ATPG Flow with Modus DFT Software Solution

Course Version 22.1

Lecture Manual **Revision 1.0**

cadence®

Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Table of Contents

ATPG Flow with Modus DFT Software Solution

ATPG Flow with Modus DFT Software Solution

Version 22.1

Revision **1.0**

Estimated time: **1.5 Days**

cadence

This page does not contain notes.

Module 1

About This Course

cadence

This page does not contain notes.

Course Prerequisites

Before taking this course, you need to know the basics of

- Test generation tools
- Digital IC testing

cadence

This page does not contain notes.

Course Objectives

In this course, you

- Introduce the Modus DFT Software Solution
- Define the ATPG (Automatic Test Pattern Generation) flow
- Build a test model (building the Modus DFT Software Solution design database)
- Build the fault model
- Build test modes
- Verify test structures (design rule checking)
- Create static tests (Automatic Test Pattern Generation)
- Write the vectors (Verilog, STIL, WGL)
- Debug the broken scan chains using the GUI and Tcl command-line techniques
- Debug the test patterns

This page does not contain notes.

Course Agenda

- Introduction to Modus DFT Software Solution
  - Introduction to Modus ATPG Flow Database and Steps to Invoke the Tool
- The ATPG Flow
  - Perform the Modus ATPG (Automatic Test Pattern Generation) Flow
  - Generating ATPG Vector
- Debug Scan Chains with GUI and Tcl Interface
  - Debugging Broken Scan Chains with Modus GUI
  - Debugging Broken Scan Chains with Tcl Command Line Interface
- Analyze Initialization Sequence Using Modus GUI
  - Analyzing Initialization Sequences Using Modus GUI
- Debugging the Test Pattern
  - Simulating and Debugging Vectors
- Additional Definition and Concepts

cadence

This page does not contain notes.

Software and Licenses

For the software and licenses used in the labs for this course, go to:

https://www.cadence.com/en_US/home/training/all-courses/82137.html

If there is additional information regarding the specific software, it is detailed in the lab document and/or the README file of the database provided with this course.

cadence

This page does not contain notes.

Become Cadence Certified by Earning a Digital Badge

Digital badges indicate mastery in a certain technology or skill and give managers and potential employers a way to validate your expertise.

- Cadence Training Services offers digital badges for our popular training courses.
- Your digital badge can be added to your email signature or social media platforms like LinkedIn or Facebook.

Benefits of Cadence Certified Digital Badges

- Validate expertise
  - Expand career opportunities
- Professional credibility
  - Stand apart from your peers
- For more information, go to www.cadence.com/training or email es_digitalbadge@cadence.com.

How do I register to take the exam?

- Log in to our Learning Management System, click on the course in your transcript, and go to the Content tab to locate the exam.

How long will it take to complete the exam?

- Most exams take 45 to 90 minutes to complete. You may retake the exam multiple times to pass the exam.

How do I access and use the digital badge?

- After you pass the exam, you get a digital badge and instructions on how to place it on social media sites.

How is the digital badge validated?

- Credly validates the digital badge as issued to you by Cadence and includes the details of the criteria you completed to earn the badge.

This page does not contain notes.

Icons Used in This Class


Best Practice


Language/
Command Syntax


Concept/
Glossary


Frequently Asked
Questions/
Quiz


Error Message


Problem & Solution


Quick Reference


GUI and Command


How To


Lab List

Throughout this class, we use icons to draw your attention to certain kinds of information. Here are the icons we use, and what they mean.

cadence

This page does not contain notes.

Module 2

Introduction to Modus DFT Software Solution

cadence

This page does not contain notes.

Module Objectives

In this module, you

- Explain the features of the Modus DFT Software Solution and describe how it integrates with the Genus™ Synthesis Solution
- Describe the ATPG (Automatic Test Pattern Generation) flow and its process
- Describe the full-chip integration
- Differentiate between the traditional flow and the full-chip DFT integration flow
- Set up and invoke the Modus DFT tool
- Explain the Modus documentation structure
- Demonstrate how to access Modus documentation and RAKs
- Display Modus commands, objects, and attributes

cadence

This page does not contain notes.

