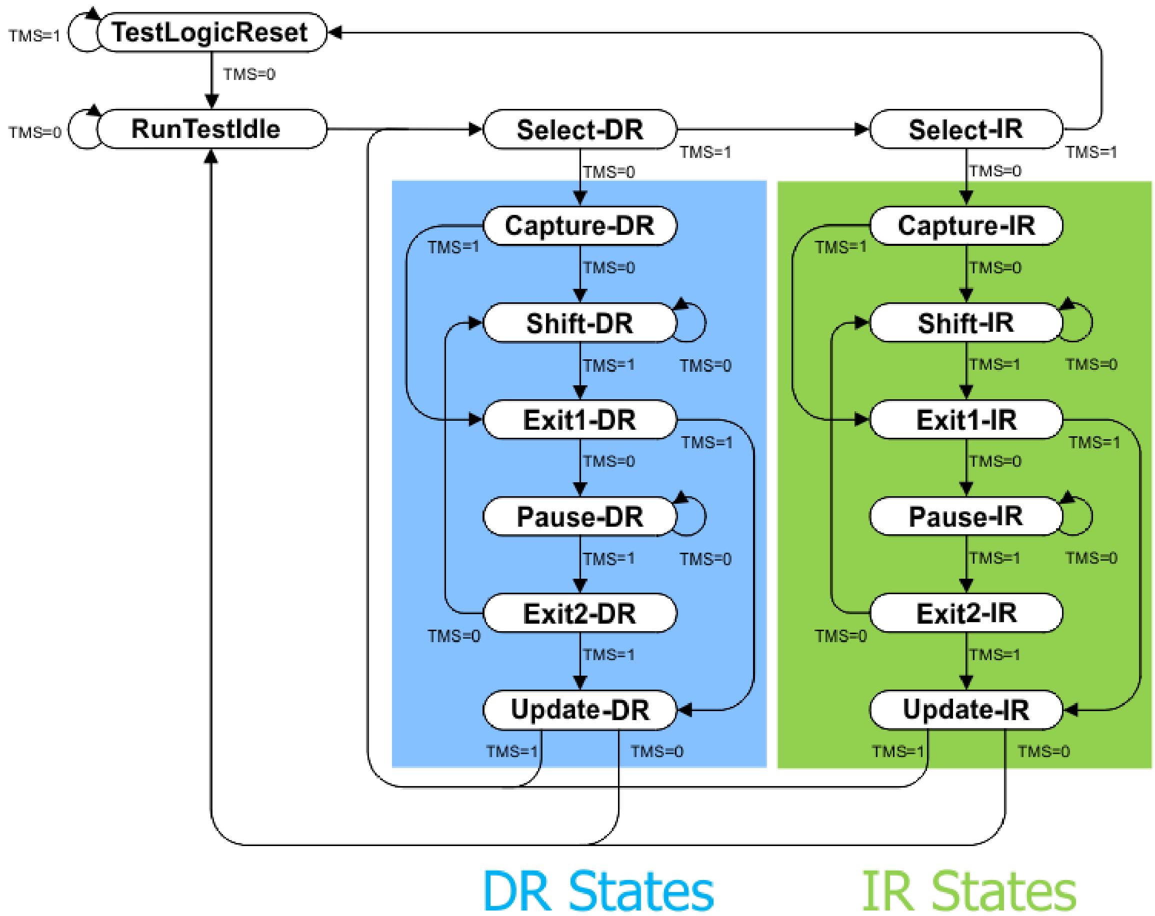


TAP Controller State Machine

The TAP controller is a synchronous finite state machine that controls the operation of the instruction and test data registers. The TAP controller's state depends on the value of the TMS line at the rising edge of each TCK clock pulse and the controller's current state.

Figure 1-4 shows the finite state machine diagram for the TAP controller of an IEEE 1149.1 circuit.

Figure 1-4. TAP Controller Finite State Machine Diagram



The TMS signal (the value shown adjacent to each state transition) controls the state transitions on each rising edge of TCK. The rising edge of the TCK clock also captures the TAP controller inputs.

The state encoding is as follows.

Table 1-1. State encoding and TAP states

Encode Value	Corresponding TAP State
1111	Test-Logic-Reset
1100	Run-Test/Idle
0111	Select-DR-Scan
0110	Capture-DR
0010	Shift-DR
0001	Exit1-DR
0011	Pause-DR

Table 1-1. State encoding and TAP states (cont.)

0000	Exit2-DR
0101	Update-DR
0100	Select-IR-Scan
1110	Capture-IR
1010	Shift-IR
1001	Exit1-IR
1011	Pause-IR
1000	Exit2-IR
1101	Update-IR

These encodings are indicated on the state[3:0] output bus on the TAP.

Boundary Scan Insertion With Tessent BoundaryScan

Tessent BoundaryScan is the Mentor Graphics boundary scan insertion tool. The tool creates and connects RTL-level boundary scan logic compliant with the IEEE 1149.1-2001 Standard.

Features of the Tessent BoundaryScan tool:

- **Instruction Support —** Full support of required IEEE 1149.1 Standard instructions.
- **Extension Support —** Support of base extensions to IEEE 1149.1, such as the Device ID register.
- **Compliant Verilog —** Generation of Verilog (IEEE 1364-2001) that is compliant with the ModelSim (Verilog), Synopsys' Design Compiler, and other industry synthesis tools.
- **RTL-Level Boundary Scan Generation —** Insertion and connection of boundary scan circuitry at the RTL level, moving the generation of test circuitry earlier in the design process.
- **Customized Boundary Scan —** Generation of default or user-customized boundary scan architectures.
- **I/O pad Synthesis —** Generation of generic I/O pads or technology-specific I/O pads.
- **Automatic Connection —** Automatic connection of boundary scan to internal scan logic.

- **Test bench Generation —** Generation of a test bench, which allows testing of the boundary scan logic after interconnection with the core application logic.
- **Test Vector Generation —** Generation of boundary scan test vectors in a variety of common test data formats, as well as ASCII, binary, STIL, WGL, and others.
- **Setup File Generation —** Generation of ATPG setup files, for designs with generated boundary scan circuitry controlling internal scan circuitry.
- **Compliant BSDL —** Production of BSDL output that is compliant with the IEEE 1149.1-2001 standard.
- **Generic Element Mapping —** Mapping of boundary scan elements to generic boundary scan library cells (which enhances re-targetability of the boundary scan circuitry).
- **Technology-Specific Element Mapping —** Mapping of boundary scan, I/O pad, and TAP controller elements to technology-specific library cells. Technology mapping provides support for replacing generic boundary scan cells, I/O pads, and the TAP controller by equivalent technology-specific cells.

Chapter 2
Getting Started With Tessent BoundaryScan

This chapter describes how to start inserting Tessent BoundaryScan within Tessent Shell and includes examples showing the most common scenarios and usages.

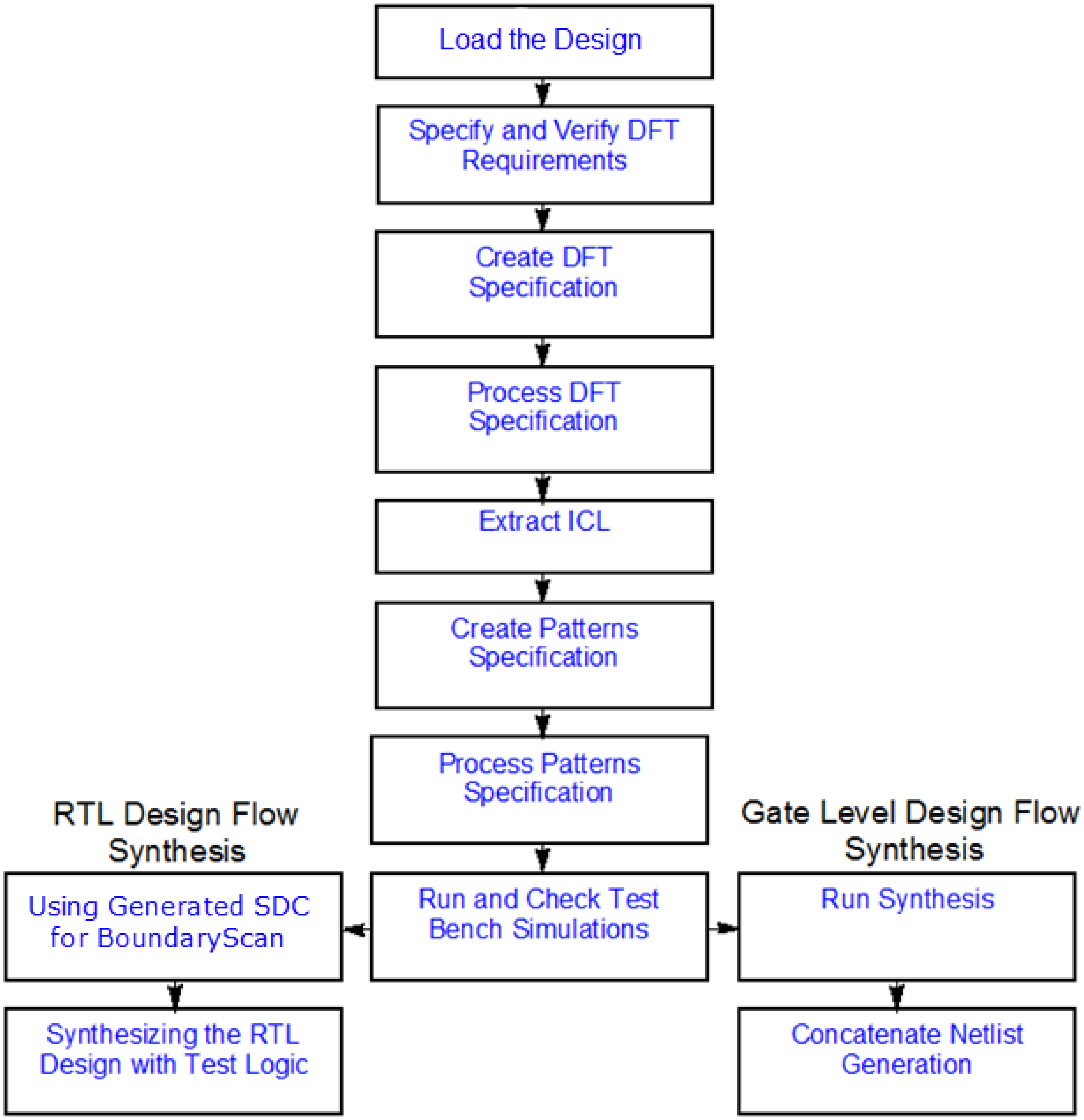
For a complete set of wrapper and property descriptions, refer to the "BoundaryScan" section of the "DftSpecification Configuration Syntax," "PatternsSpecification Configuration Syntax," and "DefaultsSpecification Configuration Syntax" sections in the *Tessent Shell Reference Manual*. The flow and steps are the same for inserting TAP, boundary scan, or embedded boundary scan. Each step describes any commands that are required for embedded boundary scan.

DFT Flow Using Tessent Shell

Tessent BoundaryScan in Tessent Shell has a basic, high-level flow sequence.

The figure below illustrates the high-level sequence of steps required to insert Tessent BoundaryScan into a design. Each step in the figure links to more detailed information about the design-for-test (DFT) flow, including examples.

Figure 2-1. Design Flow for Tessent BoundaryScan



Design Flow Prerequisites

To use the DFT design flow, you must have either an RTL or a gate-level netlist with IO pads already inserted into the design. For an RTL netlist, you must have the Tessent cell library or the pad library for the pad cells. For a gate-level netlist, you must have the Tessent cell library or the ATPG library for the standard cells, in addition to the Tessent cell library for the IO pad cells. The pad library supported by Tessent BoundaryScan-LV is natively supported in Tessent Shell if the Tessent cell library for the IO pad cells is not present.

Design Flow Dofile Example

The example dofile in this section shows you how to set up a typical design flow.

The following example dofile follows the design flow described in Figure 2-1.

Load the Design

```
set_context dft -rtl
read_cell_library ../library/adk_complete.tcelllib
read_verilog ../netlist/cpu_top.v
set_current_design cpu_top
```

Specify and Verify DFT Requirements

```
set_dft_specification_requirements -boundary_scan on
set_design_level chip
set_attribute_value tck_p -name function -value tck
set_attribute_value tdi_p -name function -value tdi
set_attribute_value tms_p -name function -value tms
set_attribute_value trst_p -name function -value trst
set_attribute_value tdo_p -name function -value tdo
set_boundary_scan_port_options ramclk_p -cell_options clock
set_boundary_scan_port_options reset_p -cell_options sample
check_design_rules
```

Create DFT Specification

```
set spec [create_dft_specification]
report_config_data $spec
```

Process DFT Specification

```
process_dft_specification
```

Extract ICL

```
extract_icl
```

Create Patterns Specification

```
create_patterns_specification
```

Process Patterns Specification

```
process_patterns_specification
```

Run and Check Test Bench Simulations

```
set_simulation_library_sources -y ../library/verilog \
   -v ../library/pad_cells.v
run_testbench_simulations
check_testbench_simulations -report_status
```

Test Logic Synthesis

```
run_synthesis
```

Related Topics

Load the Design

Specify and Verify DFT Requirements

Create DFT Specification

Process DFT Specification

Extract ICL

Create Patterns Specification

Process Patterns Specification

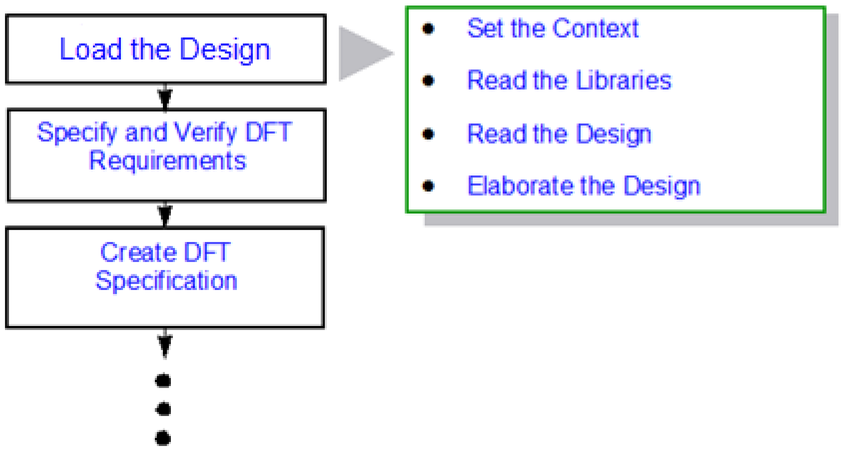
Run and Check Test Bench Simulations

Test Logic Synthesis

Load the Design

Loading the design is the first step in Tessent BoundaryScan insertion using Tessent Shell. The process consists of setting the correct context, reading libraries, reading the design, and elaborating the design.

Figure 2-2. Design Loading



Set the Context

In Tessent Shell, setting the context means two things. First and foremost, you must set the context to dft for boundary scan hardware to be created. Second, you must specify whether the design type to be read in is written in RTL. If so, you must specify the -rtl option. If the design to be read in is a gate-level Verilog netlist, you should specify the -no_rtl option.

When using the -no_rtl mode, a concatenated netlist is written out at the end of the dft insertion phase. In rtl mode, the file structure of the input design is preserved and only the modified design files are written out at the end of the dft insertion phase along with the newly created test IP. The netlist to be read in can be Verilog, VHDL, or mixed language.

If Tessent tools inserted Embedded Boundary Scan, which is now being integrated at the next level, you must open the Tessent Shell Data Base (TSDB) of the child (the embedded boundary scan's sub_block or physical_block) using the open_tsdb command. If you are using the same TSDB for both child and parent, you can reuse the TSDB (the default is tsdb_outdir), and you do not need to explicitly open the default TSDB because the existing content of the TSDB output directory is automatically visible to the tool. See the set_tsdb_output_directory command description for how to control the name and location of the TSDB output directory.

Examples

Example 1

The following example sets the context to dft and specifies that the design to be read in is written in RTL.

```
set_context dft -rtl
```

Example 2

The following example sets the context to dft and specifies that a gate-level netlist will be read in.

```
set_context dft -no_rtl
```

Example 3

The following example opens a child’s TSDB directory and therefore, exposes it at the parent level.

```
open_tsdb ../ebscan_tsdb_outdir
```

Read the Libraries

You can use the read_cell_library command to read in the library file for the pad IO macros that are instantiated in the design. If you are inserting Tessent BoundaryScan into an RTL netlist or design, reading the library for the pad IO macros is sufficient. If the Tessent cell libraries do not include the pad information, the legacy LV pad library format is natively supported by the read_cell_library command and can be used to augment the Tessent cell libraries with the pad information.

Examples

Example 1

The following example reads in the Tessent cell library file for the pad IO macros.

```
read_cell_library ../library/adk_complete.tcelllib
```

Example 2

The following example reads in the ATPG.lib files and the old pad library description when the Tessent cell libraries do not include the pad information.

```
read_cell_library ../library/atpg.lib
read_cell_library ../library/pad.library
```

Read the Design

In Tessent Shell, after setting the context and loading the required libraries, you can use the read_verilog command to read in the design.

Examples

Example 1

The following example reads in one netlist, which can be either RTL or gate level.

```
read_verilog ../netlist/cpu_top.v
```

Example 2

The following example reads in a Verilog file and directory of design files.

```
set_design_sources -format verilog -V ../design/top.v \
  -Y ../design -extensions {v gv}
```

Elaborate the Design

The next step in loading a design is to elaborate the design using the set_current_design command. The set_current_design command specifies the root of the design. If any module descriptions are missing, design elaboration will identify them. For Tessent BoundaryScan insertion, modules such as memory instances, PLL instantiations are not needed. You can specify the modules with the add_black_box -module command.

Example

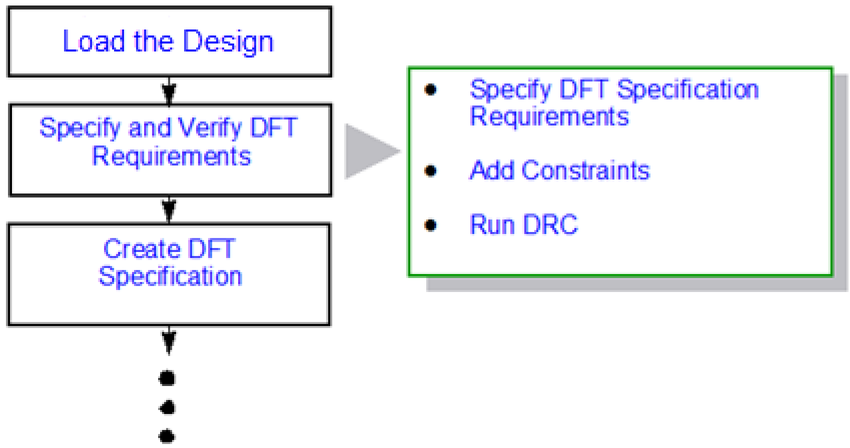
The following example shows how to use the set_current_design command.

```
set_current_design cpu_top
```

Specify and Verify DFT Requirements

The next step to insert Tessent BoundaryScan in Tessent Shell is to specify the DFT requirements, add constraints, and verify whether the DFT requirements specified are correct by running DRC (Design Rule Checking).

Figure 2-3. Specify and Verify DFT Requirements



Specify DFT Specification Requirements

To insert Tessent BoundaryScan, you must specify the DFT specification requirements with the set_dft_specification_requirements command. The DFT specification provides a template for logic and hardware you will generate and insert into the design.

You specify chip with the set_design_level command when you are inserting boundary scan at the chip level. If you are using embedded boundary scan, you must specify either sub_block or physical_block. Also, you must use the set_boundary_scan_port_options command to specify the list of pad IO ports that need boundary scan, as shown in Example 2.

Examples

The following examples show how to specify DFT specification requirements for chip and sub_block levels.

Example 1

The following example shows how the boundary scan DFT specification requirements are specified and how the design is specified at the chip level.

```
set_dft_specification_requirements -boundary_scan On
set_design_level chip
```

Example 2

The following example shows, for embedded boundary scan, how to provide a Tcl list of pad IO ports that need boundary scan cells to be inserted. The example also shows how to set the design level to sub-block.

```
set_dft_specification_requirements -boundary_scan on

set_boundary_scan_port_options -pad_io_ports [list in1 in_diff_p in_\
  diff_n out1 out_diff_p out_diff_n clk A Y]

set_design_level sub_block
```

Note

To ensure the pad IO macros function properly when using embedded boundary scan, you may need to use the add_input_constraints command to constrain some ports to either a 1 or 0 value. Typically, these ports are driven to the proper values at the next higher level in the design.

Add Constraints

To insert Tessent BoundaryScan, four TAP pins (TDI, TCK, TMS, and TDO) must be available at the chip level and connected to pad IO macros. TRST, which is optional, can be an output pin of a power-up detector.

When all five TAP pins are connected to chip level pad IO macros, a 5-pin TAP will be inserted. If TRST is connected to an internal power-on reset pin, a 4-pin TAP will be inserted.

Note

If you need to specify an internal pin for TRST, and that pin connects to a chip level pad IO macro, two insertion passes are needed. The TAP needs to be inserted in the first pass, and the boundary scan insertion is performed in a subsequent pass.

The TAP pins can be identified in the constraints section of your dofile or can be specified in the pin order file. Similarly, the power and ground pins can be identified in the constraints section or specified in the pin order file. If any special boundary-scan cell types are required, they can also be specified in the constraints section using the set_boundary_scan_port_options command.

If you are using embedded boundary scan and some ports must be constrained to a constant 1 or 0 value, use the add_input_constraints command. Typically, these values are properly driven at the next higher level in the design.

Examples

Example 1

The following example specifies the function and purpose of the five TAP pins (tck_p, tdi_p, tms_p, trst_p, tdo_p).

```
set_attribute_value tck_p -name function -value tck
set_attribute_value tdi_p -name function -value tdi
set_attribute_value tms_p -name function -value tms
set_attribute_value trst_p -name function -value trst
set_attribute_value tdo_p -name function -value tdo
```

The following specifies the power and ground pins.

```
set_attribute_value vdd* -name function -value power
set_attribute_value vss* -name function -value ground
```

The following specifies special boundary scan cell types.

```
set_boundary_scan_port_options ramclk_p -cell_options clock
set_boundary_scan_port_options reset_p -cell_options dont_touch
```

Example 2

The following example shows how to use the DefaultsSpecification wrapper to specify the five TAP pins if they are the standard port names used across all designs. The DefaultsSpecification wrapper is used when the create_dft_specification command is issued in the next step.

```
DefaultsSpecification(group) { //Legal: company | group | user
  DftSpecification {
    IjtagNetwork {
      HostScanInterface {
        Chip {
          tck : tck_p;
          tdi : tdi_p;
          tdo : tdo_p;
          tms : tms_p;
          trst : trst_p;
        }
      }
    }
  }
}
```

You also can use the set_defaults_value command to set the DefaultsSpecification and the get_defaults_value command to see the specified value.

Example 3

The following example reads in the pin order file where the five TAP pins, power, and ground are specified.

```
set_boundary_scan_port_options -pin_order_file cpu_top.pinorder.my
```

Example 4

The following example shows how you can set some ports to a constant value if you are using embedded boundary scan.

```
add_input_constraints drive_strength[1] -C0
add_input_constraints drive_strength[2] -C0
add_input_constraints edriver1 -C1
add_input_constraints edriver2 -C1
```

Run DRC

The next step in Specify and Verify DFT Requirements is to run Design Rule Checking (DRC) to make sure all the constraints are correct. Once DRC is clean, Tessent Shell moves from the SETUP to the ANALYSIS prompt.