Modus DFT Software Solution

The Modus software is a fully integrated suite of tools designed from the beginning to interact and work with each other.

It is a complete tool suite supporting all four disciplines:

- **Test Synthesis**
  - The process of inserting test logic into a design to help make it testable.
- **Test Analysis**
  - The process of identifying the test logic and verifying that it has been inserted and integrated correctly. And identify any design characteristics that will create testing problems.
- **Test Generation**
  - The process of creating test vectors that are applied at the tester.
- **Test Diagnostics**
  - The process of working backward from failure miscompares to the probable cause and helps in the production and yield management of semiconductor lines.

cadence

This page does not contain notes.

Modus DFT Software Solution (continued)

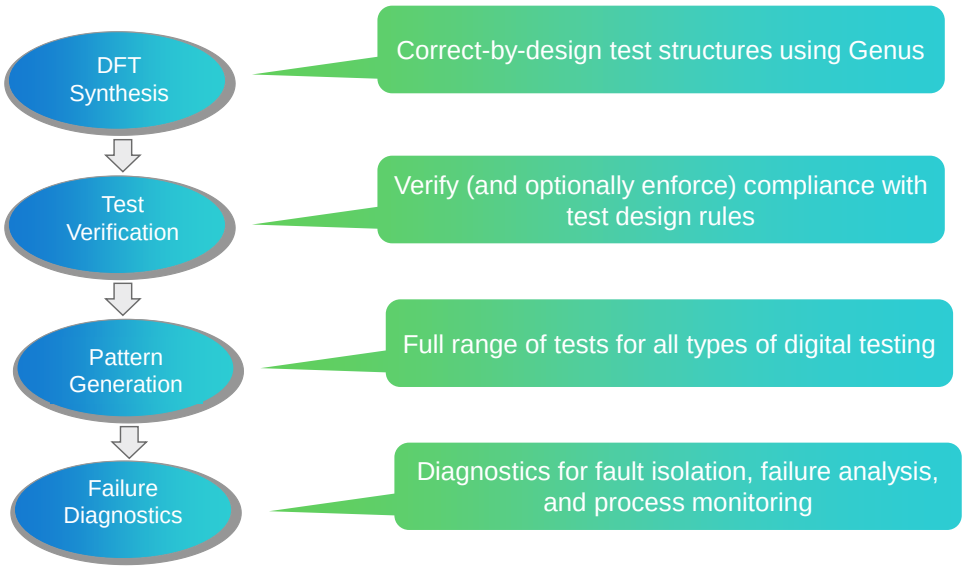
Modus DFT Software Solution provides:

- Comprehensive manufacturing test offering for logic and embedded memory IP.
- Single pass logic synthesis, test insertion, and pattern generation, that is, physically and power domain aware.
- Reduced cost of defect detection with higher quality test patterns and accurate silicon defect diagnostics.
- Applicable to all design sizes from small mixed-signal to very large SoCs.
- Test coverage analysis and augmentation.
- Common User Tcl Interface.

cadence

This page does not contain notes.

Modus DFT Software Disciplines



cadence

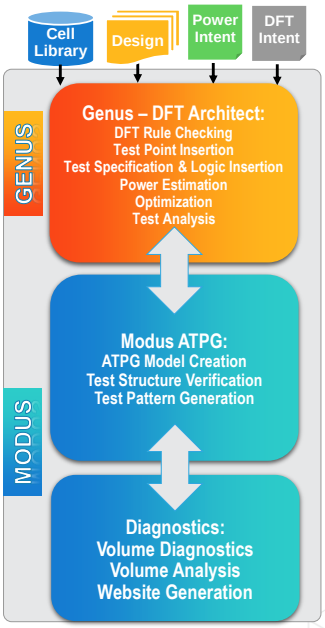
This page does not contain notes.

Modus DFT Software Solution Highlight and Features

Modus tightly integrates with Genus Synthesis Solution to help the user get up and start. Genus generates the downstream files to perform exhaustive testing, test point insertion, and reduced cost of defect detection.