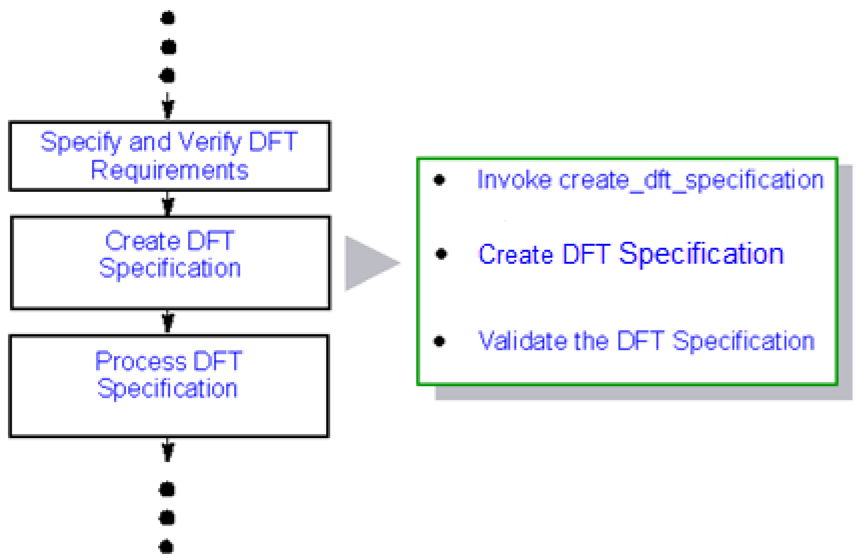
```
check_design_rules
```

Create DFT Specification

The next step in the design flow is to create a DFT specification.

The create_dft_specification command is used to create a default DFT specification based on the DFT requirements specified in the previous "Specify and Verify DFT Requirements" step. You can use the report_config_data command to report this default DFT specification. There are several methods available to edit or configure the DFT specification to meet your specific requirements.

Figure 2-4. Create DFT Specification



Invoke create_dft_specification

A DFT specification is automatically created using the create_dft_specification command. This DFT specification is stored in memory.

To report the DFT specification in memory, use the report_config_data command. The DFT specification uses IJTAG network infrastructure because this is the only supported method for incremental insertion passes. The IJTAG network is fully compliant with the 1149.1 IEEE standard. For further information about the Tessent IJTAG flow, refer to the *Tessent IJTAG User's Manual*.

To insert boundary scan into a preexisting TAP in your design, you must have an ICL for the TAP. The tool automatically uses this TAP to connect to the boundary scan chain if the TAP has ScanInterface host_bscan in the TAP ICL. You can, however, also specify which host_bscan to connect to by using the create_dft_specification -existing_host_bscan_scan_in command. The ICL for the preexisting TAP is read in automatically if it is in a location where the module description of the TAP is present. You also can use the read_icl command to read the ICL for a preexisting TAP. For requirements when using a preexisting TAP with boundary scan, refer to the *"Requirements on a TAP to be usable for BoundaryScan"* section in the *Tessent Shell Reference Manual*.

Examples

Example 1

In the following example, the DFT specification generated with the create_dft_specification is stored in a variable called dft_spec so that the variable can be used to report the DFT specification.

```
set dft_spec [create_dft_specification]
report_config_data $dft_spec
```

Example 2

The following example shows how to connect to the preexisting TAP when multiple TAPs are present. This only works if a ScanInterface host_bscan is in the ICL file for the preexisting TAP controller. If a single TAP controller is present, the tool automatically uses this controller, and you do not need to provide the host_bscan to which to connect.

```
create_dft_specification -existing_host_bscan_scan_in \
  My_TAP_INST/fromBscan
report_config_data
```

My_TAP_INST in this example is the instance name of the preexisting TAP in the design, and fromBscan is the input port on the preexisting TAP where the boundary-scan chain needs to connect.

Configure the DFT Specification

There are two ways to configure the default DFT specification according to your requirements: using the DFTVisualizer GUI or modifying the specification in memory. You do not need to edit the DFT specification if you want to use the default configuration.

Configure the DFT Specification With DFTVisualizer

One way to configure the DFT specification is to use the DFTVisualizer GUI. The edits you apply with DFTVisualizer update the DFT specification in memory.

Prerequisites

- You have created a DFT specification as described in "Create DFT Specification" on page 35.

Procedure

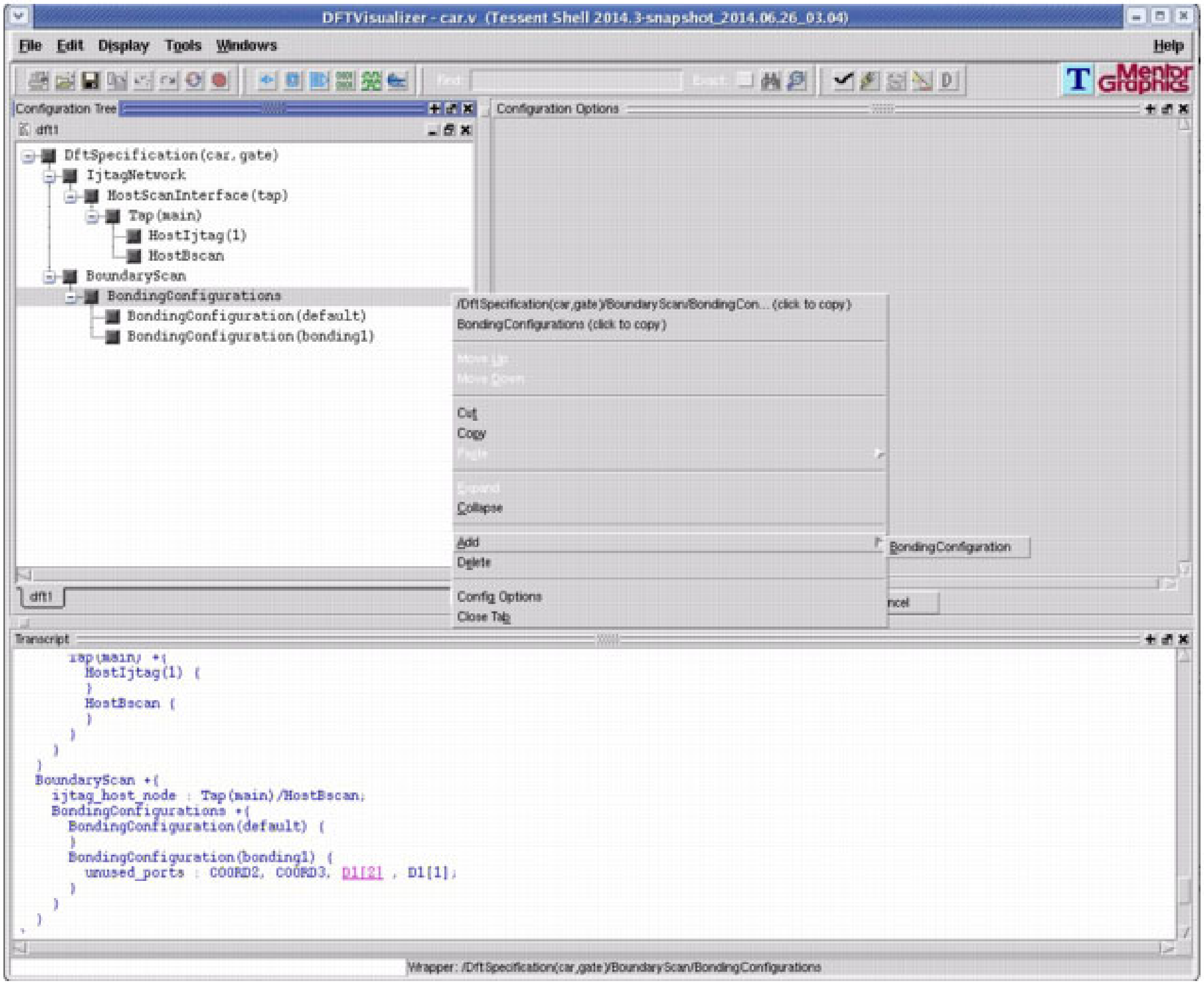
1. Open DFTVisualizer with the display_specification command.

   DFTVisualizer displays the DFT specification based on the DFT requirements specified in "Specify and Verify DFT Requirements" on page 31.

2. In the Configuration Tree pane, right-click the BoundaryScan wrapper and choose **Add > BondingConfigurations**.

3. Right-click the BondingConfigurations wrapper and choose **Add > BondingConfiguration**.

Figure 2-5. Editing the DFT Specification With DFTVisualizer



4. Click the BondingConfiguration wrapper to show the Configuration Options pane
5. Add the port to be excluded by typing in the port name manually in the unused_ports field.

   To exclude another port, click the + button and type in another port name. Repeat as necessary.
6. Click **Apply** to update the DFTVisualizer GUI and the DFT specification in memory.
7. Repeat these steps to add additional modifications.

Results

This method demonstrates using DFTVisualizer to edit the DFT specification through adding multiple package bonding configurations, which are defined by two BondingConfiguration wrappers.

To see the resulting configuration, use the report_config_data command.

```
> report_config_data $dft_spec

DftSpecification(car,gate) +{
  IjtagNetwork +{
    HostScanInterface(tap) +{
      Interface {
        tck : TCK;
        trst : TRST;
        tms : TMS;
        tdi : TDI;
        tdo : TDO;
      }
      Tap(main) +{
        HostIjtag(1) {
        }
        HostBscan {
        }
      }
    }
  }
  BoundaryScan {
    ijtag_host_interface : Tap(main)/HostBscan;
    BondingConfigurations {
      BondingConfiguration(default) {
      }
      BondingConfiguration(bonding1) {
        unused_ports : COORD2, COORD3, D1[2], D1[1], EN0, D1[0], D1[3];
      }
    }
  }
}
```

Examples

Using read_config_data, you can effectively cut and paste the manually entered DFTVisualizer edits into the dofile as shown in the example below. Then for subsequent runs, the configuration edits will already be present in the dofile, making the process repeatable via scripts.

```
read_config_data -in $dft_spec/BoundaryScan -from_string {
  BondingConfigurations +{
    BondingConfiguration(default) {
    }
    BondingConfiguration(bonding1) {
      unused_ports : COORD2, COORD3, D1[2], D1[1], EN0, D1[0], D1[3];
    }
  }
}
```

Related Topics

Configure the DFT Specification in Memory

Configure the DFT Specification in Memory

You can configure the DFT specification with the commands add_config_element and set_config_value. With this method, the dofile will already contain the commands and the modifications introduced, making the process repeatable.

Examples

The following examples show how to modify the DFT specification that is loaded in memory using the editing commands.

Example 1

The following example adds multiple package bonding configurations for Tessent BoundaryScan. The report_config_data command shows the DFT specification created.

```
> set spec [create_dft_specification]
> report_config_data $spec

DftSpecification(car,gate) {
  IjtagNetwork {
    HostScanInterface(tap) {
      Interface {
        tck : TCK;
        trst : TRST;
        tms : TMS;
        tdi : TDI;
        tdo : TDO;
      }
      Tap(main) {
        HostIjtag(1) {
        }
        HostBscan {
        }
      }
    }
  }
  BoundaryScan {
    ijtag_host_interface : Tap(main)/HostBscan;
  }
}
```

You can add the necessary BondingConfiguration wrappers and then use the report_config_data command to see how the DFT specification was updated.

```
> add_config_element $spec/BoundaryScan/BondingConfigurations
> add_config_element $spec/BoundaryScan/BondingConfigurations/\
    BondingConfiguration(default)

> add_config_element $spec/BoundaryScan/BondingConfigurations/\
    BondingConfiguration(bonding1)

> set_config_value $spec/BoundaryScan/BondingConfigurations/\
    BondingConfiguration(bonding1)/\
    unused_ports {COORD2 COORD3 D1[2] D1[1] EN0 D1[0] D1[3]}

> report_config_data

DftSpecification(car,gate) +{
  IjtagNetwork +{
    HostScanInterface(tap) +{
      Interface {
        tck : TCK;
        trst : TRST;
        tms : TMS;
        tdi : TDI;
        tdo : TDO;
      }
      Tap(main) +{
        HostIjtag(1) {
        }
        HostBscan {
        }
      }
    }
  }
  BoundaryScan {
    ijtag_host_interface : Tap(main)/HostBscan;
    BondingConfigurations {
      BondingConfiguration(default) {
      }
      BondingConfiguration(bonding1) {
        unused_ports : COORD2, COORD3, D1[2], D1[1], EN0, D1[0], D1[3];
      }
    }
  }
}
```

Example 2

The following example modifies the DFT specification using the editing commands, including the use of the delete_config_element command.

```
> set spec [create_dft_specification]
> report_config_data $spec

DftSpecification(cpu_top,rtl) {
  IjtagNetwork {
    HostScanInterface(tap) {
      Interface {
        tck : tck_p;
        trst : trst_p;
        tms : tms_p;
        tdi : tdi_p;
        tdo : tdo_p;
      }
      Tap(main) {
        HostIjtag(1) {
        }
        HostBscan {
        }
      }
    }
  }
  BoundaryScan {
    ijtag_host_interface : Tap(main)/HostBscan;
  }
}
```

You can add the necessary BondingConfiguration wrappers and then use the report_config_data to see how the DFT Specification was updated.

```
> add_config_element $spec/BoundaryScan/BondingConfigurations
> add_config_element $spec/BoundaryScan/BondingConfigurations/\
    BondingConfiguration(standard)

> add_config_element $spec/BoundaryScan/BondingConfigurations/\
    BondingConfiguration(package1)

> set_config_value $spec/BoundaryScan/BondingConfigurations/\
    BondingConfiguration(package1)/enable_signal cpu_top/cs

> set_config_value $spec/BoundaryScan/BondingConfigurations/\
    BondingConfiguration(package1)/unused_ports {D2, D3}

> delete_config_element enable_signal -in_wrapper $spec/BoundaryScan/\
    BondingConfigurations/BondingConfiguration(package1)

> delete_config_element unused_ports -in_wrapper $spec/BoundaryScan/\
    BondingConfigurations/BondingConfiguration(package1)

> set_config_value $spec/BoundaryScan/BondingConfigurations/\
    BondingConfiguration(package1)/\
    unused_ports {D1[2] D1[1] D1[0] D1[3]}

> report_config_data

> set spec [create_dft_specification]
> report_config_data $spec

DftSpecification(cpu_top,rtl) +{
  IjtagNetwork +{
    HostScanInterface(tap) +{
      Interface {
        tck : tck_p;
        trst : trst_p;
        tms : tms_p;
        tdi : tdi_p;
        tdo : tdo_p;
      }
      Tap(main) +{
        HostIjtag(1) {
        }
        HostBscan {
        }
      }
    }
  }
  BoundaryScan {
    ijtag_host_interface : Tap(main)/HostBscan;
    BondingConfigurations {
      BondingConfiguration(standard) {
      }
      BondingConfiguration(package1) {
        unused_ports : D1[2], D1[1], D1[0], D1[3];
      }
    }
  }
}
```

Related Topics

Configure the DFT Specification With DFTVisualizer

Validate the DFT Specification

In this optional step, you can validate the edits you made to the DFT specification to make sure no errors exist before you proceed to the next step.

Example

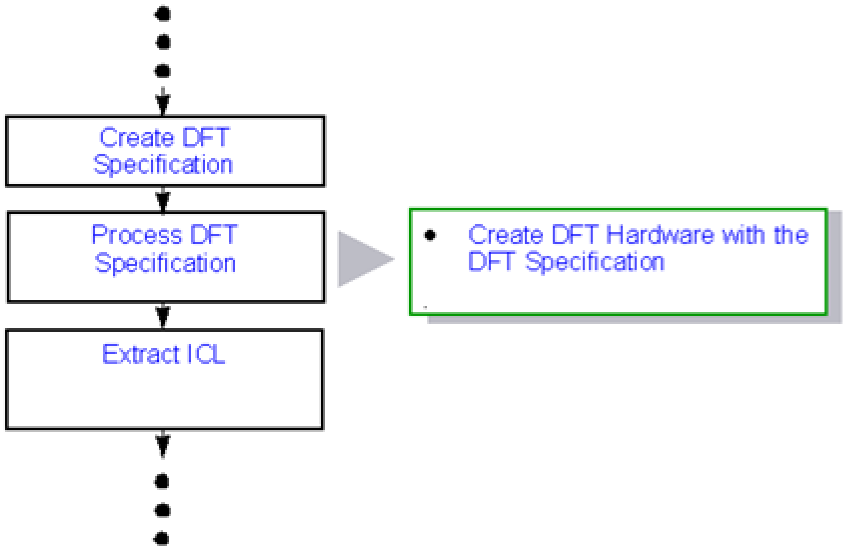
The following example validates your DFT specification.

```
process_dft_specification -validate_only
```

Process DFT Specification

The next step is to process the DFT specification that was created, edited, and validated in the previous step. This step creates and inserts the hardware for all the components that are in the DFT specification.

Figure 2-6. Process DFT Specification



Create DFT Hardware with the DFT Specification

Use the process_dft_specification command to generate and insert into the design all DFT hardware requested with the DFT specification. For Tessent BoundaryScan, the TAP controller and boundary-scan cells are inserted. IJTAG is used because boundary scan connects to the TAP, and the TAP is an IJTAG node.

Examples

Example 1

The following example generates and inserts into the design the hardware requested with the DFT specification.

```
process_dft_specification
```

Example 2

The following example generates the hardware requested with the DFT specification but does not insert the hardware into the design.

```
process_dft_specification -no_insertion
```

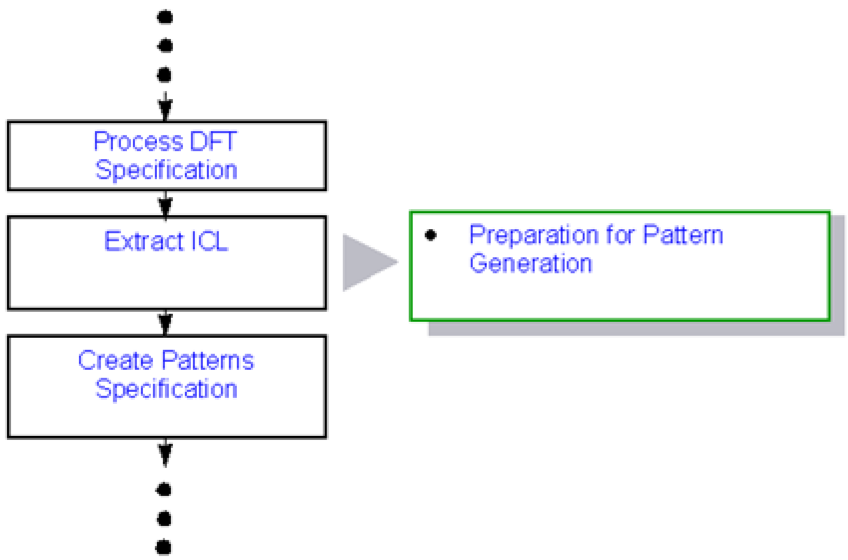
Extract ICL

The Extract ICL step verifies the proper connectivity of the ICL modules that were inserted with the process_dft_specification command. If no DRC violations are detected, the top-level ICL description is extracted.

The extract_icl command also creates an SDC file that can be used for synthesis. Refer to the "RTL Design Flow Synthesis" section for more information.

Downstream tools use the top-level ICL description for creating patterns. You can use the open_visualizer command to debug any ICL extraction DRC violations that are reported.

Figure 2-7. Extract ICL



Preparation for Pattern Generation

In this step, use the extract_icl command to find all modules (both Tessent instruments and non-Mentor Graphics instruments) with their associated ICL modules. If no DRC violations are detected, the ICL description of the root design is extracted.

The root of the design was specified with the set_current_design command during design elaboration in the Load the Design step. The Create Patterns Specification and Process Patterns Specification steps use the ICL description that was created for the root of the design. You can use the open_visualizer command to debug ICL extraction DRC violations. Refer to the "DFTVisualizer" chapter in the *Tessent Shell User's Manual*.

Example

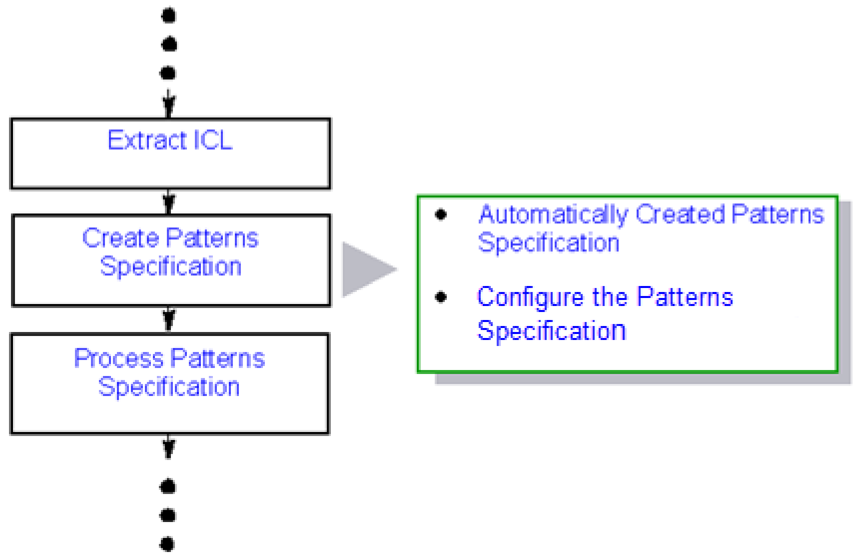
The following example extracts all ICL modules to the root of the design.

```
extract_icl
```

Create Patterns Specification

The Create Patterns Specification step creates the default patterns specification. The patterns specification is a configuration file that tells you what tests are being created. You can edit or configure the default patterns specification to generate the patterns specification you want.

Figure 2-8. Create Patterns Specification



Automatically Created Patterns Specification

The default patterns specification, which is created with the create_patterns_specification command, is only stored in memory.

To see the specification, use the report_config_data command.

Example

The following example creates a default patterns specification and stores the specification in a variable called pat_spec and then uses this variable to report the patterns specification in memory. The use of the Tcl variable is not necessary and is only used for convenience.

```
set pat_spec [create_patterns_specification]
report_config_data $pat_spec
```

For a full description of the patterns specification, see the section on the PatternsSpecification wrapper in the Tessent Shell Reference Manual. The following example shows a PatternsSpecification wrapper for module “MyDesign” with design_identifier “rtl” and pattern_id set to signoff. The SIM_PRECISION parameter is automatically set to provide accuracy of each clock period to within 1%, but it can also be set explicitly as shown here using the AdvancedOptions/Parameters wrapper.

```
PatternsSpecification(TOP, patt_id ,signoff) {
  Patterns(ICLNetwork) {
    ICLNetworkVerify(TOP) {
    }
  }
  Patterns(JtagBscanPatterns) {
    tester_period : 100ns;
    TestStep(JtagBscanTestStep) {
      BoundaryScan {
        bonding_configuration : option1;
        RunTest(test_logic_reset) {
        }
        RunTest(inst_reg) {
        }
        RunTest(id_reg) {
        }
        RunTest(bypass_reg) {
        }
        RunTest(bscan_reg) {
        }
        RunTest(input) {
        }
        RunTest(sample) {
        }
        RunTest(highz) {
        }
        RunTest(clamp) {
        }
        RunTest(output) {
        }
      }
    }
  }
}
```

Configure the Patterns Specification

Use one of the three methods to edit or create a patterns specification according to your requirements. Typically, you do not need to edit the signoff patterns specification; only the manufacturing patterns specification may need editing based on your requirements.

Method 1: Edit the patterns specification in memory

This is the preferred method because as you edit the patterns specification in memory, the commands that are used are specified in the Tcl or dofile and, therefore, the edits are repeatable for the next iteration by using only scripts.

Example

The following example shows that the default value for tester_period is 100ns and can be changed by editing the patterns specification in memory.

```
set pat_spec [create_patterns_specification]

report_config_data $pat_spec

set_config_value $pat_spec/Patterns(JtagBscanPatterns)/tester_period 50ns
```

Method 2: Write out the patterns specification, edit the file, and read the file back in

This method may be easier, but keep in mind that every time the master Tcl script or dofile runs, the patterns specification that is written out will overwrite your edits. To make sure your edits are reusable and repeatable using scripts, make a copy of the patterns specification and then edit the specification before reading it back in.

Example

The following example writes the patterns specification into a file called bonding1_config.pat_spec. After making the edits in a copy of this file, this file is read back in. Note that the file that is written out is different from the edited file that is read back in.

```
create_patterns_specification

report_config_data

write_config_data bonding1_config.pat_spec -wrappers
  PatternsSpecification(car,signoff)

read_config_data bonding1_config_edited.pat_spec

report_config_data
```

When manually editing the patterns specification, it is highly recommended that you validate the specification as shown in the following example.

```
process_patterns_specification -validate_only
```

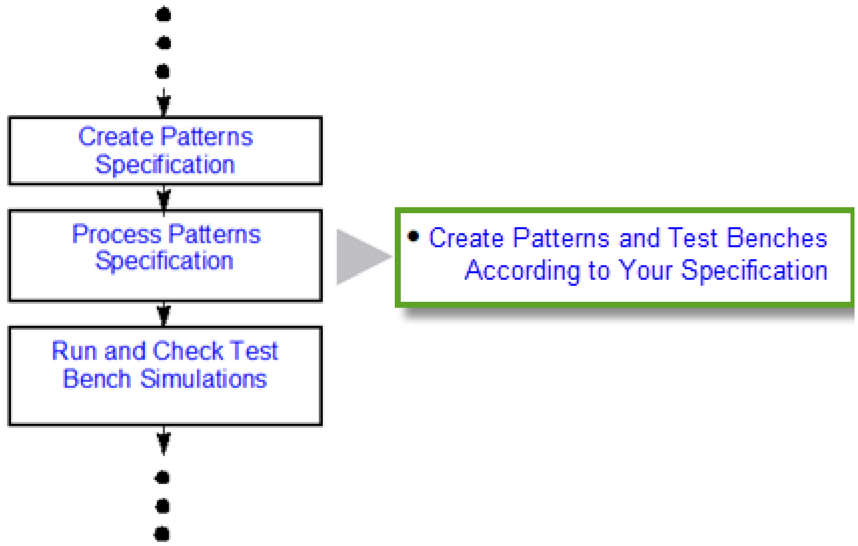
Method 3: Use the GUI to edit the Patterns Specification

The third option is to use the display_specification command and edit the patterns specification through the GUI.

Process Patterns Specification

The process patterns specification step of the design flow creates the patterns and test benches.

Figure 2-9. Process Patterns Specification



Create Patterns and Test Benches According to Your Specification

In this step of the design flow, you create the patterns or test benches according to either the default patterns specification or to the edited patterns specification that you created in the previous step.

Example

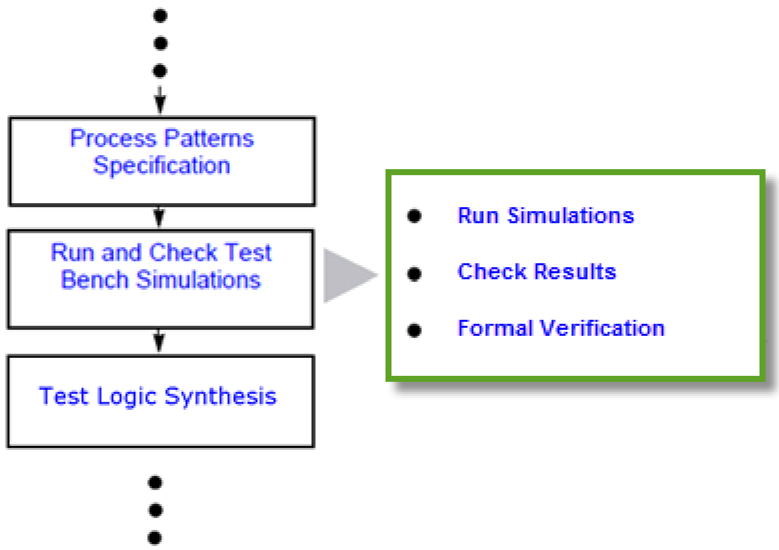
The following example generates the test benches.

```
process_patterns_specification
```

Run and Check Test Bench Simulations

Running and checking test bench simulations step is the last step in the Tessent BoundaryScan insertion using Tessent Shell. In this step, you run simulations of the boundary scan verification and then check the results.

Figure 2-10. Run and Check Test Bench Simulations



Run Simulations

Use the run_testbench_simulations command to invoke a simulation manager to run a set of simulation test benches.

The run_testbench_simulations command compiles and simulates test benches, generated for the TAP and boundary scan from the process_patterns_specification command, that are located at tsdb_outdir/patterns/<design>.patterns_signoff.

For a detailed description of the run_testbench_simulations command and its usage, see the *Tessent Shell Reference Manual*.

Example

The following example runs simulations of all patterns defined in the PatternsSpecification.

```
set_simulation_library_sources -y ../library/verilog \
   -v ../library/pad_cells.v
run_testbench_simulations
```

Check Results

Use the check_testbench_simulations command to check the status of the simulations that were previously launched by the run_testbench_simulations command.

For a detailed description of the check_testbench_simulations command and its usage, see the *Tessent Shell Reference Manual*.

Example

The following example checks the simulation results for errors.

```
check_testbench_simulations -report_status
```

Formal Verification

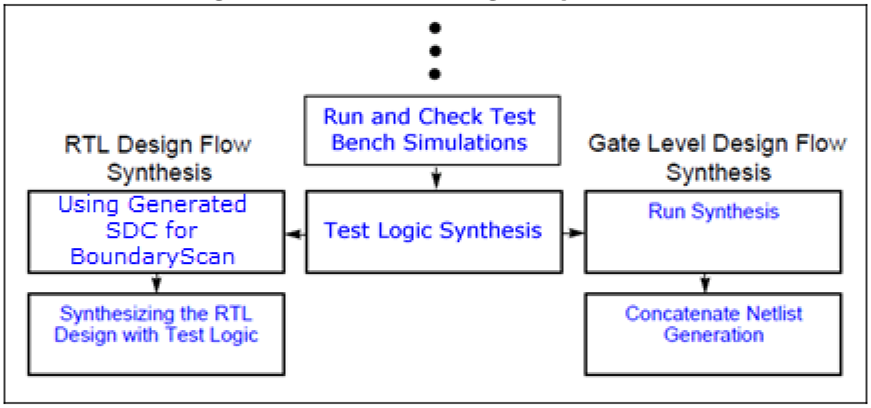
Tessent Shell-based products currently do not generate scripts for use with Synopsys® Formality®. You can however, set constraints in your design and manually create a script that is used with Formality.

For guidance on how this is accomplished, refer to the "Formal Verification" Appendix in the *Tessent Shell User's Manual*.

Test Logic Synthesis

The test logic synthesis process is different when handling a RTL or gate level design. The following sections outline the different options.

Figure 2-11. Test Logic Synthesis



RTL Design Flow Synthesis

The RTL synthesis flow that integrates the BoundaryScan RTL and associated test logic with the design RTL is an automated flow. The following sections outline the process.

Using Generated SDC for BoundaryScan

The extract_icl and extract_sdc commands both create a Synopsys Design Constraints (SDC) file that can be used for synthesis, layout and static timing analysis (STA). Since extract_sdc command requires ICL, it needs to be run after extract_icl.

When extract_icl is run on a physical block containing sub-blocks, the SDC constraints are generated for the physical module as well as the sub-blocks.

The created SDC is composed of several procedures that can be integrated into a design synthesis script.

- **tessent_set_default_variables —** This proc defines the default value of the variables used in instrument timing constraints. It is provided as a template. The user should override the values of these variables to correspond to their setup. For example, the values of the array "tessent_clock_mapping" should be overwritten with the user's chosen names for the functional clocks.
- **tessent_create_functional_clocks —** This proc defines the functional clocks used by the instrument constraints. This proc is not typically called since most users will define their functional clocks in their own timing scripts.
- **tessent_constrain_<design_name>_non_modal —** This proc defines the constraints for the BoundaryScan instrument.

For more information and examples on how to use the generated SDC procs, refer to the "Timing Constraints SDC" chapter in the *Tessent Shell User's Manual*. Additional information is also provided specific to BoundaryScan Instrument proc usage.

Synthesizing the RTL Design with Test Logic

This process is automated by a script that can be created and then processed by a synthesis tool to synthesize an RTL design that has been DFT inserted.

The write_design_import_script command can be used to generate a script that can be processed by a synthesis tool to load the RTL design that has been DFT inserted. The script file written can be combined with the SDC generated during extract_icl to synthesize a physical block or chip design unit.

Gate Level Design Flow Synthesis

The gate level design flow synthesis is a fully automated flow and only requires the run_synthesis command to synthesize the test logic and integrate into the design,

Run Synthesis

The run_synthesis command only synthesizes test logic RTL contained within the TSDB.

When creating and inserting memory BIST, boundary scan or IJTAG logic, the generated RTL is automatically written to the TSDB during process_dft_specification.

The run_synthesis command to invokes a synthesis manager to perform synthesis of the test logic RTL

For a detailed description of the run_synthesis command and its usage, refer to the *Tessent Shell Reference Manual*.

Example

The following example performs synthesis for a design and can be run at the physical_block or top design level.

```
run_synthesis
```

Concatenate Netlist Generation

When run_synthesis completes successfully, a concatenated netlist of the design which contains the synthesized test logic and modified design modules will automatically be created and placed in the dft_inserted_designs directory of the TSDB.

Chapter 3
Boundary Scan Specific Topics

Topics within this chapter cover a variety of subjects that describe cell library requirements as well as insertion of boundary scan cells in different design scenarios.

Pad Cell Library

A Tessent cell library for pad cells is needed for inserting TAP and BoundaryScan using Tessent tools. The Tessent cell library is an integrated library that contains a functional description for simulation as well as attributes for test logic insertion for each cell.

For further information on how to generate a Tessent cell library, refer to the *Tessent Cell Library Manual*.

The following sections describe alternate methods if you already have a *pad.library* file, which contains pad IO cell descriptions and verilog simulation models for the pad cells.

Using pad.library and Verilog Simulation Models for the Pad Cells

Use this procedure if you have pad cell Verilog simulation models and a *pad.library* file that contains pad cell IO descriptions.

Prerequisites

- *pad.library* file containing pad IO cell descriptions.
- Verilog models for the pad IO cells describing the functional behavior.
- Tessent Shell is invoked and ready to begin the Read the Libraries step of Load the Design as shown in Figure 2-1.

Procedure

1. Read the *pad.library* file from the appropriate directory, as shown in the following example.

   ```
   SETUP>read_cell_library ../library/pad.library
   ```

2. Read the Verilog models for the pad cells from the appropriate directory, as shown in the following example.

   ```
   SETUP>read_verilog ../library/verilog/pads.v
   ```

Results

The attributes of the pad cells described within the *pad.library* file will be applied on top of the Verilog description of the pad cells that were read in using the read_verilog command.

Creating a Tessent Cell Library from pad.library and Verilog Simulation Models

The Tessent Cell Library is not needed if you have a *pad.library* file and Verilog simulation models. There is an optional way of creating a Tessent Cell Library utilizing the *pad.library* and Verilog simulation models.

Prerequisites

- *pad.library* file containing pad IO cell descriptions
- Verilog models for the pad IO cells describing the functional behavior

Procedure

1. Invoke the LibComp tool from the unix shell prompt to compile a *libcomp.atpglib* file from the pad cell Verilog simulation models. The first parameter passed is the path to the Verilog simulation models for the pad cells.

   ```
   UNIX>libcomp pads.v -dofile
   ```

   Note
   You do not need to provide any do files with the -dofile switch.

   A *libcomp.atpglib* file will be created for all the cells present in the *pads.v* file. Ensure that any includes using “include” in the *pads.v* file are properly read in.

   Note
   For more information about libcomp startup options, see libcomp in the *Tessent Cell Library Manual.*

2. Invoke Tessent Shell and optionally create a log file for reference.

   ```
   UNIX>tessent -shell -log create_cell_lib.log
   ```

3. Read the *pad.library* and the *libcomp.atpglib* file created in step 1 using read_cell_library at the Tessent Shell prompt.

   ```
   SETUP>read_cell_library pad.library
   SETUP>read_cell_library libcomp.atpglib
   ```

4. Write the new Tessent Cell Library using write_cell_library command.

   ```
   SETUP>write_cell_library pads.tcell_lib
   ```

Results

The created Tessent Cell Library can be read into Tessent Shell using the read_cell_library command.

If LibComp is unable to translate a cell in the *pads.v* file into an *atpg.lib* model, an empty blackbox model is created for that cell. In this situation, the translation of these cells into the Tessent Cell Library as functional library elements is not completed and they will have to be manually edited into valid functional models.

Related Topics

read_cell_library [Tessent Shell Reference Manual]

write_cell_library [Tessent Shell Reference Manual]

Customizing Boundary Scan Pin Order

The boundary scan pin order that was initially created can be modified and saved to satisfy design requirements and ensure repeatability of the BIST implementation.

The initial boundary scan pin order can be created from pin placement information provided in a Design Exchange Format (DEF) file, which can be loaded during the Load the Design step. If a DEF file is not loaded, the initial pin order is obtained from the declaration order of the ports in the design file. Customization of the boundary scan pin order may be needed with a layout sorted pin order, and is most certainly needed for a port declaration ordered boundary scan chain to minimize routing.

Prerequisites

- If you want to have the initial boundary scan pin order based on port layout placement, the DEF file should be loaded during the Load the Design step. All the steps for loading a design should be completed.
- Specify and Verify DFT Requirements steps are completed.

Procedure

1. Edit the boundary scan pinorder file with the pin order you want, and save it to either the same or a different file name.

   The boundary scan pinorder file is created once an error-free Design Rule Checking (DRC) result is obtained with the check_design_rules command in the Run DRC step of the flow. The pinorder file is named *<design_name>.pinorder* and is located in the Tessent Shell Data Base (TSDB) tsdb_outdir/dft_inserted_designs folder. An example of a pinorder file is shown in Figure 3-1.

Figure 3-1. Example PINORDER File

```
//--------------------------------------------------------
//  File created by: Tessent Shell
//          Version:
//       Created on:
//--------------------------------------------------------


// PortName          PinName OptionList LogicalGroups
// --------          ------- ---------- -------------
   clk1_p            I1      -          -
   clk2_p            I2      -          -
   clk3_p            I3      -          -
   clk4_p            I4      -          -
   ramclk_p          I5      -          -
   reset_p           I6      -          -
   enable_p          I7      -          -
   paddr_p[10]       O1      -          -
   paddr_p[9]        O2      -          -
   paddr_p[8]        O3      -          -
   paddr_p[7]        O4      -          -
       .
       .
       .
```

2. On subsequent iterations, read in the customized pinorder file while in SETUP mode, prior to running check_design_rules as shown in this example:

   ```
   SETUP>set_boundary_scan_port_options -pin_order_file \
         ./tsdb_outdir/dft_inserted_designs/cpu_top.pinorder.modified
   ```

   The specified pinorder file will be validated during the check_design_rules command rather than a pinorder file being created. The pinorder file that was specified will be shown in the pin_order_file : *filename* property of the DftSpecification/BoundaryScan wrapper when the DftSpecification is created.

Results

Once the DFT hardware specified in the DftSpecification is created and inserted using process_dft_specification, the boundary scan chain order can be observed within the port listing of the BSDL file that is created. The BSDL file is located in the TSDB *<root_directory>/ instruments/<design_name>_<design_id>_bscan.instrument* folder and is named *<design_name>.bsdl*.

Related Topics

Tessent Shell Data Base (TSDB) [Tessent Shell Reference Manual]

set_boundary_scan_port_options [Tessent Shell Reference Manual]

DftSpecification/BoundaryScan wrapper [Tessent Shell Reference Manual]

process_dft_specification [Tessent Shell Reference Manual]

Inserting Tessent Boundary Scan on a Custom or Preexisting TAP Controller

The Tessent Shell design flow natively supports a custom or preexisting TAP controller connection to Boundary Scan cells. Custom or preexisting TAP controllers can also be connected to other controllers, such as memory BIST or hybrid TestKompress/LBIST.

For Tessent Shell to work with the preexisting TAP controller, an ICL(Instrument Connectivity Language) description for the TAP controller needs to be created. The preexisting TAP controller must have the port functions described in the "Requirements on a TAP to be usable for BoundaryScan" section within the DftSpecification/BoundaryScan wrapper description in the *Tessent Shell Reference Manual*.

Prerequisites

- An ICL description for a the custom or preexisting TAP controller.
- A TAP controller that meets the requirements outlined in the DftSpecification/BoundaryScan wrapper description.

Procedure

1. Read in the Verilog model and the ICL description of the preexisting or custom TAP.

```
SETUP>read_verilog design/my_custom_TAP.v
SETUP>read_icl design/my_custom_TAP.icl
```

   If there is only a single HostBscan interface defined, the boundary scan chain will be connected to the preexisting TAP controller that is read in if all the requirements of the TAP controller are met.

2. If the preexisting TAP controller has more than one HostBscan interface, you can explicitly specify which interface the boundary scan chain will be connected to using the following:

```
ANALYSIS>create_dft_specification -existing_bscan_host_scan_in \
my_custom_TAP_INST/fromBscan1
```

   my_custom_TAP_INST in the example above is the instance name of the preexisting custom TAP controller in the design and fromBscan1 is the port on the preexisting TAP where the boundary scan chain is to be connected.

Related Topics

create_dft_specification [Tessent Shell Reference Manual]

read_icl [Tessent Shell Reference Manual]

read_verilog [Tessent Shell Reference Manual]

Read the Design

Instrument Connectivity Language (ICL) [Tessent Shell Reference Manual]

Sharing TAP Ports Between a Preexisting TAP and Tessent TAP

TAP pins can be shared between a preexisting TAP and the Tessent TAP controller in situations where a preexisting TAP does not meet DftSpecification requirements or there is a design requirement to not disturb the preexisting TAP controller.

Follow this procedure to add a Tessent TAP controller to connect to the boundary scan chain in cases where:

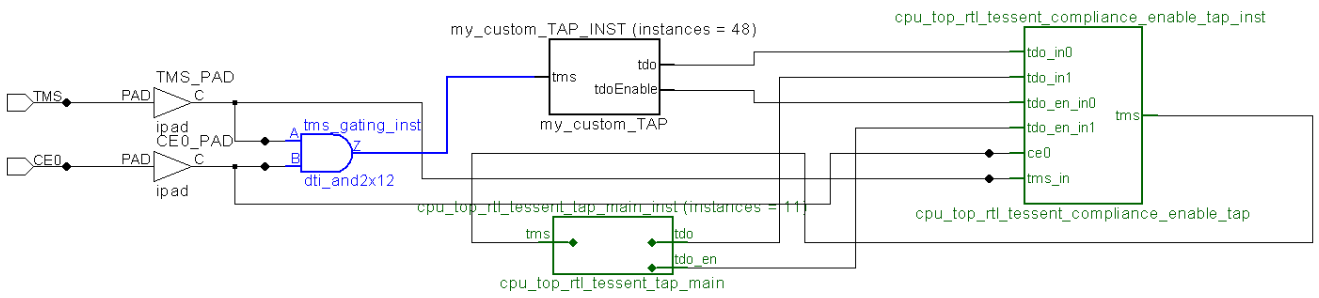
- A preexisting TAP controller is not to be disturbed.
- The preexisting custom TAP controller does not meet the requirements described in the DftSpecification/BoundaryScan wrapper description and cannot be used for boundary scan.

Prerequisites

- The IEEE 1687 standard requires that the state of a TAP controller should not change when its scan path is not selected. When implementing the methodology outlined in this procedure, the preexisting TAP will have its own TMS gating. When the Tessent TAP is selected by its specified compliance enable signals, the preexisting TAP is gated off, preserving its state.

  Figure 3-2 shows the example implementation outlined in this procedure. The preexisting TAP module is my_custom_TAP, shown in black. The preexisting TAP does not need to be described in ICL. The TMS gating logic for this TAP is shown in blue, and must also pre-exist in the design. The complexity of the TMS gating logic depends on how many compliance enables are specified to enable the Tessent TAP. The Tessent TAP and compliance enable module inserted by Tessent Shell in this procedure are shown in green.

Figure 3-2. Sharing TAP Ports Example



Procedure

1. Specify the TAP pins during the "Specify DFT Specification Requirements" section of the design flow, as shown in the following example:

```
SETUP> set_attribute_value tck_p -name function -value tck
SETUP> set_attribute_value tdi_p -name function -value tdi
SETUP> set_attribute_value tms_p -name function -value tms
SETUP> set_attribute_value trst_p -name function -value trst
SETUP> set_attribute_value tdo_p -name function -value tdo
```

2. Specify the compliance enable (CE) pins during the "Create DFT Specification" section of the design flow:

```
ANALYSIS> create_dft_specification \
         -active_low_compliance_enables CEO \\<pin_port_list>
```

For this example, there is a single active low compliance enable pin named "CE0".

When specifying the compliance enable pins with create_dft_specification, the active_high_compliance_enables and active_low_compliance_enables properties within the HostScanInterface/Interface wrapper are filled with the named ports, and they are also added to the BoundaryScan/ BoundaryScanCellOptions wrapper as compliance_enable1 and compliance_enable0, respectively. For this example there would only be active_low_compliance_enables and compliance_enable0 entries as shown in the following:

```
ANALYSIS> report_config_data
DftSpecification(cpu_top,rtl) +{
  IjtagNetwork +{
    HostScanInterface(tap) +{
      Interface +{
        tck : tck_p;
        trst : trst_p;
        tms : tms_p;
        tdi : tdi_p;
        tdo : tdo_p;
        active_low_compliance_enables : CE0;
      }
      Tap(main) +{
        HostIjtag(1) {
        }
        HostBscan {
        }
      }
    }
  }
  BoundaryScan +{
    ijtag_host_interface : Tap(main)/HostBscan;
    BoundaryScanCellOptions {
      CE0 : compliance_enable0;
    }
  }
}
DefaultsSpecification(user) {
}
```

If boundary scan is to be inserted, then those pins listed with the compliance_enable0/ compliance_enable1 attributes will appear in the COMPLIANCE_PATTERNS attribute in the BSDL file. If an internal pin is provided in the list, then a warning is given that the BSDL file will not be IEEE Std 1149.1 compliant.