Modus DFT Software Solution Components:

- Modus scan offers DFT Rule checking, Test Point Insertion, and Optimization.
- Modus scan tightly integrated into Genus.
- Modus ATPG.
- Modus Diagnostics and Trend Analysis.



cadence

This page does not contain notes.

Modus: Installation, Setup, and Invocation

Install Modus using either InstallScape or Softload utilities.

- Install base release first
- Install the most recent update

Setup the Modus Tool

- Update your .cshrc file:
  - setenv CDS_LIC_FILE yourLicenseFilePath

    setenv MODUS_PATH {MODUS_Install_Path}

    set path = ($path $MODUS_PATH/tools/bin)
- The $MODUS_PATH should be last in your path after Incisive® and Genus

Bring up the Modus Tcl shell

- **`modus`**

Bring up the Modus Tcl shell with GUI

- **`modus -gui`**

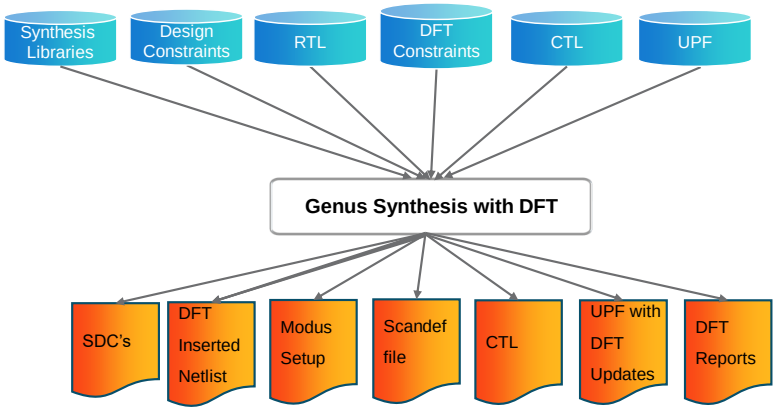
cadence

This page does not contain notes.

Genus Synthesis with DFT: Inputs and Outputs

Genus synthesis with DFT generates downstream files to get started with Modus ATPG flow:

- Inputs:
  - Synthesis libraries
  - Design Constraints
  - RTL
  - DFT Constraints
  - CTL's
  - UPF
- Outputs:
  - SDC's
  - DFT Inserted Netlist
  - Modus Setup
  - Scandef file
  - Scan Abstract Model/CTL
  - UPF with DFT Updates
  - DFT Reports



This page does not contain notes.

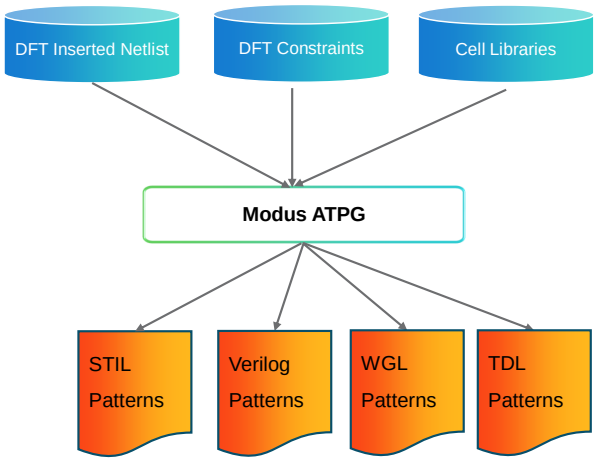
Modus ATPG: Inputs and Outputs

Inputs:

- DFT Inserted Netlist
- Test Constraints
- Cell Libraries

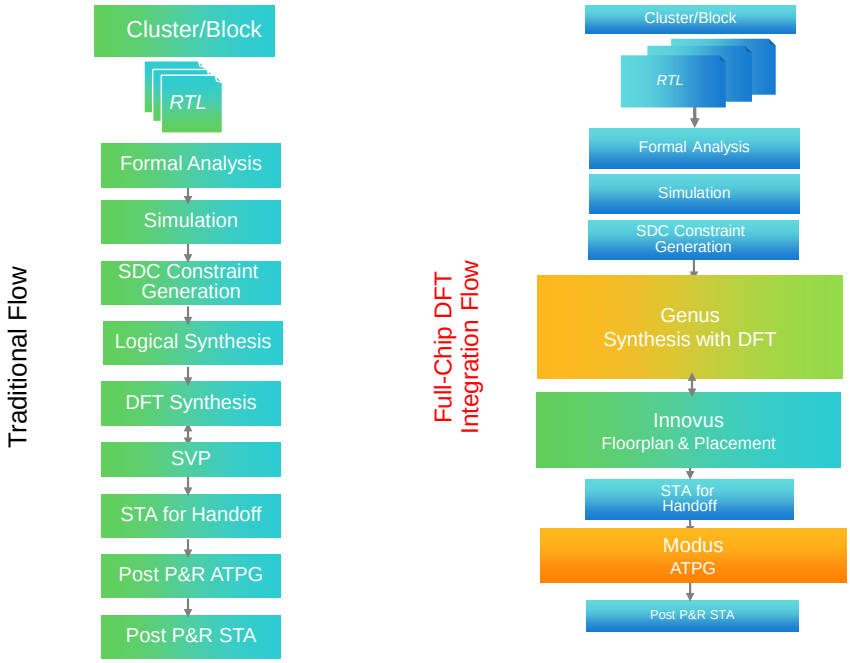
Outputs:

- STIL Patterns
- Verilog Patterns
- WGL Patterns
- TDL Patterns



This page does not contain notes.

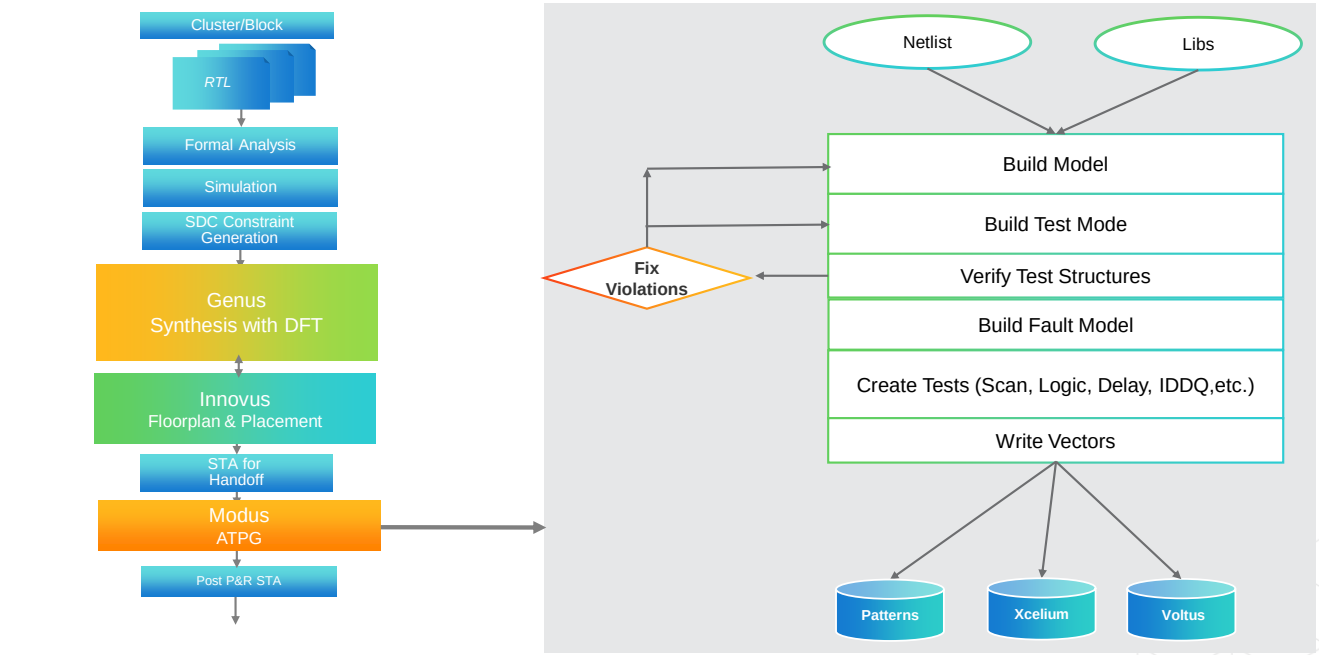
Full-Chip DFT Traditional and Integrated Flow