3. Complete the Process DFT Specification section of the design flow. Prior to starting the Extract ICL step, use the add_ijtag_logical_connection command to allow extract_icl to bypass the TMS gating to the preexisting TAP.

    a. Manually change the context to patterns -ijtag for add_ijtag_logical_connection:

    ```
    INSERTION>set_context patterns -ijtag
    ```

    b. Specify the logical path across the TMS gating logic to the preexisting TAP. In this example it would be from the output of TMS_PAD to the output of tms_gating_inst as seen from Figure 3-2.

    ```
    SETUP>add_ijtag_logical_connection -to tms_gating_inst/Z -from
    TMS_PAD/C
    ```

If this step is not completed, an I2 ICL Extraction error will occur.

4. Complete the Extract ICL step and continue with the remaining steps of the design flow.

Results

The Tessent TAP will be created and selected during boundary scan tests when the CE0 port is “0”. This value will automatically be set during boundary scan test simulations and will be documented in the generated BSDL file. The Tessent TAP module is named <*design_name*>_<*design_id*>_tessent_tap_<*id*>.

A module named “<*design_name*>_<*design_id*>_tessent_compliance_enable_<*id*>” will be created that muxes the TDO signal and gates the TMS signal to the Tessent TAP based on the compliance enable signal values.

If you are using a compliance enable pin and the logic to enable the TAP that is already present in the design, refer to Example 2 in the IjtagNetwork wrapper description in the *Tessent Shell Reference Manual*. This example shows the steps to follow when you want to connect the TAP controller to internal pins which are not directly on the pads associated to the TAP ports.

Related Topics

set_attribute_value [Tessent Shell Reference Manual]

create_dft_specification [Tessent Shell Reference Manual]

HostScanInterface/Interface [Tessent Shell Reference Manual]

BoundaryScanCellOptions [Tessent Shell Reference Manual]

AC JTAG

If there are any AC JTAG pad IO cells present in the design, then a Tessent Cell Library describing them needs to created and read in during the “Load the Design” portion of the design flow.

For more information, refer to Chapter 2 “Library Model Creation” of the *Tessent Cell Library Manual*.

If an old *pad.library* file is available and verilog models are present, then the procedure described in “Using pad.library and Verilog Simulation Models for the Pad Cells” can be used.

Related Topics

Load the Design

Embedded Boundary Scan

The embedded boundary scan (EBscan) flow is typically used where the pad IO cells are present inside a module that is either a sub-block or a physical region. If the boundary scan cells need to be present within this sub-block or the physical region, then the embedded boundary scan flow is used.

To implement this flow, the command set_boundary_scan_port_options is used to specify the ports that require embedded boundary scan cells. This process is performed in the Specify DFT Specification Requirements step after Load the Design completed. The following procedure shows an example of how this is done for both sub-block and physical region implementations.

Prerequisites

- Load the Design steps have been completed and the design is loaded and elaborated.

Procedure

1. Start the Specify and Verify DFT Requirements step by beginning to define the DFT specification requirements.

   ```
   ## Begin Specify and Verify DFT Requirements step
   SETUP>set_dft_specification_requirements -boundary_scan on
   ```

2. Specify at what level boundary scan is to be inserted.

   For sub-block implementation:

   ```
   SETUP>set_design_level sub_block
   ```

   For physical region implementation:

   ```
   SETUP>set_design_level physical_block
   ```

3. Specify the list of pad IO ports that need embedded boundary scan cells by using the set_boundary_scan_port_options command.

   ```
   ## The following will insert embedded boundary scan cells
   SETUP>set_boundary_scan_port_options -pad_io_ports [list in1 \
   in_diff_p in_diff_n out1 out_diff_p out_diff_n clk A Y]
   ```

4. Use the add_input_constraints command to specify the constants that are required for the pad IO to operate.

   Note

   Some of the pad IO constants that are required may come from the next higher level, either through test setup or other initialization setup.

```
##Constraints specified for pins where the value comes from the next
level
SETUP>add_input_constraints vss -C0
SETUP>add_input_constraints Ten3[2] -C0
SETUP>add_input_constraints Ten3[1] -C0
SETUP>add_input_constraints Ten4[2] -C0
SETUP>add_input_constraints vdd -C1
SETUP>add_input_constraints Ten1 -C1
SETUP>add_input_constraints Ten2 -C1
SETUP>add_input_constraints Ten3[0] -C1
SETUP>add_input_constraints Ten4[1] -C1
```

5. Run check_design_rules to verify the implementation and then continue with the rest of the design flow.

```
##Running DRC
SETUP>check_design_rules
```

Related Topics

add_input_constraints [Tessent Shell Reference Manual]

set_dft_specification_requirements [Tessent Shell Reference Manual]

set_boundary_scan_port_options [Tessent Shell Reference Manual]

Adding a Test Data Register to the TAP Controller

User-defined bits for enhancing test control and status monitoring can be added through the creation of a Test Data Register (TDR) that is inserted and connected to the TAP controller.

The Tdr/DataInPorts wrapper specifies the number of data-in ports to create for test status monitoring on the TDR, the naming of the ports, and the connections to make to them. The Tdr/DataOutPorts wrapper specifies the number of data-out ports to create for test control on the TDR, the naming of the ports, and the connections to make to them. For more information, refer to the Tdr section in the *Tessent Shell Reference Manual*.

The procedure below provides an example for how one can create a default DFT specification and manually add a TAP using the DFTVisualizer GUI. A method is shown for copying the DFT specification, inserting the TDR without using the DFTVisualizer GUI and reading the DFT Specification back into memory. This is used to facilitate creation of dofiles to make the process repeatable without having to manually edit it within DFTVisualizer.

Prerequisites

- Load the Design steps have been completed and the design is loaded and elaborated.
- Specify and Verify DFT Requirements steps are completed.

Procedure

1. Create a DFT specification and also direct the output to a Tcl variable:

```
##To create a dft specification
set spec [create_dft_specification]
```

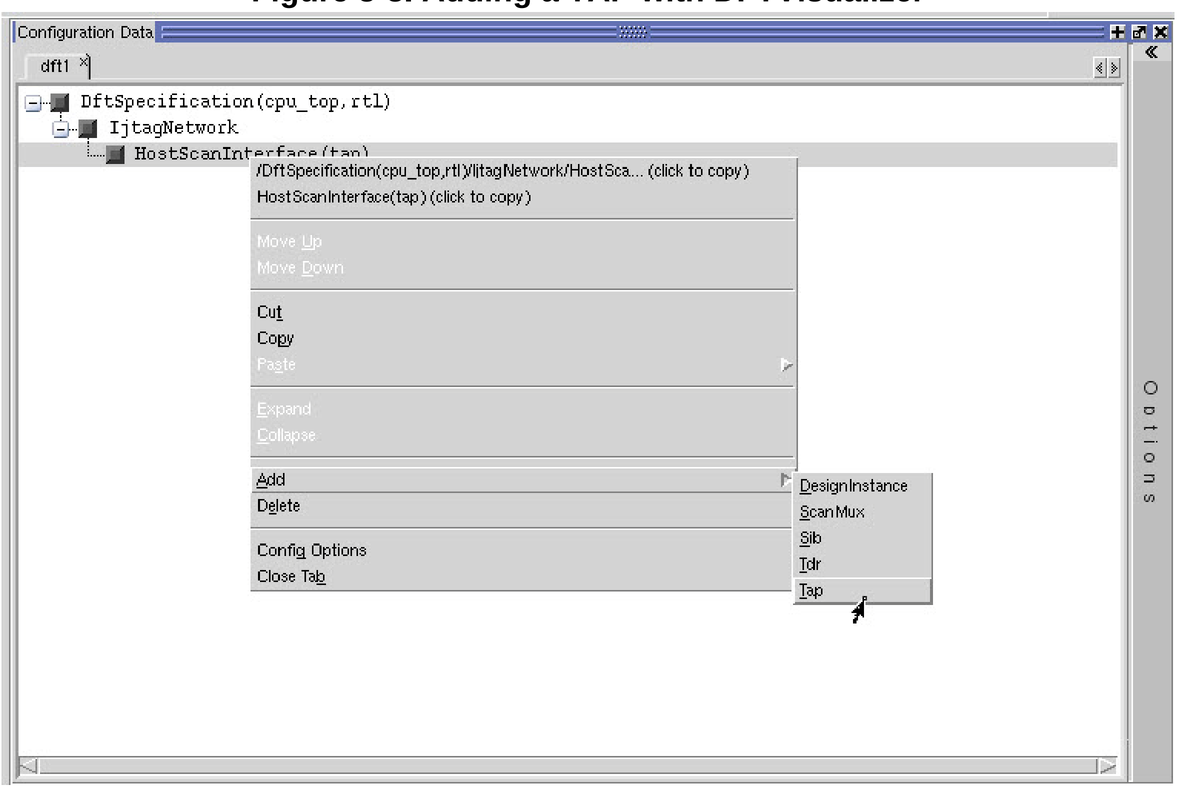
2. Invoke DFTVisualizer and display the Configuration Data Window and Configuration Options pane.

```
display_specification
```

3. Within DFTVisualizer, add in the TAP structure by selecting and highlighting the location within the DftSpecification tree, right clicking and selecting **Add>Tap**, as shown in Figure 3-3. The path this example implements to the TAP named "main" is:

```
/IjtagNetwork/HostScanInterface(tap)/Tap(main)
```

Figure 3-3. Adding a TAP with DFTVisualizer



4. In the Console window, use report_config_data to list the current structure.

```
report_config_data $spec
```

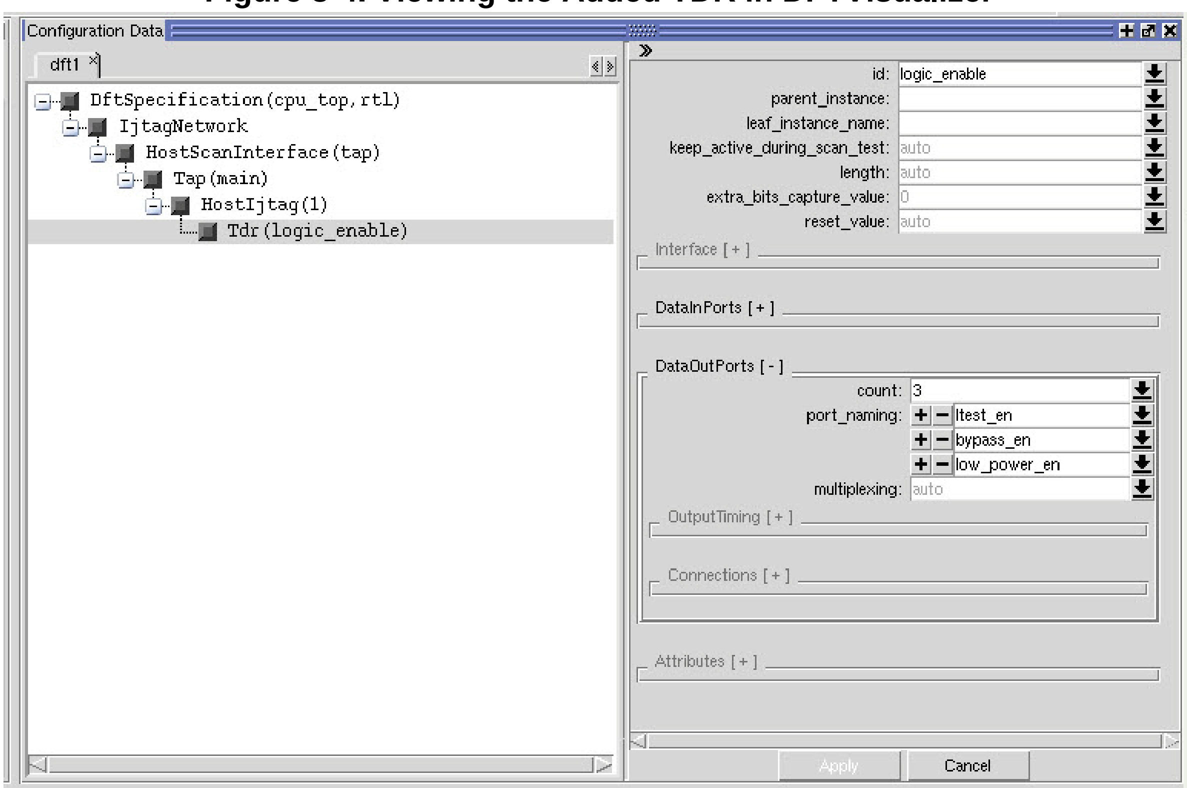
The configuration is reported to the console.

5. Once the configuration is reported, cut and paste the report information into the dofile, reading the DFTVisualizer edits implementing the TAP and the manually entered TDR, by using the read_config_data command as shown in the example below. Then, for subsequent runs, the configuration edits will already be present in the dofile, making the process repeatable via scripts.

```
read_config_data -in_wrapper $spec/IjtagNetwork/
HostScanInterface(tap)/Tap(main)/HostIjtag(1) -from_string {
   Tdr(logic_enable) {
    DataOutPorts +{
      count       : 3;
      port_naming : ltest_en, bypass_en, low_power_en;
      }
   }
}
```

6. The TDR that was added can be inspected within DFTVisualizer as shown in Figure 3-4.

Figure 3-4. Viewing the Added TDR in DFTVisualizer



7. Create the hardware from the DFT specification;

```
process_dft_specification
```

Related Topics

create_dft_specification [Tessent Shell Reference Manual]

display_specification [Tessent Shell Reference Manual]

report_config_data [Tessent Shell Reference Manual]

BSDL-Only Flow

Boundary scan patterns can be generated in Tessent Shell using a Boundary Scan Definition Language (BSDL) file created outside of the Tessent environment. No design data other than the BSDL file is required for this pattern generation flow; however, if needed, you can also include a Serial Vector Format (SVF) file in this flow.

Prerequisites

- A valid BSDL file.

Procedure

1. The configuration data to specify the patterns to be applied to DFT components that are inserted into a design is defined within the BoundaryScan wrapper in the PatternsSpecification. An example of the wrapper contents is shown in Figure 3-5. This would be saved as a file, and for this example it will be named example.patterns_spec_signoff. The third party BSDL file is specified in the BoundaryScan wrapper with the bsdl_file property.

Figure 3-5. Example PatternsSpecification for Third Party BSDL

```
PatternsSpecification(example,bscan,signoff) {
  Patterns(JtagBscanPatterns) {
    tester_period : 100ns;
    TestStep(JtagBscanTestStep) {
      BoundaryScan {
        bsdl_file : EP432P12.bsdl;
        RunTest(test_logic_reset) {
        }
        RunTest(inst_reg) {
        }
        RunTest(bypass_reg) {
        }
        RunTest(bscan_reg) {
        }
        RunTest(input) {
        }
        RunTest(sample) {
        }
        RunTest(highz) {
        }
        RunTest(clamp) {
        }
        RunTest(output) {
        }
      }
    }
  }
}
```

```
# dofile to create the above
set_context patterns -ijtag
set pat [create_patterns_specification -bsdl_file ./EP432P12.bsdl]
set_config_value $pat/Patterns(JtagBscanPatterns)/tester_period 100ns
process_patterns_specification
write_config_data example.patterns_spec_signoff
exit
```

In most cases, the BSDL description should be sufficient to generate patterns, but if an initialization sequence is needed it can be specified as follows:

```
PatternsSpecification(example,bscan,signoff) {
  Patterns(JtagBscanPatterns) {
    tester_period : 100ns;
    TestStep(JtagBscanTestStep) {
      BoundaryScan {
        bsdl_file : EP432P12.bsdl;
        RunTest(test_logic_reset) {
        }
        RunTest(inst_reg) {
        }
        RunTest(bypass_reg) {
        }
        RunTest(bscan_reg) {
        }
        RunTest(input) {
        }
        RunTest(sample) {
        }
        RunTest(highz) {
        }
        RunTest(clamp) {
        }
        RunTest(output) {
        }
      }
    }
    ProcedureStep(init) {
      svf_file : EP432P12.svf;
    }
  }
}


# dofile to create the above
set_context patterns -ijtag
set pat [create_patterns_specification -bsdl_file ./EP432P12.bsdl]
set_config_value $pat/Patterns(JtagBscanPatterns)/tester_period 100ns
set patterns [get_config_elements Patterns -in_wrappers $pat]
set init_proc [add_config_element ProcedureStep(init) -in $patterns]
set_config_value svf_file -in $init_proc EP432P12.svf
process_patterns_specification
write_config_data example.patterns_spec_signoff
exit
```

Note

The BSDL package include directory is assumed to be in the same location as the BSDL file. If it is not, specify the location with the bsdl_package_directories argument or create a symbolic link to the package directory.

For more explanation of the RunTest wrapper and parameters, refer to the BoundaryScan topic in the "Configuration-Based Specification" chapter in the *Tessent Shell Reference Manual*.

2. Create a Tcl file containing the Tessent Shell steps to set the proper context, read in the configuration wrapper and generate the patterns. An example file is shown in Figure 3-6 and will be named bsdl_only.tcl for this procedure.

Figure 3-6. Tessent Shell dofile for Third Party BSDL Processing

```
set_context pattern -ijtag -design_id bscan
read_config_data ./example.patterns_spec_signoff
process_patterns_specification
exit
```

3. Invoke Tessent Shell and specify bsdl_only.tcl as the dofile to execute.

```
tessent -shell -dofile bsdl_only.tcl -log bsdl_patterns.log -replace
```

Results

The test bench "JtagBscanPatterns.v" is created in the tsdb_outdir/Patterns/example_bscan.patterns_signoff directory. For details on how the tsdb_outdir directory structure is organized, refer to "Tessent Shell Data Base (TSDB)" in the *Tessent Shell Reference Manual*.

Related Topics

BoundaryScan wrapper [Tessent Shell Reference Manual]

RunTest [Tessent Shell Reference Manual]

PatternsSpecification wrapper [Tessent Shell Reference Manual]

Dividing Boundary Scan for Logic Test

In a majority of designs, the boundary scan chain needs to be included in the Logic testing portion of the design so the chip boundaries are isolated with a boundary scan chain. For most designs, the single boundary scan chain is too long as there is some type of compression or Built In Self Test (BIST) incorporated to test the logic portion of the design.

The boundary scan chain needs to be divided, or segmented into shorter chain lengths in this situation to match how the functional design flops are stitched into scan chains. The max_segment_length_for_logictest property in the BoundaryScan wrapper can be implemented to properly segment the boundary scan chain.

The example presented in this section will segment the boundary scan chain so a maximum of 200 flops are present in any segment.

Prerequisites

- Load the Design steps have been completed and the design is loaded and elaborated.

- DFT Specification requirements and constraints are defined.

Procedure

1. Run Design Rule Checking (DRC) to ensure all constraints are correct and move Tessent Shell from Setup to Analysis mode if no errors are found.

```
SETUP>check_design_rules
```

2. Create the "DftSpecification(design_name,id)" configuration wrapper and copy the newly created wrapper object to the variable "spec" to customize the specification later.

```
##Create a dft specification
ANALYZE>set spec [create_dft_specification]
```

3. Set the value for the max_segment_length_for_logictest parameter in the BoundaryScan wrapper to 200.

```
ANALYZE>set_config_value \   $spec/BoundaryScan/
max_segment_length_for_logictest 200
```

4. Create and insert the hardware for the TAP and boundary scan into the design.

```
ANALYZE>process_dft_specification
```

5. Inspect the results to confirm the boundary scan is now segmented into the smaller chain segments for inclusion with logic testing, and save the output for later reference if wanted.

   List the currently available instrument dictionaries created by process_dft_specification.

```
ANALYZE>get_instrument_dictionary -list
```

   A list will be returned similar to that shown below:

```
DftSpecification LibraryCells RtlCells
mentor::ijtag::DftSpecification
mentor::jtag_bscan::DftSpecification
mentor::memory_bisr
tshell_global
```

   List the scan chains and scan_in/scan_out ports created. This also identifies how many scan chains segments were created.

```
ANALYSIS>format_dictionary [get_instrument_dictionary \
mentor::jtag_bscan::DftSpecification  logic_test_scan_chains]
```

Optionally, save the output created in step 6 for later reference by using the script below. This saves the output to a file named “logic_test_scan_chains.dictionary”.

```
set fp [open logic_test_scan_chains.dictionary w]
puts $fp "set logic_test_scan_chains {"
puts $fp [format_dictionary [get_instrument_dictionary
mentor::jtag_bscan::DftSpecification  logic_test_scan_chains ] ]
puts $fp "}"
close $fp
```

Results

The single boundary scan chain that is connected between the TDI and TDO is now segmented, with muxes added in and controlled by logic_test_enable. The single boundary scan chain connectivity is also maintained.

Related Topics

create_dft_specification [Tessent Shell Reference Manual]

set_config_value [Tessent Shell Reference Manual]

format_dictionary [Tessent Shell Reference Manual]

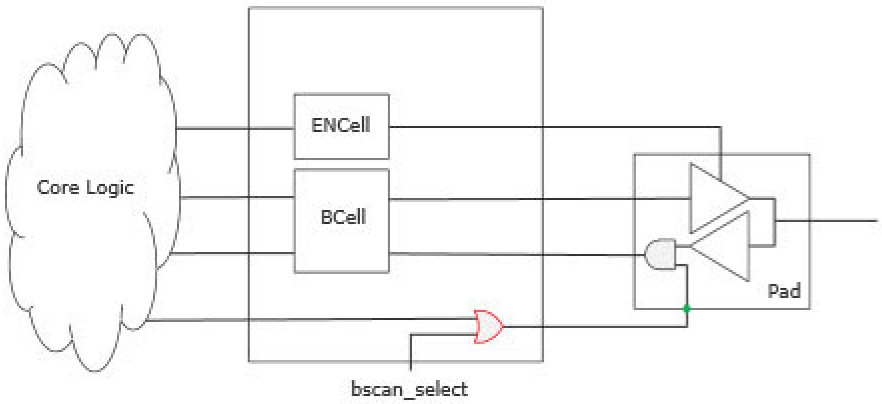
Pad Cell Input Path Considerations for Boundary Scan Testing

A simple pad cell input path will consist of a plain buffer connecting the pad cell pin assigned the pad_pad_io pin attribute to the pin assigned the pad_from_pad attribute. Bidirectional pad cells will also have a tri-state buffer on the output path. More complex input or bidirectional pad cells can have additional gating logic in the input path. The purpose of this additional gating logic in the input path is to prevent the core logic from toggling along with the signaling on the chip port.

During boundary scan testing, it is necessary to make the input path transparent so the value on the chip port can be observed in a boundary scan cell. Tessent Shell will add the necessary logic to properly drive the pad cell input enable port during boundary scan testing. The appropriate pad_input_enable_high or pad_input_enable_low Pin Attributes need to be applied to the pad cell input enable pin for this to be implemented by Tessent Shell.

Figure 3-7 shows a bidirectional pad cell with an active high input enable pad cell pin, highlighted by a green connection dot. This pad cell will need the pad_input_enable_high pin attribute applied to this pin.

Figure 3-7. Bidirectional Pad Cell with Active High Input Enable



Tessent Shell will automatically add the appropriate logic, shown in red for this example, to achieve the transparency needed on the input path during boundary scan testing. If core logic drives the pad cell input enable, the added logic will combine the boundary scan select with that signal. If the pad cell input enable was tied to a logic level, the added logic will utilize the same tie value. In the case of a pad cell configured with an auxiliary input, the auxiliary input enable signal will also be logically combined to properly activate the pad cell input enable.

Pad Cell Library Attribute Considerations for Boundary Scan Testing

Certain pad cell attributes will affect the way boundary scan logic is created and inserted into a design, depending on other considerations.

If a cell pad instance includes a port that has the pad_force_disable attribute, the corresponding pins on the instance will be checked for the has_functional_source attribute when DFT hardware is generated with the process_dft_specification command. If this attribute is true, a mux will be added during boundary scan insertion. The inputs will be the functional source and force disable signals, and the select pin of the mux will be connected to the boundary scan select signal.

If you use the “no insertion” flow (that is, the -no_insertion switch on the process_dft_specification command), you will need to add this circuitry manually.

For a full description of pin attributes, see the Tessent Cell Library Manual.

Multiple Bonding Configurations

This section describes the process flow to add support for multiple package bonding configurations.

When you have multiple package bonding configurations that need to be supported, you will need to edit the DftSpecification wrapper to add the various package options. One way, as demonstrated here, is to use DFTVisualizer to specify the bonding configurations and then read it into the main Tcl or dofile so the process is repeatable without manually editing in DFTVisualizer. You can also edit the DftSpecification as described in "Configure the DFT Specification in Memory" on page 40.

Prerequisites

- Load the Design steps have been completed and the design is loaded and elaborated.
- Specify and Verify DFT Requirements steps have been completed.

Procedure

1. Create a DFT specification and also direct the output to an environment variable. Report out the DFT specification for verification:

```
##To create a dft specification
ANALYSIS>set spec [create_dft_specification]

##Report on what dft spec was created
ANALYSIS>report_config_data $spec
```

2. Invoke DFTVisualizer to modify the dft specification so multiple bonding configurations can be added. DFTVisualizer will automatically open the Configuration Data Window and display the DftSpecification(<design_name>,<design_id>) wrapper configuration when display_specification is executed.

```
##Invoke DFTVisualizer and display the Configuration Data Window
ANALYSIS>display_specification
```

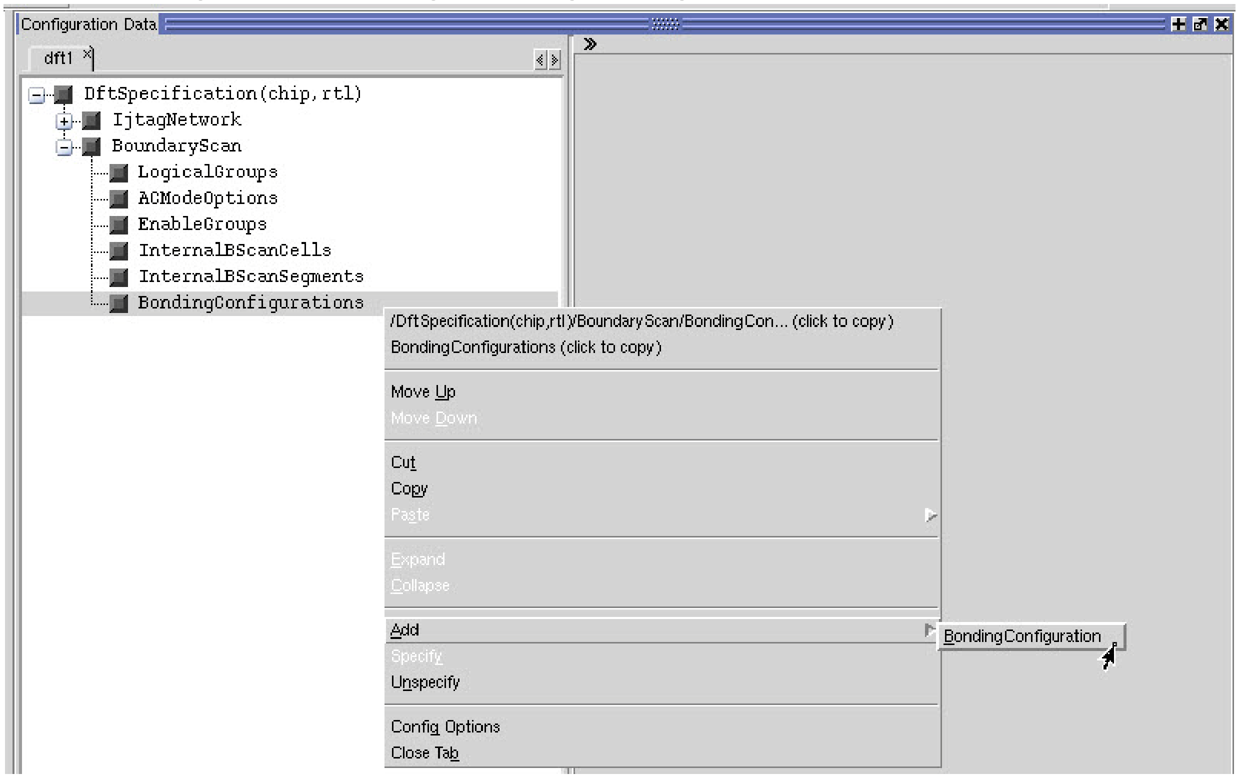
3. Expand the DftSpecification tree by clicking the + icon to expand DftSpecification/ BoundaryScan and display BondingConfigurations under BoundaryScan.

    a. Select BondingConfigurations in the tree and right click to display a menu of options available.

    a. Select **Add > BondingConfiguration** to add a configuration as shown in Figure 3-8.

    Repeat as needed for the number of bonding configurations required in your design. Two will be added for this example.

Figure 3-8. Adding Bonding Configurations in DFTVisualizer

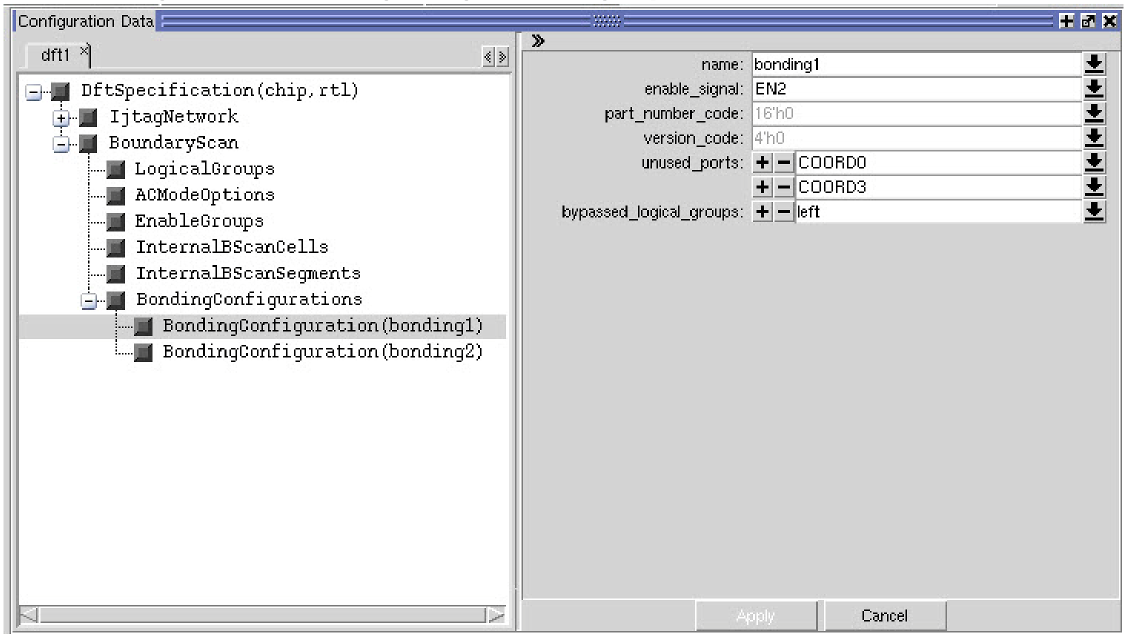


4. Click  at the top of the Options tab in the Configuration Data Window to expand the Configuration Options pane.

   a. Select a BondingConfiguration in the tree and fill in the options you want, as shown in Figure 3-9.

      The names are provided by the user and will be bonding1 and bonding2 for this example. Additionally, bonding1 has an enable signal EN2 and two unused ports, identified as COORD0 and COOR3. Finally, only the left logical group is bypassed during the bonding1 package option. If logical groups are bypassed, they must be provided in the pinorder file.

Figure 3-9. Configuring Bonding Options in DFTVisualizer



a. If needed, additional unused ports and bypassed logical groups can be added by clicking the  icon in those option fields.

For this example, the second BondingConfiguration is named bonding2 and has unused ports of D1[0], D1[1], D1[2] and D1[3].

5. Use report_config_data to list the current configuration:

```
ANALYSIS>report_config_data $spec
```

6. Once the configuration is reported, you can cut and paste this information into the dofile and read the DFTVisualizer edits using the read_config_data command as shown in the example below. Then, for subsequent runs, the configuration edits will already be present in the dofile, making the process repeatable via scripts.

```
read_config_data -in $spec/BoundaryScan -from_string {
BondingConfigurations +{
      BondingConfiguration(default) {
      }
      BondingConfiguration(bonding1) {
        enable_signal : EN2;
        unused_ports : COORD0, COORD3;
        bypassed_logical_groups : left;
      }
      BondingConfiguration(bonding2) {
        unused_ports : D1[2] D1[1] D1[0] D1[3];
      }
   }
}
```

7. If you are running Tessent Shell in the dft context with -no_rtl and the library does not have a clock_gating_and entry, the following operation needs to be completed to create the RTL for the AND clock gater.

```
set_config_value /DftSpecification(car,gate)/use_rtl_cells On
```

8. Complete any other DftSpecification configurations that may be required and continue the rest of the design flow.

Custom Boundary Scan Cells

Custom boundary scan cells can be combined with pad cells in the overall boundary scan chain. Custom boundary scan cells need to already be inserted into the design and connected to the pad cell.

Note
Tessent Shell does not support the insertion of custom boundary scan cells.

The custom boundary scan cell is described using the same format as found in the *.tcd_bscan file. This file is normally created by the process_dft_specification when inserting a boundary scan chain into a sub or physical block, and the block contains a Core(<module_name>)/ BoundaryScan wrapper. For this application, the wrapper will be manually created to specify the custom boundary scan cell and interface. Tessent Shell automatically recognizes this description when matching modules within higher level parent modules and uses it to stitch the boundary scan chain in the parent module.

Examples

Example 1

This example shows a sample *.tcd_bscan* file for custom input cells, named *ipad_bscan_combo.tcd_bscan*, and the process used for stitching them with pad cells that are automatically inserted.

The contents of the custom *ipad_bscan_combo.tcd_bscan* file is listed below:

```
Core(ipad_bscan_combo){
 BoundaryScan {
  Interface {
    select_jtag_input : SJI_in;
    capture_shift_clock : bscan_clk;
    update_clock : update_bscan;
    shift_en : shift_bscan;
    scan_in : bscanIn;
    scan_out : bscanOut;
    scan_out_launch_edge : negedge;
  }
  ExternalPort(PAD) {
  }
  Cell(cell0) {
    function : input;
    bsdl_cell_type : BC_2;
    external_port : PAD;
  }
 }
}
```

The *ipad_bscan_combo.tcd_bscan* file is located in the netlist directory along with the *ipad_bscan_combo.v* Verilog file in this example.

```
SETUP>set_context dft -no_rtl

##Read the Tessent library for standard cells and pad cells.
SETUP>read_cell_library ../library/adk_complete.tcelllib

#Read the verilog
SETUP>read_verilog netlist/cpu_top.v
SETUP>set_design_sources -format verilog -Y netlist -extensions v
SETUP>read_verilog netlist/ipad_bscan_combo.v
SETUP>set_current_design cpu_top

##set_dft_specification_requirements cannot be specified until the design
has been read in.
ANALYSIS>set_dft_specification_requirements -boundary_scan On

##Need to set the design level before running check_design_rules
ANALYSIS>set_design_level chip
```

The remainder of the design flow steps are unchanged beyond this point, and would merge at the Add Constraints step within "Specify and Verify DFT Requirements" on page 31.

Example 2

This example shows a sample *.tcd_bscan* file for custom output cells. In this case, the netlist will already have the output bscan cells inserted and connected, as well as a combinational cell for the enable. The process used for stitching the custom boundary scan outputs with pad cells that are automatically inserted is the same as that given in Example 1.

```
Core(opad_bscan_en_combo){
 BoundaryScan {
  Interface {
    select_jtag_output : SJO_in;
    force_disable      : forcedis;
    capture_shift_clock : bscan_clk;
    update_clock : update_bscan;
    shift_en : shift_bscan;
    scan_in : bscanIn;
    scan_out : bscanOut;
    scan_out_launch_edge : negedge;
  }
  ExternalPort(PAD) {
    buffer_type : three_state;
    control_cell : cell1;
  }
  Cell(cell0) {
    function : output;
    bsdl_cell_type : BC_2;
    external_port : PAD;
  }
  Cell(cell1) {
    function : control;
    bsdl_cell_type : BC_2;
    control_enable_value : 1;
  }
 }
}
```

Debugging Failing JtagBscan Simulations

Tessent Shell enables you to run and check test bench simulations. You can identify, view, and analyze failed simulation patterns. A series of examples is presented that outline the methods to accomplish each of these.

The command for running a set of simulation test benches in Tessent Shell is run_testbench_simulations. This command is normally run with no arguments because it automatically uses the design name, design id, and pattern id found in the previously processed PatternsSpecification(design_name, design_id, pattern_id) wrapper.

The command check_testbench_simulations is used to check the status of simulations that were previously launched by run_testbench_simulations. Arguments can be passed to generate a status report or a status Tcl dictionary. If no argument is provided, the command will update a status line each second while running, and report an error message for any failed simulations when completed.

Check Test Bench Status Example

The following example runs a set of test bench simulations based on the design name, design id, and pattern id from the previously processed PatternsSpecification. Test bench simulations are checked and any failing pattern simulations are reported.

```
SETUP>run_testbench_simulation
SETUP>check_testbench_simulations
// Error: 1 out of 2 simulations failed:
// JtagBscanPatterns with 943 unexpected miscompares
```

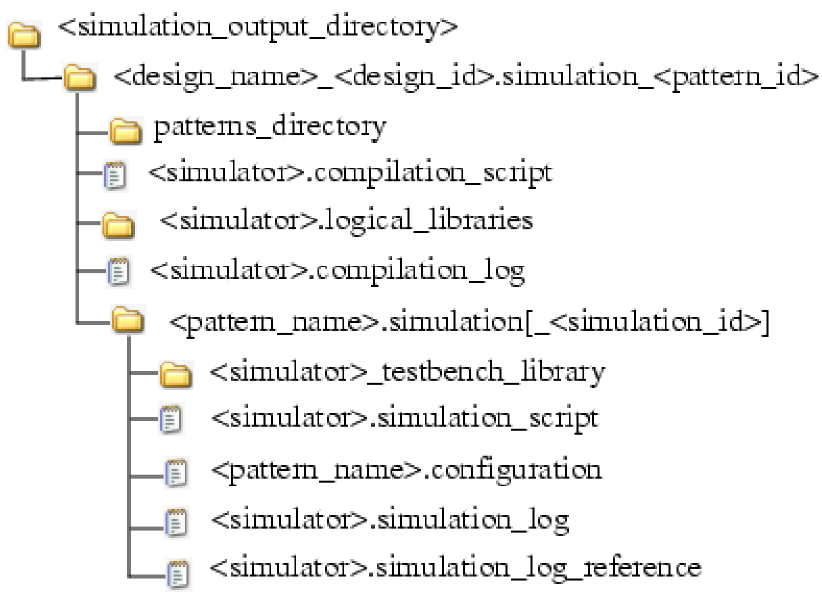
Identifying Detailed Simulation Status Example

This example shows another way to report test bench simulation status and the resulting output based on the same run_testbench_simulation shown in the previous example:

```
SETUP>check_testbench_simulations -report_status
// Simulation status for ./simulation_outdir/
cpu_top_gate.simulation_signoff
// ===================================================================
// -----------------  ------  -----------  -----------  ------------
// Pattern Name       Status  Unexpected   Missing      Date
//                            Miscompares  Miscompares
// -----------------  ------  -----------  -----------  ------------
// ICLNetwork         pass       0            0         Thu May 07
// JtagBscanPatterns  fail       943          0         Thu May 07
```

The -report_status argument creates a report listing the simulation status of each pattern found inside the <design_name>_<design_id>.simulation_<pattern_id> directory as shown in Figure 3-10. This report identifies any pattern id that failed test bench simulation and needs to be re-run to capture and store waveform data for further debug in the waveform viewer.

Figure 3-10. Simulation Output Directory Contents



Viewing Simulation Waveforms for Analysis Example

The following example shows how you would run simulation on a specific pattern_id that failed test bench simulation and then save the simulation waveforms for viewing in the waveform window for further analysis.

Availability of the failing pattern and design environment should be confirmed since a different PatternsSpecification could have been processed after the patterns that failed were run. The run_testbench_simulations command with the -report_list argument lists the patterns available in the *<design_name>_<design_id>.patterns_<pattern_id>/simulation.data_dictionary* file. This file is always updated upon successful validation and processing of the PatternsSpecification.

```
SETUP>run_testbench_simulations -report_list
List of pattern(s) for directory './tsdb_outdir/patterns/
cpu_top_gate.patterns_signoff':
  ICLNetwork  JtagBscanPatterns
```

Once the pattern availability is confirmed, as seen in the sample above with the listing of JtagBscanPatterns, the simulations can be re-run with waveform storage enabled.

```
SETUP>run_testbench_simulations -select JtagBscanPatterns \
       -store_simulation_waveforms on
```

Running run_testbench_simulations with the -store_simulation_waveforms argument enabled creates a *vsim.wlf* file in the directory where the JtagBscanPatterns were simulated.

In another unix shell window, navigate to the folder where the *vsim.wlf* file was created and open it using the viewer in Questa ModelSim as shown in the example below:

```
UNIX>cd \
     simulation_outdir/cpu_top_gate.simulation_signoff/ \
     JtagBscanPatterns.simulation
UNIX>vsim -view vsim.wlf
```

While analyzing the waveforms of the failing patterns, you have the option of rerunning the simulation in the Tessent Shell window with new files or other options such as delay_mode_zero, unit_delay or others. After any adjustments are made, you can reload the new *vsim.wlf* file in the waveform window to view the results.

Restoring a Previous Simulation Analysis Session Example

If for some reason you quit a Tessent -shell session similar to that outlined in the previous example, and returned later to continue analysis, you can read the simulation library in the new

Tessent -shell session using the set_simulation_library_sources command. The session can be restored by following the sequence shown:

```
UNIX>tessent -shell -log simulation_debug.log
SETUP>set_context patterns
SETUP>set_simulation_library_sources \
      -v ../library/adk_complete.v -v ../library/picdram.v
SETUP>run_testbench_simulations -report_list
List of pattern(s) for directory './tsdb_outdir/patterns/
cpu_top_gate.patterns_signoff':
  ICLNetwork  JtagBscanPatterns

SETUP>run_testbench_simulations -select JtagBscanPatterns \
      -store_simulation_waveforms on -wait
```

In another unix shell window, navigate to the folder where the *vsim.wlf* file was created and open it using the viewer in Questa ModelSim as shown in the example below:

```
UNIX>cd \
     simulation_outdir/cpu_top_gate.simulation_signoff/ \
     JtagBscanPatterns.simulation
UNIX>vsim -view vsim.wlf
```

Related Topics

run_testbench_simulations [Tessent Shell Reference Manual]

check_testbench_simulations [Tessent Shell Reference Manual]

set_simulation_library_sources [Tessent Shell Reference Manual]

Chapter 4
MemoryBIST Insertion with BoundaryScan

This chapter describes the design flow that can be followed if TAP, BoundaryScan, and MemoryBIST are to be inserted for a design.

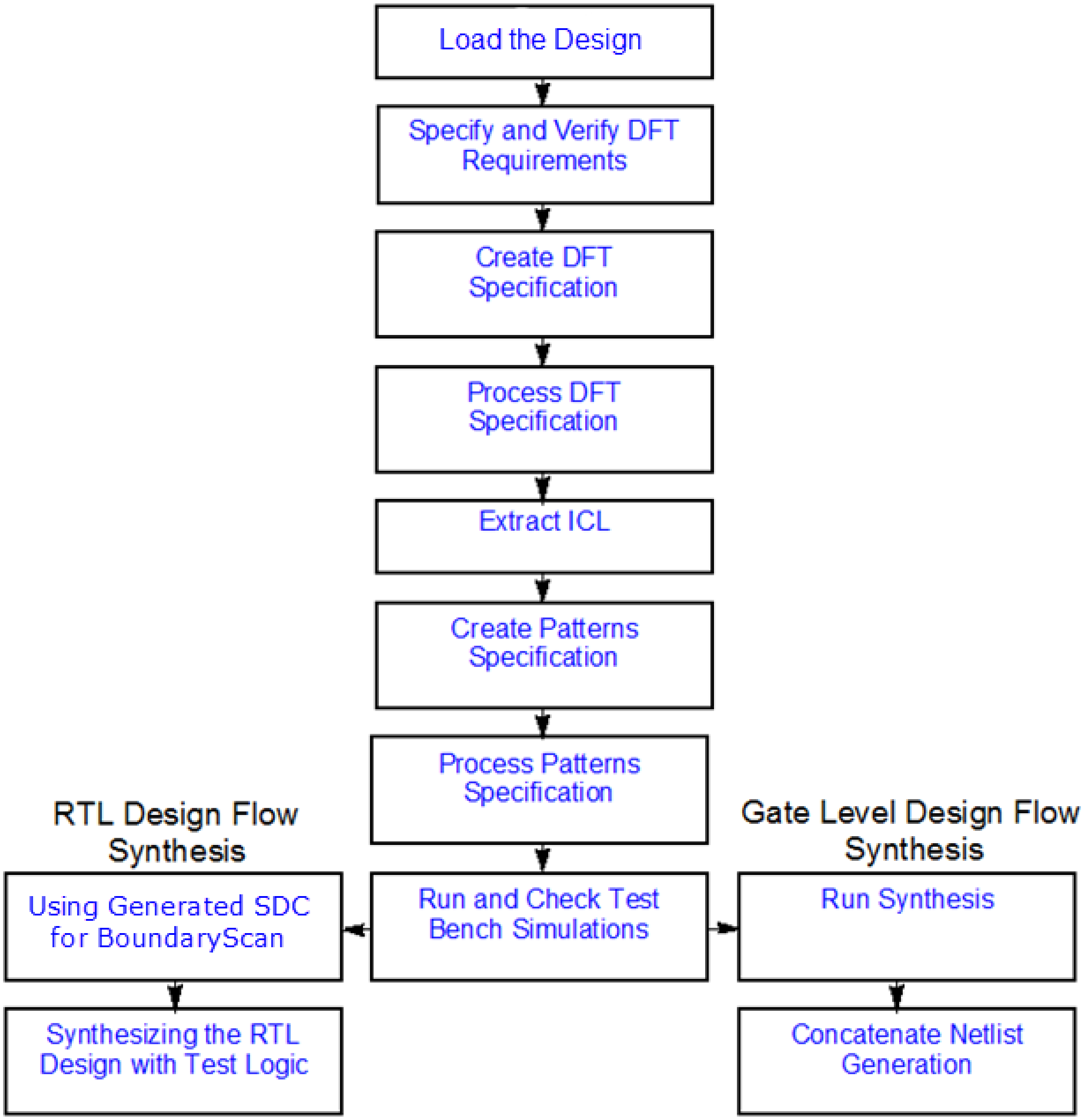
Overview

Two example design flows are described that show different methods of inserting MemoryBIST with BoundaryScan and TAP into a design.