cadence

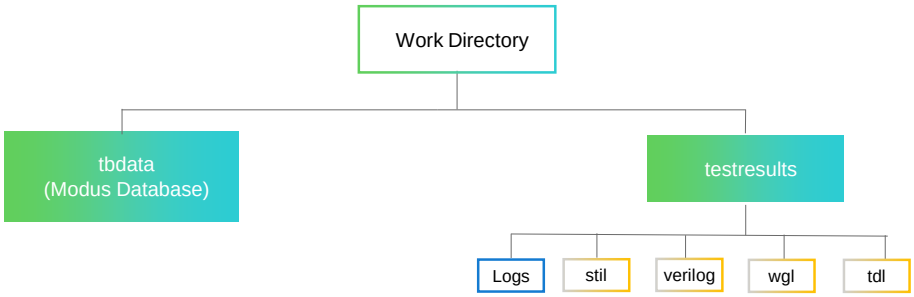
This page does not contain notes.

Modus Automatic Test Pattern Generation ATPG Flow



This page does not contain notes.

Modus Design Database Directory Structure



Directory

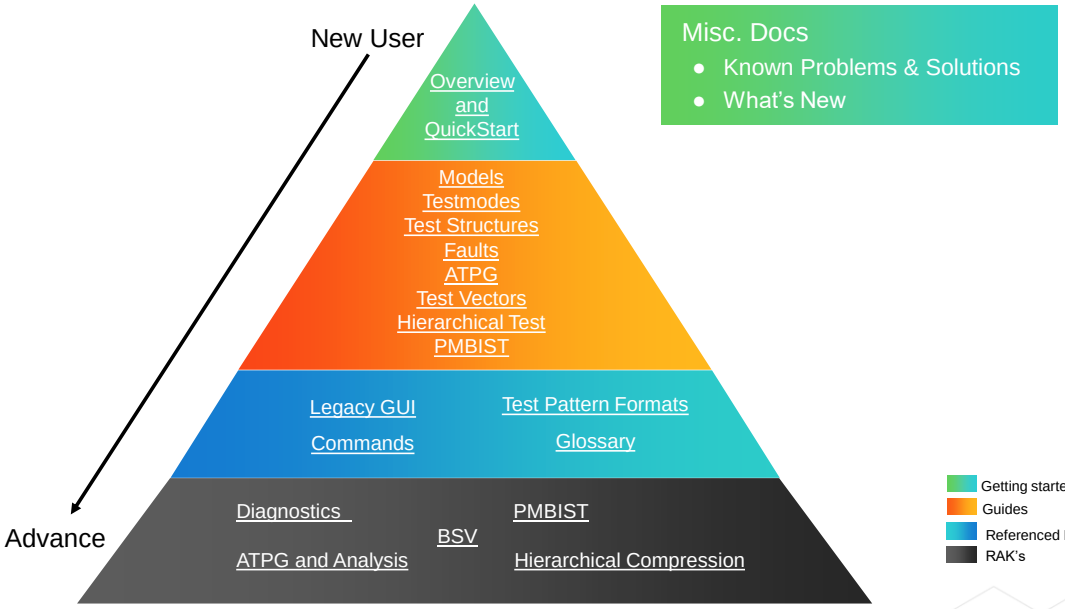
Pattern output Directories

Command log file

cadence

This page does not contain notes.