One design flow inserts MemoryBIST, BoundaryScan and TAP in a single pass at the chip design level. The other flow inserts MemoryBIST in a first pass within a sub_block or physical_block, then inserting BoundaryScan and TAP in a second pass at the chip level. When memoryBIST needs to be inserted inside a physical_block or a sub_block, then the design level needs to also be set to physical_block or a sub_block. A physical_block describes a design module where physical layout will be completed, whereas a sub_block describes a design module that can be instantiated inside another layout region. The layout region can be another design module or the entire chip

The design flow that is used to insert MemoryBIST with BoundaryScan and TAP is the same as described in Getting Started With Tessent BoundaryScan and is shown in Figure 4-1 below.

Figure 4-1. Design Flow for Tessent Shell



Related Topics

Tessent MemoryBIST User's Manual for use With Tessent Shell

TAP, BoundaryScan, and MemoryBIST

The example provided in this section covers the design implementation for MemoryBIST, BoundaryScan and TAP all located at the chip design level. The example does this in a single implementation pass where the TAP, BoundaryScan and MemoryBIST are all generated and inserted at the same time. The generation and insertion could also be done for each in three separate passes.

Load the Design steps. In this step, the design and libraries are loaded in:

```
set_context dft -no_rtl
read_cell_library ../library/adk_complete.tcelllib
read_cell_library ../library/memory.lib
set_design_sources -format tcd_memory -V ../library/picdram.memlib
read_verilog ../netlist/cpu_top.v
set_current_design cpu_top
```

Specify and Verify DFT Requirements steps. In the example below, set_dft_specification_requirements has both -boundary_scan On and -memory_test On. The command set_design_level is set to “chip”. The DRC is run when the command check_design_rules is issued.

```
set_design_level chip
set_dft_specification_requirements  -memory_test on \
                                    -boundary_scan on

set_attribute_value tck_p -name function -value tck
set_attribute_value tdi_p -name function -value tdi
set_attribute_value tms_p -name function -value tms
set_attribute_value trst_p -name function -value trst
set_attribute_value tdo_p -name function -value tdo
set_attribute_value vdd* -name function -value power
set_attribute_value vss* -name function -value ground
add_clocks 0 ramclk_p -Period 5ns
set_boundary_scan_port_options ramclk_p -cell_options clock
check_design_rules
```

Create DFT Specification steps:

```
set spec [create_dft_specification]
report_config_data $spec
//display_spec
set_config_value /DftSpecification(cpu_top,gate)/use_rtl_cells On
```

Process DFT Specification and Extract ICL steps:

```
process_dft_specification
extract_icl
```

Create Patterns Specification steps:

```
set pat_spec [create_pat_specification]
report_config_data $pat_spec
```

Process Patterns Specification step:

```
process_patterns_specification
```

Run and Check Test Bench Simulations steps:

```
set_simulation_library_sources -v ./library/adk_complete.v \
                               -v ./library/picdram.v
run_testbench_simulations
check_testbench_simulations
check_testbench_simulations -report_status
```

Related Topics

set_simulation_library_sources [Tessent Shell Reference Manual]

run_testbench_simulations [Tessent Shell Reference Manual]

check_testbench_simulations [Tessent Shell Reference Manual]

MemoryBIST Insertion Before Tap and BSCAN

The example provided in this section covers the initial design insertion for MemoryBIST on a sub_block or physical_block, and finally the insertion of TAP and BoundaryScan at the chip level on a second pass.

If default tsdb_outdir is not used for MemoryBIST, then while inserting TAP and BoundaryScan, use the open_tsdb command to point to the tsdb outdir of where the MemoryBIST is inserted.

Procedure

1. MemoryBIST insertion: Load the Design steps:

    ```
    set_context dft -rtl
    read_cell_library ../library/adk.tcelllib
    set_design_sources -format verilog \
         -y {../data/design/mem ../data/design/rtl} -extension v
    read_verilog ../data/design/rtl/blockA.v
    set_current_design blockA
    report_memory_instances
    ```

2. MemoryBIST insertion: Specify and Verify DFT Requirements steps. MemoryBIST is inserted on a sub_block level.

    ```
    set_design_level sub_block
    set_dft_specification_requirements -memory_test on
    add_clock CLK -period 12ns -label clka
    check_design_rules
    ```

3. MemoryBIST insertion: Create DFT Specification steps:

```
create_dft_specification
report_config_data
report_config_syntax DftSpecification/MemoryBist
```

4. MemoryBIST insertion: Process DFT Specification and Extract ICL steps:

```
process_dft_specification
extract_icl
```

5. MemoryBIST insertion: Create Patterns Specification steps:

```
set spec [create_patterns_specification]
report_config_data $spec
```

6. MemoryBIST insertion: Process Patterns Specification step:

```
process_patterns_specification
```

7. MemoryBIST insertion: Run & Check Test Bench Simulations steps:

```
run_testbench_simulations
check_testbench_simulations
```

8. The prior seven steps need to be repeated for any other sub_blocks or physical_blocks with memories that need MemoryBIST insertion before the design level is changed to chip level.

9. At the chip-level, if you have memories that need MemoryBIST insertion, you will create a DFT Specification as shown below which inserts the MemoryBIST as well as the TAP and BoundaryScan cells.

Note

If the default “tsdb_outdir” is not used for MemoryBIST insertion, then you must use the open_tsdb command to point to the “tsdb outdir” for where the MemoryBIST is inserted while inserting TAP and BoundaryScan cells.

10. BoundaryScan and Tap insertion: Load the Design steps:

```
set_context dft -rtl
read_cell_library ../library/adk.tcelllib
read_verilog ../data/design/rtl/top.v \
             ../data/design/fusebox/*.v

set_current_design top
```

11. BoundaryScan and Tap insertion: Specify and Verify DFT Requirements steps. The TAP signals can be specified here, as shown in Step 2 of TAP, BoundaryScan, and MemoryBIST, or the specified by the DefaultsSpecification as assumed in this example procedure.

```
set_dft_specification_requirements  -boundary_scan on
set_design_level chip
add_clocks clka -period 3ns
add_clocks clkb -period 12ns
set_attribute_value vddq -name function -value power
set_attribute_value vss -name function -value ground
check_design_rules
```

12. BoundaryScan and Tap insertion: Create DFT Specification steps:

```
set spec [create_dft_spec]
cat top.bisr_segment_order
report_conf_data $spec
```

13. BoundaryScan and Tap insertion: Process DFT Specification and Extract ICL steps:

```
process_dft_specification
extract_icl
```

14. BoundaryScan and Tap insertion: Create Patterns Specification steps:

```
set_defaults_value \
   PatternsSpecification/SignOffOptions/ \
    simulate_instruments_in_lower_physical_instances on
set spec [create_patterns_specification]
```

15. BoundaryScan and Tap insertion: Process Patterns Specification step:

```
process_patterns_specification
```

16. BoundaryScan and Tap insertion: Run and Check Test Bench Simulations steps:

```
set_simulation_library_sources -v \
   ../library/verilog/adk.v
run_testbench_simulations
exit
```

Related Topics

run_testbench_simulations [Tessent Shell Reference Manual]

set_simulation_library_sources [Tessent Shell Reference Manual]

Chapter 5
Tap, BoundaryScan and LPCT Type 2 TestKompress

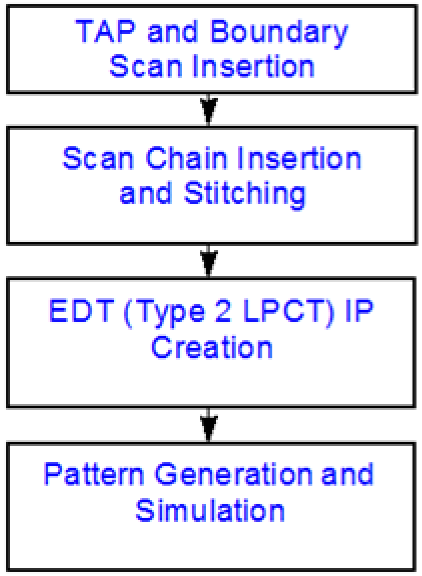
You can use the TAP to control TestKompress, where the TAP's TDI and TDO pins are used as the EDT channel in and channel out pins. The BoundaryScan chain is segmented into smaller Reduced Pin Count Test (RPCT) segments so they can also be part of the logic testing.

Overview

The benefit of implementing TestKompress with a Tap and Type 2 Low Pin Count Test (LPCT) controller and segmenting boundary scan cells into RPCT segments is that the combinational logic that is present between the boundary scan cell and the first tier or level of functional flops is tested during logic testing with TestKompress. An example design is provided that implements this architecture.

The design flow shown in Figure 5-1 shows the high-level overview of the steps that will be implemented.

Figure 5-1. Tap, Boundary Scan, and LPCT Type 2 TK Design Flow



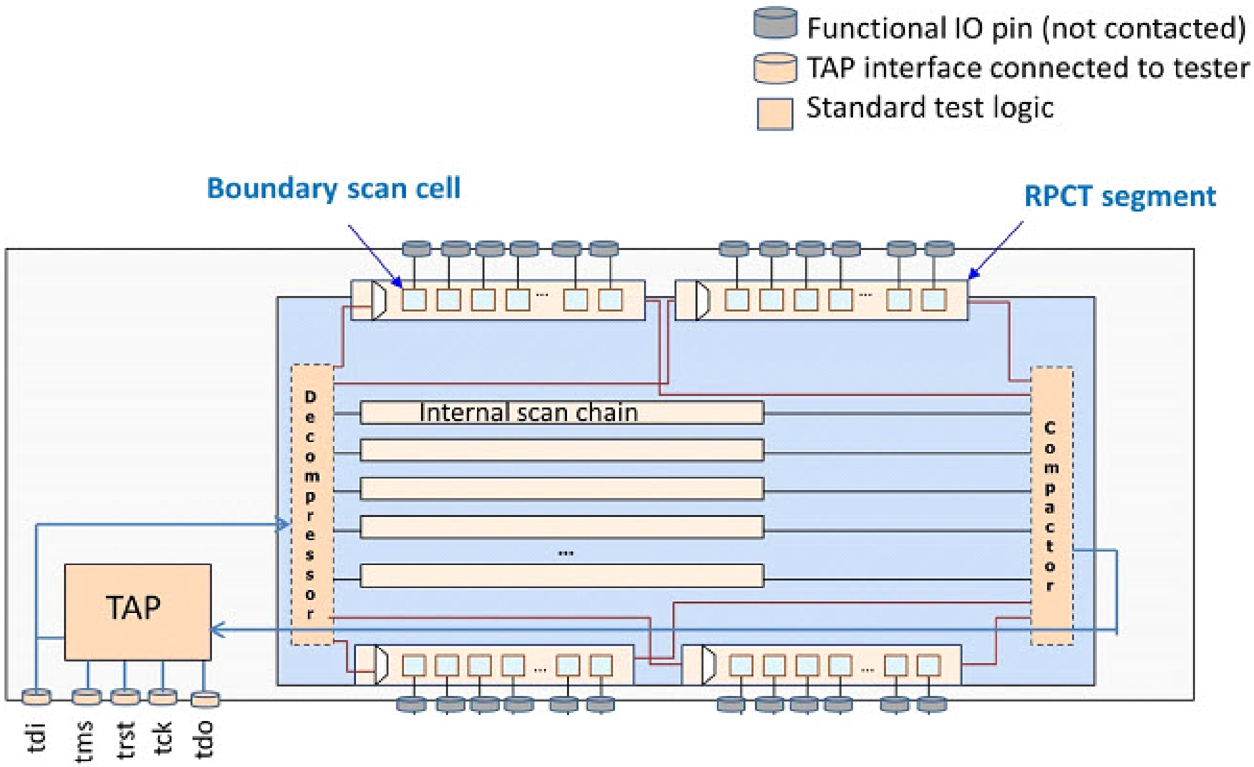
The TAP and boundary scan chains will be inserted first, followed by scan insertion and stitching of the functional design flops. The segmented RPCT boundary scan chains are declared as preexisting scan chains for the scan insertion step.

In the IP creation step, the signals from the TAP controller that are needed to be connected to the Type 2 LPCT controller are specified. The TAP controller's TDI and TDO pins will be used as the EDT channel in and channel out pins respectively.

Finally, patterns are created with TDI and TDO as the single channel for TestKompress. The TAP states are then cycled and validated via verilog simulation to ensure the design's integrity.

Figure 5-2 shows how a single boundary scan chain that is connected between the TDI and TDO of the TAP controller, gets divided into smaller RPCT (Reduced Pin Count Test) segments and how it is connected between the decompressor and the compactor of the TestKompress IP. For further information, refer to "Type 2 LPCT Controller" in the *Tessent TestKompress User's Manual*.

Figure 5-2. Tap, Boundary Scan and Type 2 LPCT TestKompress Implementation



Design Flow for TAP Control of TestKompress

The design steps, considerations and example scripts to implement this architecture are described in the following sections.

TAP and Boundary Scan Insertion

Insert the TAP and Boundary scan chain.

The example script shown provides specific details, which you can use as a guide. The design flow follows the DFT Flow Using Tessent Shell steps.

Procedure

1. Estimate the scan chain length for the functional design cells and apply this length to segment the boundary scan into smaller segments.

   If the scan chain length for the functional design cells is not known, pick a number for the boundary scan chain length to ensure it will not be the longest chain and impact compression.

2. Apply the estimated length to segment the boundary scan chain into smaller segments.

   Use max_segment_length_for_logictest property in the BoundaryScan wrapper to specify how many boundary scan cells are to be in each scan chain. The example script sets the boundary scan chain length to 250 cells.

Examples

This example script inserts a TAP and boundary scan chain following the DFT Flow Using Tessent Shell steps. A TDR is also connected to the TAP controller that provides control for the boundary scan chain to be either a single chain, or segmented and used with logic testing. The use of max_segment_length_for_logictest property is also shown and sets the boundary scan chain length to 250 cells.

If you need to perform custom editing between the insertion of DFT components and the saving of the design, you can define a Tcl procedure with the name “process_dft_specification.post_insertion”. This procedure is called automatically be called during process_dft_specification after all DFT components have been inserted and before the write_design command is invoked.

The example script calls makes use of this technique to insert clock control logic that provides the proper clocking for segmented boundary scan chains during logic testing. The contents of this file are provided in Figure 5-3 and can be customized to meet your design requirements. For more information on this method, refer to the process_dft_specification command description.

```
##Design Loading
set_context dft -no_rtl
read_cell_library ../../libs/adk_complete.tcelllib
read_cell_library ../../libs/memory.lib
read_verilog ../netlist/cpu_top.v
set_current_design cpu_top

##Specify and Verify DFT Requirements
set_dft_specification_requirements -boundary_scan On
set_design_level chip
set_attribute_value tck_p -name function -value tck
set_attribute_value tdi_p -name function -value tdi
set_attribute_value tms_p -name function -value tms
set_attribute_value trst_p -name function -value trst
set_attribute_value tdo_p -name function -value tdo
set_boundary_scan_port_options ramclk_p -cell_options clock
check_design_rules

##Create DFTSpecification
set spec [create_dft_specification]
set_config_value $spec/BoundaryScan/max_segment_length_for_logictest 250
read_config_data -in $spec/IjtagNetwork/HostScanInterface(tap)/ \
    Tap(main)/HostIjtag(1) -from_string {
          Tdr(logic_enable) {
            DataOutPorts +{
              count       : 3;
              port_naming : ltest_enable, bypass_enable, \
                            low_power_enable;
            }
          }
    }
report_config_data $spec
set_config_value /DftSpecification(cpu_top,gate)/use_rtl_cells On

##Custom proc to be run after process_dft_specification
source post_dft_insertion_procedure.tcl

##Process DFTSpecification
process_dft_specification

##Inspect the results to confirm the boundary scan is now segmented
##into smaller chain segments, as described in Step 5 of
## Dividing Boundary Scan for Logic Test.The maximum length was
##set by max_segment_length_for_logictest to 250 flops.
get_instrument_dictionary -list
format_dictionary [get_instrument_dictionary
mentor::jtag_bscan::DftSpecification  logic_test_scan_chains]

##Save the above into a separate file called
##logic_test_scan_chains.dictionary for later reference.
set fp [open logic_test_scan_chains.dictionary w]
puts $fp "set logic_test_scan_chains {"
puts $fp [format_dictionary [get_instrument_dictionary
mentor::jtag_bscan::DftSpecification  logic_test_scan_chains ] ]
puts $fp "}"
close $fp
```

```
##Extract ICL
extract_icl

##Create Patterns Specification
set pat_spec [create_pat_specification]

##Without this the select of the mux on the functional clocks will
##be X, causing simulation failures
set_config_value $pat_spec/AdvancedOptions/ConstantPortSettings/scan_en1
0
##Process Patterns Specification
process_patterns_specification

##Run & Check Test Bench Simulation
set_simulation_library_sources -v ../../libs/adk_complete.v \
    -v ../../libs/pads.v -v ../../libs/ram.v
run_testbench_simulations
check_testbench_simulations
```

Figure 5-3. post_dft_insertion_procedure.tcl Example File (design name = cpu_top)

```
proc process_dft_specification.post_insertion {cpu_top args} {

##The following is needed for controlling the clocks for segmented
##boundary scan chain during logic/scan test

##Creating a scan_en1 port
create_port scan_en1
##Deleting the existing connection on logic_test_enable port
delete_connection cpu_top_gate_tessent_bscan_logical_group_DEF_inst/
logic_test_enable
##Creating connecting from the TDR to control logic_test_enable
##When this TDR is set to 1, then in logic testing mode
create_connection cpu_top_gate_tessent_tdr_logic_enable_inst/ltest_enable
cpu_top_gate_tessent_bscan_logical_group_DEF_inst/logic_test_enable

##The clockbscan, updatebscan and shiftBscan2Edge are intercepted by
##a mux. The other inputs of the clockbscan, updatebscan is tck
##whereas the other input of shiftBscan2Edge is scan_en1.
set tck_tap_point \
    [get_fanins cpu_top_gate_tessent_bscan_interface_I/tck]
intercept_connection cpu_top_gate_tessent_bscan_interface_I/ \
        to_bscan_capture_shift_clock \
    -cell_function_name mux \
    -input2 $tck_tap_point \
    -select cpu_top_gate_tessent_bscan_logical_group_DEF_inst/ \
        logic_test_enable \
    -leaf_instance_prefix clockBscan_
intercept_connection cpu_top_gate_tessent_bscan_interface_I/ \
        to_bscan_update_clock \
    -cell_function_name mux \
    -input2 $tck_tap_point \
    -select cpu_top_gate_tessent_bscan_logical_group_DEF_inst/ \
        logic_test_enable \
    -leaf_instance_prefix updateBscan_
intercept_connection cpu_top_gate_tessent_bscan_interface_I/ \
        to_bscan_shift_en \
    -cell_function_name mux \
    -input2 scan_en1 \
    -select cpu_top_gate_tessent_bscan_logical_group_DEF_inst/ \
        logic_test_enable \
    -leaf_instance_prefix shiftBscan2Edge_

##Do not need muxes to choose between functional clock and tck when
##OCC is present.
##Adding muxes to all the functional clocks in the design
intercept_connection /inpad0/C -cell_function_name mux \
    -input2 $tck_tap_point -select scan_en1 \
    -leaf_instance_prefix clk1_p_
intercept_connection /inpad1/C -cell_function_name mux \
    -input2 $tck_tap_point -select scan_en1 \
    -leaf_instance_prefix clk2_p_
intercept_connection /inpad2/C -cell_function_name mux \
    -input2 $tck_tap_point -select scan_en1 \
    -leaf_instance_prefix clk3_p_
intercept_connection /inpad3/C -cell_function_name mux \
    -input2 $tck_tap_point -select scan_en1 \
    -leaf_instance_prefix clk4_p_
intercept_connection /inpad4/C -cell_function_name mux \
```

```
-input2 $tck_tap_point -select scan_en1 \
-leaf_instance_prefix ramclk_p_
```

Related Topics

TAP and Boundary Scan Insertion

add_scan_chains [Tessent Shell Reference Manual]

read_icl [Tessent Shell Reference Manual]

Scan Chain Insertion and Stitching

“Inserting EDT Logic During Synthesis” [Tessent TestKompress User's Manual]

“Low Pin Count Test Controller” [Tessent TestKompress User's Manual]

"Generating and Verifying Test Patterns" [Tessent TestKompress User's Manual]

EDT (Type 2 LPCT) IP Creation

set_lpct_pins [Tessent Shell Reference Manual]

Scan Chain Insertion and Stitching

The second step is to insert scan chains for the rest of the functional design logic.

Prerequisites

- Complete TAP and Boundary Scan Insertion steps.

Procedure

1. Use the steps outlined in the example script provided as a guide to complete this stage of the design flow. Supporting scripts for the example are provided in Figure 5-4 through Figure 5-7.

2. Use the add_scan_chains -internal property to declare the boundary scan chains as preexisting, as shown by the red highlighted lines in the example script.

   The icl file for the design created in TAP and Boundary Scan Insertion will be read in using read_icl, and synthesized along with the functional design scan chains. A PDL file is also created and read in for the TAP controller and the TDR that is used to enable a segmented boundary scan chain.

Examples

```
set_context dft -scan
read_cell_library ../library/adk_complete.tcelllib
read_verilog ../from_step1/cpu_top.v_full
read_verilog ../from_step1/synthesized/boundary_scan_cells.v
read_verilog ../from_step1/synthesized/tessent_tap_main.v

##Reading icls and pdls before set_current_design
read_icl ../from_step1/tsdb_outdir/dft_inserted_designs/ \
    cpu_top_gate.dft_inserted_design/cpu_top.icl
dofile ./dofiles/cpu_top_gate_tessent_tap_main.pdl
dofile ./dofiles/cpu_top_gate_tessent_tdr_logic_enable.pdl

set_current_design cpu_top

## Add clock definitions
add_clocks 0 tck_p
add_clocks 0 clk1_p
add_clocks 0 clk2_p
add_clocks 0 clk3_p
add_clocks 0 clk4_p
add_clocks 0 ramclk_p


add_input_constraints trst_p -C1
add_input_constraints tms_p -C0

##Scan Enable scan_en1 as preExisting
set_scan_enable scan_en1
add_input_constraints scan_en1 -C0
add_nonscan_instances cpu_top_gate_tessent_tap_main_inst
add_nonscan_instances cpu_top_gate_tessent_tdr_logic_enable_inst

## Reading in preexisting bscan chains
dofile ./dofiles/e_chains.dofile

set_system_mode analysis
insert_test_logic -max_length 250 -clock merge \
    -edge merge -new_scan_po
report_test_logic
report_scan_chains
write_design -output generated/cpu_top_scan.v -replace
write_atpg_setup generated/cpu_scan -replace
```

Figure 5-4. Example cpu_top_gate_tessent_tap_main.pdl File

```
## Called from the Test_Setup procedure in Figure 5-7.
iProcsForModule cpu_top_gate_tessent_tap_main
iProc tap_main_setup { } {
//SelectJtagOutput is 0 so the value from the core is captured into
//the output boundary scan cells.
    iWrite select_jtag_output 0b0
//SelectJtagInput is 1 so the value from the boundary scan register
//or cell is captured into the first level of functional flops.
    iWrite select_jtag_input 0b1
iApply
}
```

Figure 5-5. Example cpu_top_gate_tessent_tdr_logic_enable.pdl File

```
## Called from the Test_Setup procedure in Figure 5-7.
iProcsForModule cpu_top_gate_tessent_tdr_logic_enable
iProc logic_enable { } {
    iWrite tdr[2:0] 0b100
iApply
}
```

Figure 5-6. Example e_chains.dofile

```
add_scan_groups group1 existing_chains.testproc
add_scan_chains -internal chain0 group1 \
    cpu_top_gate_tessent_bscan_logical_group_DEF_inst/\
    clk1_p_logic_scanin \
    cpu_top_gate_tessent_bscan_logical_group_DEF_inst/\
    CELL249_BSCAN_SO
add_scan_chains -internal chain1 group1 \
    cpu_top_gate_tessent_bscan_logical_group_DEF_inst/\
    expdout_p_6_logic_scanin \
    cpu_top_gate_tessent_bscan_logical_group_DEF_inst/\
    CELL0_BSCAN_SO
```

Figure 5-7. Example existing_chains.testproc File

```
set time scale 1ns;
alias int_clocks = clk1_p, clk2_p, clk3_p, clk4_p;
timeplate global =
  force_pi 0;
  measure_po 15;
  pulse tck_p 25 50; // tck_p
  pulse clk1_p 25 50; //
  pulse clk2_p 25 50; //
  pulse clk3_p 25 50; //
  pulse clk4_p 25 50; //
  period 100;
end; // timeplate global


procedure Test_Setup = // Describes the setup phase to be applied
                      //once at the beginning of test
timeplate global;
iCall cpu_top_gate_tessent_tap_main_inst.tap_main_setup ;
iCall cpu_top_gate_tessent_tdr_logic_enable_inst.logic_enable ;
end; // procedure Test_Setup }}}


procedure Shift = // Describes one clock cycle of shift.
  timeplate global;
  cycle =
    force scan_en1      1;
    force trst_p       1; // trst_p
    pulse tck_p;          // tck_p
    force tms_p        0; // tms_p
    force_sci;
    measure_sco;
  end;
end; // procedure Shift


procedure Load_Unload =//Proceedure to load/unload the scan chains.
timeplate global;
   cycle =
      force trst_p        1; // trst_p
      force tck_p         0;  // tck_p
      force tdi_p         0;  // tdi_p
      force int_clocks    0;
      force tms_p         0;  // tms_p
end;
  apply Shift 250; //Auto adjusted by FastScan to match the longest
                   //scan chain
end; // procedure Load_Unload
```

EDT (Type 2 LPCT) IP Creation

The third design flow step for implementing TAP control of TestKompress is to create the Embedded Deterministic Testing (EDT) compression logic IP .

Prerequisites

- Complete TAP and Boundary Scan Insertion steps.
- Complete Scan Chain Insertion and Stitching steps.

Procedure

1. Follow the steps outlined in the example script provided below as a guide to complete this stage of the design flow. The example shows the settings needed for LPCT Type 2 IP creation.

   Caution

   You must specify the test_logic_reset signal from the Tessent TAP as active low because it is driven as active low. Failure to do this will result in failing simulations and a condition that is difficult to debug.

2. Synthesize the EDT IP logic RTL that was generated into core netlist Verilog gates using the Synopsis Design Complier script generated during the IP creation. For more information, see "Inserting EDT Logic During Synthesis" in the *Tessent TestKompress User's Manual*.

Examples

```
##Using a Type 2 LPCT Controller
set_lpct_controller On -TAp_controller_interface On \
    -Generate_scan_enable On
set_lpct_controller -shift_control clock

##LPCT Pin connections from LPCT controller
set_lpct_pins output_scan_en scan_en1
set_lpct_pins TEST_Clock_connection \
    clk1_p_mux/A1 clk2_p_mux/A1 clk3_p_mux/A1 clk4_p_mux/A1 \
    clockBscan_mux/A1 updateBscan_mux/A1

##Use the TAP's TDI and TDO as the external channel to drive EDT
set_edt_options -location internal -channels 1
set_edt_pins input_channel 1 tdi_p tdi_i/C
set_edt_pins output_channel 1 tdo_p tdo_i/I

##LPCT Pin connections to LPCT controller pins
set_lpct_pins reset - cpu_top_gate_tessent_tap_main_inst/\
   test_logic_reset -active low
set_lpct_pins capture_dr - cpu_top_gate_tessent_tap_main_inst/\
   capture_dr_en
set_lpct_pins shift_dr - cpu_top_gate_tessent_tap_main_inst/\
   shift_dr_en
set_lpct_pins update_dr - cpu_top_gate_tessent_tap_main_inst/\
   update_dr_en
set_lpct_pins test_mode - cpu_top_gate_tessent_tdr_logic_enable_inst/\
   ltest_enable
set_lpct_pins tms tms_p tms_i/C
set_lpct_pins clock tck_p tck_i/C
```

Related Topics

"Low Pin Count Test Controller" [Tessent TestKompress User's Manual]

"Inserting EDT Logic During Synthesis" [Tessent TestKompress User's Manual]

Pattern Generation and Simulation

The final step for implementing TAP control of TestKompress is generating the compressed test patterns and verifying the integrity of the inserted test circuitry and scan chains through simulation.

Guidelines and tips for successful implementation of this step for implementing this architecture are presented. For additional information on generating and verifying EDT test patterns, refer to "Generating and Verifying Test Patterns" in the *Tessent TestKompress User's Manual*.

For the architecture implementation described in this chapter, make sure that during the EDT (Type 2 LPCT) IP Creation phase, the set_lpct_pins TEST_clock_connection property is used as described in the example for that section. If there are no On-Chip-Clock (OCC) generators in the design, then insert TCK for all the functional clocks. The TCK clock needs to be controlled from the output of the LPCT controller.

Note
The following needs to be understood if you are planning on inserting OCC in your design.

The OCC generator is inserted only for functional clocks. The TCK clock will have a clock gating function added and will not receive an OCC generator.

There are two modes in which patterns can be generated during pattern generation - a SLOW and FAST capture mode. During SLOW capture mode, the slow speed clock is used for both shift and capture during "stuck-at" testing. In this mode (SelectJtagOutput = 0 and SelectJtagInput =1), the logic states into and out of the core are being sourced and captured by the boundary scan registers. However, during the FAST capture mode (SelectJtagOutput = 1 and SelectJtagInput = 1), the logic state from the core side is not captured in the output boundary scan cells. If not setup correctly, what ATPG predicts happens will not match simulation results, creating errors that are very difficult to diagnose.

In some designs, it may be that the "at-speed" coverage is very low due to the fact that any interaction between the boundary scan cells and the core is not covered. If the boundary scan cells can be synthesized to the same frequency as the fast capture clock, then during transition testing the interaction can be tested "at-speed". Typically, this is not desired because during boundary scan test the clock used is TCK, which is a slow speed clock.

Appendix A
Tessent Core Description

This section describes the configuration data syntax used to describe the following macro module types: core and boundary scan segments.

This appendix uses the following syntax conventions when documenting wrappers and properties used in the library descriptions.

Table A-1. Conventions for Command Line Syntax

Convention	Example	Usage
UPPercase	-STatic	Required argument letters are in uppercase; in most cases, you may omit lowercase letters when entering literal arguments, and you need not enter in uppercase. Arguments are normally case insensitive.
Boldface	set_fault_mode **<u>Uncollapsed</u>** | **Collapsed**	A boldface font indicates a required argument.
[ ]	exit [-force]	Square brackets enclose optional arguments. Do not enter the brackets.
Italic	dofile ***filename***	An italic font indicates a user-supplied argument.
{ }	add_fault_sites {**-ALl** | **-UNDEFINED_Cells** } [-VERBose]	Braces enclose arguments to show grouping. Do not enter the braces.
|	add_fault_sites {**-ALl** | **-UNDEFINED_Cells** } [-VERBose]	The vertical bar indicates an either/or choice between items. Do not include the bar in the command.
Underline	set_dofile_abort **<u>ON</u>** | **OFf**	An underlined item indicates either the default argument or the default value of an argument.
…	add_clocks ***off_state*** ***primary_input_pin***… [-Internal]	An ellipsis follows an argument that may appear more than once. Do not include the ellipsis when entering commands.

Table A-2. Syntax Conventions for Configuration Files

Convention	Example	Usage
Italic	***scan_in :*** *port_pin_name*;	An italic font indicates a user-supplied value.

Table A-2. Syntax Conventions for Configuration Files (cont.)

Convention	Example	Usage
Underline	wgl_type : generic | lsi;	An underlined item indicates the default value.
|	logic_level : both | high | low;	The vertical bar separates a list of values from which you must choose one. Do not include the bar in the configuration file.
...	port_naming : *port_naming*, ...;	Ellipses indicate a repeatable value. The comment "// repeatable" also indicates a repeatable value.
//	// default: ijtag_so	The double slash indicates the text immediately following is a comment and tells the tool to ignore the text.

Core

In Tessent Shell, descriptions of core elements, such as the memory library, the boundary scan information, or the fuse box interface, are presented to the tool in form of TCD files (Tessent Core Description files). After loading, they are hierarchically organized under the 'Core' root entry, which is unique for a given module name.

Usage

```
Core(module_name) {
  Memory {
  }
  BoundaryScan {
  }
  FuseBoxInterface  {
  }
}
```

Description

The Core wrapper collects all TCD data read into the tool. Such descriptions are automatically read in during module matching. See the set_design_sources -format tcd_memory command description for information about where they are looked for. See the read_core_descriptions command description to learn how to read them in explicitly.

You can also report on the loaded TCD information. You do this using the report_config_data command. An example is "report_config_data Core(ModuleName)/Memory -partition tcd", which report the contents of the Memory entries under Core. To see the supported syntax, use the report_config_syntax command, for example "report_config_syntax Core/Memory".

Arguments

- *module_name*

  The name of the module, equivalent of the current design module name. You don't need to specify this when loading a memory TCD file. The tool with auto-generate and auto-configure the Core-level wrapper for you.

Related Topics

set_design_sources [Tessent Shell Reference Manual]

read_core_descriptions [Tessent Shell Reference Manual]

report_config_data [Tessent Shell Reference Manual]

report_config_syntax [Tessent Shell Reference Manual]

set_module_matching_options [Tessent Shell Reference Manual]

BoundaryScan

Specifies the embedded boundary scan chain already implemented into the design module_name.

Usage

```
Core(module_name) {
  BoundaryScan {
    Interface {
    }
    CustomBsdlCellInfo {
      cell_type_id : cell_info_description; // repeatable
    }
    ExternalPort(port_name) {
    }
    Cell(id) {
    }
    NonScannableInstances {
      instance_name ; // repeatable
    }
  }
}
```

Description

Describes the pads and boundary-scan cells contents of a module module_name already instantiated in the design. Such descriptions are automatically read in during module matching. See the set_design_sources -format tcd_bscan command description for information about where they are looked for. See the read_core_descriptions command description to learn how to read them in explicitly. See the set_module_matching_options command description for information about the name matching process. Note that the legacy LogicVision .lvbscan format is supported natively and is automatically translated into this format when read.

To see the content of a read-in Core(ModuleName)/BoundaryScan, use the “report_config_data Core(ModuleName)/BoundaryScan -partition tcd” command.

To see the supported syntax, use the “report_config_data [get_config_value Core/ BoundaryScan -partition meta:tcd -object]” command.

Arguments

None.

Interface

This section describes the common ports of the boundary scan hardware of the module module_name.

Usage

```
Core(module_name) {
  BoundaryScan {
    Interface {
      select                       : port_name ;
      reset                        : port_name ;
      force_disable                : port_name ;
      select_jtag_input            : port_name ;
      select_jtag_output           : port_name ;
      select_jtag_enable           : port_name ;
      capture_shift_clock          : port_name ;
      capture_shift_clock_inv      : port_name ;
      bscan_clock                  : port_name ;
      update_clock                 : port_name ;

      capture_en                   : port_name ;
      shift_en                     : port_name ;
      update_en                    : port_name ;

      scan_in                      : port_name ;
      scan_out                     : port_name ;
      scan_out_launch_edge         : posedge | negedge ;

      ac_init_clock0               : port_name ;
      ac_init_clock1               : port_name ;
      ac_signal                    : port_name ;
      ac_mode_enable               : port_name ;
    }
  }
}
```

Description

This section describes the common ports of the boundary scan hardware of the module module_name. Theses ports usually connect to a higher level boundary scan interface block or to a TAP.

Arguments

- select : *port_name*

  This signal is used to enable the boundary-scan register logic in the module. If this signal is active the capture, shift and update enable signals effect the boundary scan register. The signal is furthermore used to gate the clocks bscan_clock, capture_shift_clock and capture_shift_clock_inv. The signal is optional and it assumed to be active high.

- reset : *port_name*

  This signal is used to reset the boundary scan register logic in the module. The signal is active low.

- force_disable : *port_name*

  This disables all pad cell drivers. The external ports will show a Z signal. This is limited to those pad cells that can be disabled. The signal overrides the settings from the enable boundary scan cells. The signal is active high. The signal is needed to support the HIGHZ instruction.

- select_jtag_input : *port_name*

  This is the select signal of the SJI multiplexer that switches between the pad cells from_pad port and the output of the boundary scan cell. The output of the multiplexer is connected to the core. When the signal is high it selects the boundary scan cell output. This signal is mandatory, in case the module contains input boundary scan cells.

- select_jtag_output : *port_name*

  This is the select signal of the SJO multiplexer that switches between the core signal and the output of the boundary scan cell. The output of the mux is connected to the pad cells to_pad port. When the signal is high it selects the boundary scan cell output. The signal is mandatory, in case the module contains output or bidirectional boundary scan cells.

- select_jtag_enable : *port_name*

  This is the select signal of the SJI multiplexer that switches between the functional pad enable and the output of the enable boundary scan cell. The output of the multiplexer is connected to the pad enable port. When this signal is high it selects the output of the boundary scan cell. This entry is optional. The select of the SJE multiplexer can be connected to select_jtag_output instead of this dedicated signal, when this signal is not specified.

- capture_shift_clock: *port_name*

  This is a gated version of the test clock. The clock should be active during shift and capture cycles. The inactive state of the clock is the low state. You need to specify either bscan_clock or capture_shift_clock or capture_shift_clock_inv.

- capture_shift_clock_inv : *port_name*

  Inverted version of capture_shift_clock. This is the legacy clock timing that is used in ETAssemble.You need to specify either bscan_clock or capture_shift_clock or capture_shift_clock_inv.

- bscan_clock : *port_name*

  Clock that needs to be continuously running during the boundary scan test. This is usually the test clock or a gated version of the test clock that is disabled outside of the boundary scan test. You need to specify either bscan_clock or capture_shift_clock or capture_shift_clock_inv.

- update_clock : *port_name*

  A gated clock that pulses the update register in the boundary scan cell. The disabled stated of the clock is the low state. You need to specify this clock when using the capture_shift_clock or the capture_shift_clock_inv clock. You need to specify either this entry or update_en.

- capture_en : *port_name*

  When this signal is high the boundary scan cells will capture the value from either pad, the core or the update register dependent on the type of the boundary scan cell. You need to specify this signal when you use bscan_clock. The signal is active high.

- shift_en : *port_name*

  This is the shift enable of the boundary scan register. It is the select of the scan multiplexer inside the boundary scan cells and a high value here will connect the boundary scan cell to a scan chain. This entry is mandatory.

- update_en : *port_name*

  When this signal is high the update elements in the boundary scan cells take over the value of the boundary scan register. This entry is needed when you use bscan_clock. You need to specify either this entry or update_clock.

- scan_in : *port_name*

  This is the scan input of the boundary scan segment inside the module. It is used to shift in data from the TAP or a boundary scan interface block or from other parts of the boundary scan register.

- scan_out : *port_name*

  This is the scan output of the boundary scan segment inside the module. It is used to shift out data to the TAP or a boundary scan interface block or to other parts of the boundary scan register.

- scan_out_launch_edge : posedge | negedge

  This entry is only valid, if scan_out is defined. This specified the edge on which the data on the scan output is valid.

- ac_init_clock0 : *port_name*

  This clock is used to trigger the initialization of the test receiver in AC input and AC bi-directional pad cells. The off state of this clock is low.

- ac_init_clock1 : *port_name*

  This clock is used to trigger the initialization of the test receiver in AC input and AC bidirectional pad cells.. The off state of this clock is high.

- ac_signal : *port_name*

  This signal gives the pulse(s) for the extest pulse or train according the the IEEE 1149.6 standard. This signal is mandatory when the module contains AC output or inout cells.

- ac_mode_enable: *port_name*

  This enables the AC functionality in the described module. This signal enables the input AC functionality and the output functionality for those ports that don't have an AC select cell.

CustomBsdlCellInfo

This wrapper defines custom boundary scan cell types and describes their capture behavior.

Usage

```
Core(module_name) {
  BoundaryScan {
    CustomBsdlCellInfo {
      cell_type_id : cell_info_description ; // repeatable
    }
  }
}
```

Description

The CustomBsdlCellInfo wrapper is used to define custom boundary scan cell types and describe their capture behavior. The CustomBsdlCellInfo descriptions will be saved to the TS_BSCAN_CELLS BSDL package file along side the BSDL file in the Tessent Shell Data Base (TSDB) and be used when generating IO test patterns.

Standard 1149.1 built-in cell types, such as those prefixed with BC_ or AC_, as well as custom cell types, are specified in the BoundaryScan/Cell wrapper to describe boundary scan cells in the design. The custom boundary scan cell types specified in the BoundaryScan/Cell wrapper must also be described in the CustomBsdlCellInfo wrapper.

Arguments

- *cell_type_id* : *cell_info_description* ;

  A repeatable label string and complex string pair that defines and describes the custom cell type.

  The cell_type_id string value is user-defined, and is specified in the BoundaryScan/Cell/bsdl_cell_type property to identify the custom cell type for a boundary scan cell.

  The cell_info_description syntax must exactly match the CELL_INFO syntax as defined in the IEEE 1149.1 standard, “Annex B.10 User-supplied BSDL packages” section. Note that the cell_info_description must be enclosed in quotes because the description contains spaces.

Examples

The following example shows how to define, and use, a custom boundary scan cell type that is a variation of the standard BC_7 cell type.

The standard BC_7 cell type is described by the following CELL_INFO in the IEEE 1149.1 standard:

```
constant BC_7 : CELL_INFO :=
 ((BIDIR_IN, EXTEST, PI), (BIDIR_OUT, EXTEST, PO),
  (BIDIR_IN, SAMPLE, PI), (BIDIR_OUT, SAMPLE, PI),
  (BIDIR_IN, INTEST, UPD), (BIDIR_OUT, INTEST, PI));
```

We want to create a custom cell type that differs in the highlighted INTEST behavior. For the standard cell, this portion can be read as “for this cell used as a bidirectional cell acting as an input (BIDIR_IN) while INTEST is in effect, the capture flip-flop loads the value of the Update flip-flop (or latch) data (UPD) during Capture-DR controller state”.

The custom cell to be defined has the following behavior for the highlighted section:

```
(BIDIR_IN, INTEST, X)
```

This indicates an unknown value is loaded rather than the UPD value, as is done in the standard cell type.

The example below shows the definition and use of the described custom cell type:

```
Core(core) {
  BoundaryScan {
    CustomBsdlCellInfo {
      my_BC_7 : "((BIDIR_IN, EXTEST, PI), (BIDIR_OUT, EXTEST, PO),
                  (BIDIR_IN, SAMPLE, PI), (BIDIR_OUT, SAMPLE, PI),
                  (BIDIR_IN, INTEST, X),  (BIDIR_OUT, INTEST, PI))";
    }
    Cell(my_bidir){
      function : bidir ;
      bsdl_cell_type : my_BC_7 ;
      ...
    }
  ...
  }
}
```

ExternalPort

This section describes the properties of a port that will be connected directly to a top level port and the connected pad cell.

Usage

```
Core(module_name) {
  BoundaryScan(module_name) {
    ExternalPort(port_name) {

      differential_inverse_of     : port_name;
      differential_type           : voltage | current;

      buffer_type                 : three_state | two_state |
                                    low_only | high_only;
      highz_during_force_disable : on | off | auto;
      pull_resistor               : high | low | none;

      control_cell                : cell_id;


      ac_hp_time                  : time;
      ac_lp_time                  : time;
      ac_hp_location              : on_chip | off_chip;

      auxiliary_output            : port_name;
      auxiliary_output_enable     : port_name;
      auxiliary_input             : port_name;
      auxiliary_input_enable      : port_name;
    }
  }
}
```

Description

This section describes the properties of a port port_name on the tcd_bscan module that will be directly connected to a top level port. The pad cell is inside the tcd_bscan segment and this wrapper describes the properties of the pad cell.

Arguments

- differential_inverse_of : *port_name*

  This entry is used for the associated port of a differential pair. The port_name refers to the ExternalPort(port_name) wrapper of the representative port. This entry is exclusive with all other entries. The settings for the differential pair need to be done in the wrapper of the representative port.

- differential_type : voltage | current

  Specifies the nature of the differential pair. If this is set to voltage it means that the signals are similar to the other logic signals. In case of current the differential pair is directly accessible from the tester and the tester needs to determine how to deal with this differential port. If this entry is specified there needs to be a ExternalPort() wrapper referring to this wrapper with the differential_inverse_of property to this port.

- buffer_type : three_state | two_state | low_only | high_only

  This specifies the values the pad driver can drive and if the driver can be disabled. See the table for an overview of the combinations.

Table A-3. buffer types and their available state

buffer_type	can drive a 0	can drive a 1	can be disabled to Z
three_state	yes	yes	yes
two_state	yes	yes	no
low_only	yes	no	yes
high_only	no	yes	yes

A three_state port can drive 0 and 1 values and is enables and disabled by an extra enable cell. A two_state pot can drive a 0 and a 1, but cannot be disabled. Note that you will not be able to implement a fully IEEE 1149.1 HIGHZ instruction, if you use such a driver in your design, because this buffer type lacks the needed ability to disable the pad driver. The types low_only and high_only are asymmetric drivers like open emitter and open collector driver. The enable signal of a driver can be overridden by the force disable signal that switches off all driver except the two_state ones. This entry is only valid for output or inout ports.

- highz_during_force_disable : on | off | <u>auto</u>

  Specifies if the pad driver can be disabled with the force disable signal. This entry is only valid for output or inout ports. In case of auto it is assumed that driver of the buffer_type three_state, low_only and high_only can be disabled.

- pull_resistor : high | low | <u>none</u>

  Specifies if a pull resistor is present in the pad.

Table A-4. Valid combinations of the pull_resistor and buffer_type

pull_resistor	buffer_type	bsdl disable result
none	three_state	Z
none	two_state	-
none	low_only	weak1
none	high_only	weak0
high	low_only	pull1
low	high_only	pull0

The table above shows which combinations are valid. This entry is only valid for output and inout ports.

- control_cell : *cell_id*

  This refers to a Cell(*cell_id*) wrapper that contains the description of the control cell that enables and disables the pad. This entry is only valid for output and inout ports.

- ac_hp_time : *time*

  This describes the AC high pass behavior of the pad according to the IEEE 1149.6 standard.

- ac_lp_time : *time*

  This describes the AC low pass behavior of the pad according to the IEEE 1149.6 standard.

- ac_hp_location : on_chip | off_chip

  This described if the AC high pass is on the chip or needs to be added on the outside.

- auxiliary_output : *port_name*

  This specifies a port that can be multiplexed with the data from the core or the data bscan cell. The output of the multiplexer is connected to the pad of the external port. The external port needs to be an output or inout port. You need to specify also the auxiliary_output_enable entry.

- auxiliary_output_enable : *port_name*

  This is the multiplexer enable for the auxiliary_output above. You need to specify both entries together.

- auxiliary_input : *port_name*

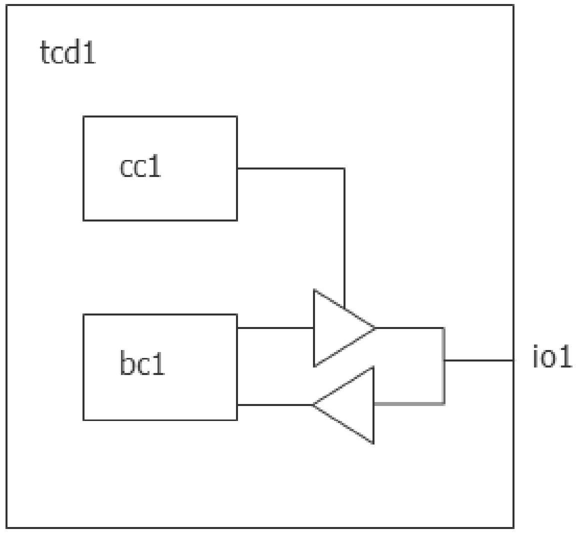
  This specifies a port that gets the value from the pad of the external port, if enabled with the auxiliary_input_enable. The external port needs to be an input or inout port. You need to specify also the auxiliary_input_enable entry.

- auxiliary_input_enable : *port_name*

  This is the enable for the auxiliary_input above. You need to specify both entries together.

Examples

Figure A-1. Bidirectional Pad with Data and Enable Cell



The core description of the bidirectional pad with boundary scan and enable cell :

```
Core(tcd1) {
   BoundaryScan {
      Interface { // not shown in the figure
         force_disable          : force_disable1;
         select_jtag_input      : select_jtag_input1;
         select_jtag_output     : select_jtag_output1;
         bscan_clock            : bscan_clock1;
         capture_en             : capture_enable1;
         shift_en               : shift_enable1;
         update_en              : update_enable1;
         scan_in                : scan_input1;
         scan_out               : scan_output1;
         scan_out_launch_edge : negedge;
      }
      ExternalPort(io1) {
         buffer_type            : three_state;
         control_cell           : cc1;
      }
      Cell(cc1) {
         function               : control;
         bsdl_cell_type         : BC_2;
         control_enable_value : 1;
         safe_value             : 0;
      }
      Cell(bc1) {
         function               : bidir;
         bsdl_cell_type         : BC_7;
         external_port          : io1;
         safe_value             : 0;
      }
   } }
```

Cell

This section describes a boundary scan cell.

Usage

```
Core(module_name) {
  BoundaryScan {
    Cell(id) {
      function                : output | input | bidir |
                                control | control_reset | internal |
                                observe_only | clock | ac_select;
      bsdl_cell_type          : cell_type_id;
      external_port           : port_name;
      control_enable_value    : 0 | 1;
      ac_select_cell          : cell_id;
      ac_select_port          : port_name;
      safe_value              : 0 | 1 | x;
      ac_type                 : on | off;
    }
  }
}
```

Description

This section describes a boundary scan cell.

Arguments

- function : output | input | bidir | control | control_reset | internal | observe_only | clock | ac_select

  The following table shows the function of each boundary scan cell.

Table A-5. Cell functions

function	description	BSDL function
output	A data cell that drives an output external port. The cell may also be able to observe the value from the core.	OUTPUT2 or OUTPUT3
input	A data cell that observes an input external port. The cell may also be able to drive the signal to the core.	INPUT
bidir	A data cell for an inout external port.	BIDIR
control	A control cell that enables and disables a group of output and/or inout external ports.	CONTROL
control_reset	Same as control, but forced to the disabled state during the Test-Logic_reset TAP state.	CONTROLR

Table A-5. Cell functions (cont.)

function	description	BSDL function
internal	Cell not associated with an external port.	INTERNAL
observe_only	A cell that captures values from input, output or inout external ports.	OBSERVE_ONLY
clock	A cell that observes the state of a clock external port.	CLOCK
ac_select	A cell that enables the AC functionality on a group of output and/or inout external ports.	INTERNAL

- bsdl_cell_type : *cell_type_id*

  This is the entry that will be used to describe the cell in the BSDL file. This can either be a build in type like the BC_# or AC_# or a custom cell type described in a package file.

- external_port : *port_name*

  This refers to an ExternalPort(port_name) wrapper. This cell is a data boundary scan cell of the external port.

- control_enable_value : 0 | 1

  This specifies the enable value for cells with the function control, control_reset and ac_select. For those cells the entry is mandatory.

- ac_select_cell : *cell_id*

  This refers to another Cell(cell_id) wrapper. The other cell is an AC select cell and will enable and disable the AC functionality of this cell.

- safe_value : 0 | 1 | x

  This specified the save value of the cell. The safe value will be used in the BSDL file.

- ac_type : on | off

  Specifying this to on implies that this is an AC data boundary scan cell. It needs to have the function input, output, bidir, or observe_only. With the function property set to input or observe_only, the cell observes the IEEE 1169.9 test receiver on the port described by the external_port property. With the function property set to output or bidir, the cell determines the polarity of the IEEE 1149.6 pulse or train on the port described by the external_port property.

NonScannableInstances

This wrapper defines BoundaryScan instances that will have the is_non_scannable attribute set to true.

Usage

```
Core(module_name) {
  BoundaryScan {
    NonScannableInstances {
      instance_name ; // repeatable
    }
  }
}
```

Description

The NonScannableInstances wrapper specifies the BoundaryScan instances that will have the is_non_scannable attribute set to true. If the NonScannableInstances wrapper is not present, all instances in *module_name* will be set as non-scannable. If the NonScannableInstances wrapper is present, but empty, all instances in *module_name* are scannable. If the NonScannableInstances wrapper is present and contains instances, those listed instances that are within the module *module_name* are set as non-scannable.

Arguments

- *instance_name* ;

  A repeatable string that specifies a BoundaryScan instance within the design *module_name* that will have the is_non_scannable attribute set to true.

Examples

The non-scannable BoundaryScan instances can be queried by using two attributes:

- is_non_scannable = true
- is_non_scannable_reason = “imported_from_bscan”

Using these attributes, the following command will show the BoundaryScan instances that are non-scannable:

```
get_instances -filter
{is_non_scannable&&is_non_scannable_reason=="imported_from_bscan"}
```

Note that the attributes will apply to all child instances.

Appendix B
Support For AC Pins (IEEE 1149.6)

This section describes the commands and templates Tessent uses to create device circuitry compliant with the IEEE 1149.6 standard for differential or capacitively coupled pins.

Specifying AC Pins in Your Design

AC boundary scan is inserted in your design when AC pins are present. This can be controlled via several Tessent Shell commands.

The set_dft_specification_requirements command can be used to include or exclude AC boundary scan in your design. The optional -ac_boundary_scan switch on this command controls whether AC boundary scan will be inserted. When this switch is set to “off,” AC boundary scan (IEEE 1149.6) is not implemented. If the command is not used, or if -ac_boundary_scan is set to “auto” (the default), AC boundary scan will be inserted.

AC Pins in Configuration Specifications

The wrappers and properties of Tessent's configuration specifications will affect how AC boundary scan is implemented in your design.

The DftConfiguration and PatternsSpecification wrappers can be used to control how IEEE 1149.6 AC pins and test patterns are handled in your design. This method provides a means of defining and maintaining alternate configurations from a variety of sources.

For a full description of how to configure test structures in your design using the configuration data syntax, see the Configuration-Based Specification chapter in the Tessent Shell Reference Manual.

AC Pins in the DftSpecification Wrapper

Several wrappers and properties in the DftSpecification wrapper affect the way AC pins are managed in your design.

For a full description of this wrapper, see the DftSpecification section of the Tessent Shell Reference Manual.

The Interface wrapper inside the EmbeddedBoundaryScan wrapper includes the following arguments:

- **ac_init_clock0** — defines the name of the ac_init_clock0 port. Default: "bscan_ac_init_clk0".
- **ac_init_clock1** — defines the name of the ac_init_clock1 port. Default: "bscan_ac_init_clk1".
- **ac_signal** — defines the name of the ac_signal port. Default: "bscan_ac_signal".
- **ac_mode_en** — defines the name of the ac_mode_en port. Default: "bscan_ac_mode_en".

These ports are only created when AC pads are present in the design.

The BoundaryScanCellOptions wrapper inside the BoundaryScan and EmbeddedBoundaryScan wrappers includes the following argument:

- **add_dot6_from_pad_cell** — identifies the AC input and inout ports where the from_pad data signal will be intercepted with an additional boundary scan cell.

These ports can also be defined using the set_boundary_scan_port_options command.

The ACModeOptions wrapper inside the BoundaryScan and EmbeddedBoundaryScan wrappers can be used to specify a number of IEEE 1149.6 properties of the boundary scan chain. See the ACModeOptions section of the Tessent Shell Reference Manual for complete information.

AC Pins in the PatternsSpecification Wrapper

Several wrappers and properties in the PatternsSpecification wrapper will affect the way AC pins are managed in your design.

For complete information on this wrapper, see the PatternsSpecification section of the Tessent Shell Reference Manual.

The LoadBoardInfo wrapper includes the dot6_ttest argument. This property (referred to as TTest in the IEEE 1149.6 standard) specifies the minimum time required by the slowest coupling capacitor, among all AC Loopbacks, to fully discharge. It is used mainly by DC BoundaryScan tests such as dot6_dc_input and dot6_dc_output to ensure that DC levels are not captured by the test receiver.

The default value of dot6_ttest is three times the slowest time constant among all AC pads that have an on-chip high-pass or low-pass filter (as recommended by the IEEE 1149.6 standard). If no on-chip filter is present or if no BSDL file can be found, the value defaults to three times the TCK period.

The ACLoopbacks wrapper inside the LoadBoardInfo wrapper specifies connection loopbacks from AC source ports to AC destination ports through a coupling capacitor. This wrapper takes arguments in pairs (*destination_port* : *source_port*) with the following restrictions:

- *source_port* and *destination_port* must be output and input respectively. Inout ports are not allowed.
- 1149.1 ports are not allowed.
- To loop back differential pairs, specify only the positive leg's port names. The negative leg will be inferred from the BSDL file or taken from the differential_inverse_of properties of the specified ports.

Value	Description
dot6_ac_00	This test measures AC parameters Vhyst_edge and Thyst on contacted 1149.6 input pins by applying a valid logic 0 to the input pins and expecting to capture a logic 0. This test applies AC waveforms to contacted inputs and uses the EXTEST_PULSE code.

Value	Description
dot6_ac_01	This test measures AC parameters Vhyst_edge and Thyst on contacted 1149.6 input pins by applying an invalid logic 0 to the input pins and expecting to capture a logic 1. This test applies AC waveforms to contacted inputs and uses the EXTEST_PULSE code. This test is meant for manufacturing patterns and will never pass simulation because it assumes that the test receiver will react correctly to an invalid 0 or 1.
dot6_ac_10	This test measures AC parameters Vhyst_edge and Thyst on contacted 1149.6 input pins by applying an invalid logic 1 to the input pins and expecting to capture a logic 0. This test applies AC waveforms to contacted inputs and uses the EXTEST_PULSE code. This test is meant for manufacturing patterns and will never pass simulation because it assumes that the test receiver will react correctly to an invalid 0 or 1.
dot6_ac_11	This test measures AC parameters Vhyst_edge and Thyst on contacted 1149.6 input pins by applying a valid logic 1 to the input pins and expecting to capture a logic 1. This test applies AC waveforms to contacted inputs and uses the EXTEST_PULSE code.
dot6_ac_input	This test verifies the AC operation of the 1149.6 receivers. It performs the following tasks: Applies a checkerboard pattern to contacted AC input/bidirectional pins. Disables all output/bidirectional pads. Applies a valid AC waveform to contacted input and inout pins. Tests the “no transition” detection capability, if present, by applying two consecutive test patterns while keeping the AC pin level constant. If the pad contains either a Low-Pass filter or an embedded High-Pass filter, the absence of transition in the second pattern will make its corresponding boundary-scan cell capture its default value. Uses both EXTEST_PULSE and EXTEST_TRAIN codes.
dot6_ac_output	This test verifies the AC operation of the 1149.6 transmitters and loopbacks. It performs the following tasks: Enables all AC output pads. Ignores and turns off all non-AC pads, if possible. Uses both AC and DC loopbacks if present. Uses both EXTEST_PULSE and EXTEST_TRAIN codes. Alternatively disables and enables ACSelect cells. Runs the test on one Enable Group at a time. Strobes AC waveforms on all AC contacted outputs.

Value	Description
dot6_ac_select_cells	This test verifies that the netlist properly implements the AC grouping described in the BSDL. Only one ACSelect cell is enabled at a time, and its action is verified by checking that all AC output pins in its fanout properly toggle during the RunTestIdle state when the EXTEST_PULSE instruction is loaded. This test is necessary only during netlist verification in the design flow. In manufacturing, the dot6_ac_output test ensures that all AC select cells are properly operating by enabling or disabling them all at once.
dot6_dc_00	This test measures DC parameters Vthreshold and Vhyst_level on contacted 1149.6 input pins by applying a valid logic 0 to the input pins and expecting to capture a logic 0. All AC pad drivers are disabled for this test.
dot6_dc_01	This test measures DC parameters Vthreshold and Vhyst_level on contacted 1149.6 input pins by applying an invalid logic 0 to the input pins and expecting to capture a logic 1. All AC pad drivers are disabled for this test. This test is meant for manufacturing patterns and will never pass simulation because it assumes that the test receiver will react correctly to an invalid 0 or 1.
dot6_dc_10	This test measures DC parameters Vthreshold and Vhyst_level on contacted 1149.6 input pins by applying an invalid logic 1 to the input pins and expecting to capture a logic 0. All AC pad drivers are disabled for this test. This test is meant for manufacturing patterns and will never pass simulation because it assumes that the test receiver will react correctly to an invalid 0 or 1.
dot6_dc_11	This test measures DC parameters Vthreshold and Vhyst_level on contacted 1149.6 input pins by applying a valid logic 1 to the input pins and expecting to capture a logic 1. All AC pad drivers are disabled for this test.
dot6_dc_input	This 1149.6 test verifies the behavior of all 1149.6 AC input pins when operating in EXTEST mode. Just as the non-AC (in other words, DC) pins “input” test, this pattern applies zero and one logic values at the AC pins and checks that their 1149.6 test-receiver's associated bscan cells capture the same value. In addition, whenever the target tester supports it, the test will also verify that all AC pins can reliably detect invalid input voltage levels sitting in between the “zero” and “one” threshold voltages.

Value	Description
dot6_dc_output	This 1149.6 test is equivalent to the existing output IO test for normal DC pads. Just as in the output test, the dot6_dc_output test applies checkerboard values, activates output cells one enable group at a time, and uses load board loopbacks as well as IO internal loopbacks. In addition, this test also covers 1149.6-specific features such as coupling capacitors discharge time, ACSelect cells, and output inversion during the RunTestIdle state of the TAP.

Appendix C
Getting Help

There are several ways to get help when setting up and using Tessent software tools. Depending on your need, help is available from documentation, online command help, and Mentor Graphics Support.

Tessent Documentation System

At the center of the documentation system is the InfoHub that supports both PDF and HTML content. From the InfoHub, you can access all locally installed product documentation, system administration documentation, videos, and tutorials. For users who want to use PDF, you have a PDF bookcase file that provides access to all the installed PDF files.

For information on defining default HTML browsers, setting up browser options, and setting the default PDF viewer, refer to the "Documentation Options" in the *Mentor Documentation System* manual.

You can access the documentation in the following ways:

- **Shell Command —** On Linux platforms, enter **mgcdocs** at the shell prompt or invoke a Tessent tool with the -manual invocation switch.
- **File System —** Access the Tessent InfoHub or PDF bookcase directly from your file system, without invoking a Tessent tool. For example:

  HTML:

  ```
  firefox $MGC_DFT/docs/infohubs/index.html
  ```

  PDF

  ```
  acroread $MGC_DFT/docs/pdfdocs/_bk_tessent.pdf
  ```

- **Application Online Help —** You can get contextual online help within most Tessent tools by using the "help -manual" tool command. For example:

  > help dofile -manual

  This command opens the appropriate reference manual at the "dofile" command description.

Mentor Support Services

Mentor provides a range of industry-leading support services that keep design teams productive and up-to-date with Mentor products.

A Mentor support contract includes the following:

- **Software Updates —** Get the latest releases and product enhancements to keep your environment current.
- **Mentor Graphics Support Center —** Access our online knowledge base, personalized to your Mentor products.
- **Support Forums —** Learn, share, and connect with other Mentor users.
- **Technical Support —** Collaborate with Mentor support engineers to solve complex design challenges.
- **Regular Communications —** Receive the latest knowledge base articles and announcements for your Mentor products.
- **Mentor Ideas —** Share ideas and vote for your favorites to shape future products.

More information is available here:

`https://support.mentor.com`

If your site is under a current support contract, but you do not have a Support Center login, register today:

`https://support.mentor.com/register`

Third-Party Information

For information about third-party software included with this release of Tessent products, refer to the *Third-Party Software for Tessent Products*.

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. **ORDERS, FEES AND PAYMENT.**

   1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement (each an "Order"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not those documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order or presented in any electronic portal or automated order management system, whether or not required to be electronically accepted, will not be effective unless agreed in writing and physically signed by an authorized representative of Customer and Mentor Graphics.

   1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice. Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax, consumption tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.

   1.3. All Products are delivered FCA factory (Incoterms 2010), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation, setup files and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Except for Software that is embeddable ("Embedded Software"), which is licensed pursuant to separate embedded software terms or an embedded software supplement, Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 4.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer provides any feedback or requests any change or enhancement to Products, whether in the course of receiving support or consulting services, evaluating Products, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **BETA CODE.**

3.1. Portions or all of certain Software may contain code for experimental testing and evaluation (which may be either alpha or beta, collectively "Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. Mentor Graphics may choose, at its sole discretion, not to release Beta Code commercially in any form.

3.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

3.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 3.3 shall survive termination of this Agreement.

4. **RESTRICTIONS ON USE.**

4.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Except for Embedded Software that has been embedded in executable code form in Customer's product(s), Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use Products except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer becomes aware of such unauthorized disclosure or use. Customer acknowledges that Software provided hereunder may contain source code which is proprietary and its confidentiality is of the highest importance and value to Mentor Graphics. Customer acknowledges that Mentor Graphics may be seriously harmed if such source code is disclosed in violation of this Agreement. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, disassemble, reverse-compile, or reverse-engineer any Product, or in any way derive any source code from Software that is not provided to Customer in source code form. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' trade secret and proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Products or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Products, or disclose to any third party the results of, or information pertaining to, any benchmark.

4.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use, or as permitted for Embedded Software under separate embedded software terms or an embedded software supplement. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or on-site contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.

4.3. Customer agrees that it will not subject any Product to any open source software ("OSS") license that conflicts with this Agreement or that does not otherwise apply to such Product.

4.4. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense, or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

4.5. The provisions of this Section 4 shall survive the termination of this Agreement.

5. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer with updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at http://supportnet.mentor.com/supportterms.

6. **OPEN SOURCE SOFTWARE.** Products may contain OSS or code distributed under a proprietary third party license agreement, to which additional rights or obligations ("Third Party Terms") may apply. Please see the applicable Product documentation (including license files, header files, read-me files or source code) for details. In the event of conflict between the terms of this Agreement

(including any addenda) and the Third Party Terms, the Third Party Terms will control solely with respect to the OSS or third party code. The provisions of this Section 6 shall survive the termination of this Agreement.

7. **LIMITED WARRANTY.**

7.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification, improper installation or Customer is not in compliance with this Agreement. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

7.2. THE WARRANTIES SET FORTH IN THIS SECTION 7 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

8. **LIMITATION OF LIABILITY.** TO THE EXTENT PERMITTED UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 8 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

9. **THIRD PARTY CLAIMS.**

9.1. Customer acknowledges that Mentor Graphics has no control over the testing of Customer's products, or the specific applications and use of Products. Mentor Graphics and its licensors shall not be liable for any claim or demand made against Customer by any third party, except to the extent such claim is covered under Section 10.

9.2. In the event that a third party makes a claim against Mentor Graphics arising out of the use of Customer's products, Mentor Graphics will give Customer prompt notice of such claim. At Customer's option and expense, Customer may take sole control of the defense and any settlement of such claim. Customer WILL reimburse and hold harmless Mentor Graphics for any LIABILITY, damages, settlement amounts, costs and expenses, including reasonable attorney's fees, incurred by or awarded against Mentor Graphics or its licensors in connection with such claims.

9.3. The provisions of this Section 9 shall survive any expiration or termination of this Agreement.

10. **INFRINGEMENT.**

10.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to such action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

10.2. If a claim is made under Subsection 10.1 Mentor Graphics may, at its option and expense: (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.

10.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; (h) OSS, except to the extent that the infringement is directly caused by Mentor Graphics' modifications to such OSS; or (i) infringement by Customer that is deemed willful. In the case of (i), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

10.4. THIS SECTION 10 IS SUBJECT TO SECTION 8 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, FOR DEFENSE,

SETTLEMENT AND DAMAGES, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

11. **TERMINATION AND EFFECT OF TERMINATION.**

11.1. If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.

11.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination of this Agreement and/or any license granted under this Agreement, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.

12. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and European Union ("E.U.") and United States ("U.S.") government agencies, which prohibit export, re-export or diversion of certain products, information about the products, and direct or indirect products thereof, to certain countries and certain persons. Customer agrees that it will not export or re-export Products in any manner without first obtaining all necessary approval from appropriate local, E.U. and U.S. government agencies. If Customer wishes to disclose any information to Mentor Graphics that is subject to any E.U., U.S. or other applicable export restrictions, including without limitation the U.S. International Traffic in Arms Regulations (ITAR) or special controls under the Export Administration Regulations (EAR), Customer will notify Mentor Graphics personnel, in advance of each instance of disclosure, that such information is subject to such export restrictions.

13. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. The parties agree that all Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to U.S. FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. government or a U.S. government subcontractor is subject solely to the terms and conditions set forth in this Agreement, which shall supersede any conflicting terms or conditions in any government order document, except for provisions which are contrary to applicable mandatory federal laws.

14. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.

15. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FlexNet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 15 shall survive the termination of this Agreement.

16. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the U.S. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, U.S., if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America or Japan, and the laws of Japan if Customer is located in Japan. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply, or the Tokyo District Court when the laws of Japan apply. Notwithstanding the foregoing, all disputes in Asia (excluding Japan) arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. Nothing in this section shall restrict Mentor Graphics' right to bring an action (including for example a motion for injunctive relief) against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

17. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

18. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements. Any translation of this Agreement is provided to comply with local legal requirements only. In the event of a dispute between the English and any non-English versions, the English version of this Agreement shall govern to the extent not prohibited by local law in the applicable jurisdiction. This Agreement may only be modified in writing, signed by an authorized representative of each party. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 170330, Part No. 270941