Modus Documentation Structure



cadence

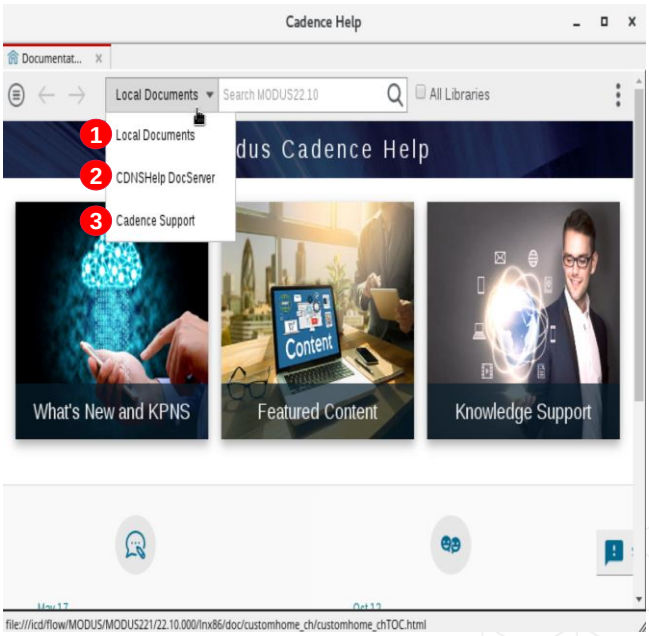
This page does not contain notes.

Accessing the Modus Documentation

`cdnshelp`

You can access the Modus documentation through the following:

- Type **`cdnshelp`** on the Modus prompt after launching Modus.
  1. Displays content shipped with the release (default)
  2. Displays doc kit posted on CDNSHelp backend server (for dynamic content)
  3. Displays content from COS



cadence

This page does not contain notes.

Accessing the Modus Documentation (continued)

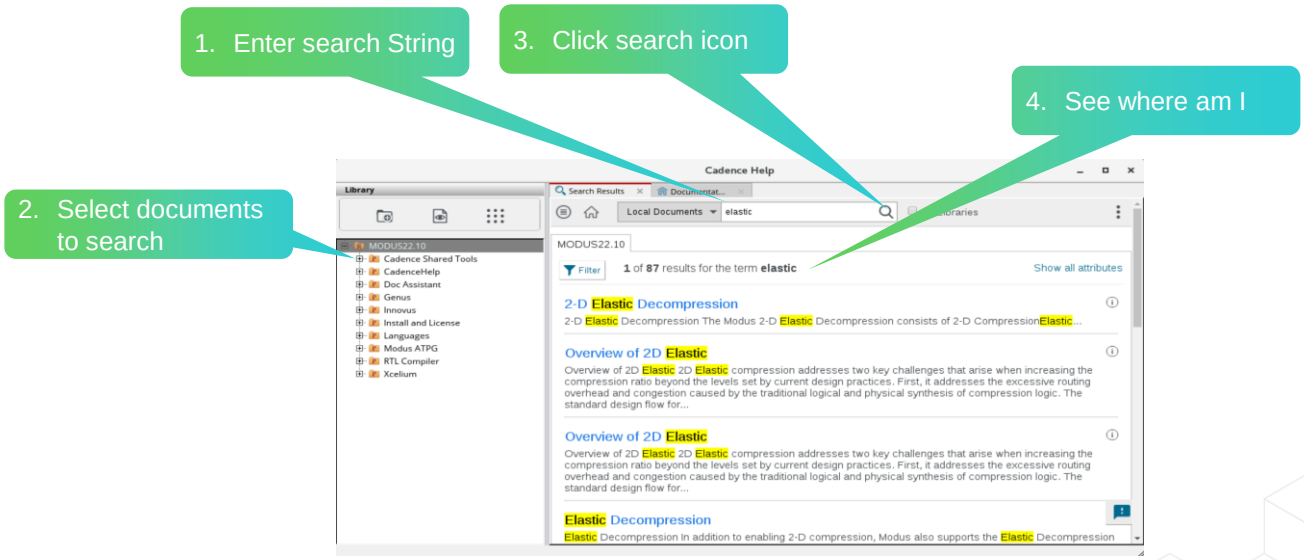
You can also access the Modus documentation through:

- **Customer Online Support (COS)**
  - Sign in to COS (https://support.cadence.com)
    - Select Modus Test from the Products list on COS home page to access Modus documentation, RAKs, videos, training, and application notes.
- **From install directory**
  - Full documentation set (HTML and PDFs) found in **<Modus_install_path>/doc**
    - HTML Index page: **<Modus_install_path>/tools/tb/etc/doc/debooks.html**
- **Modus_Portfolio.pdf – A collated document containing all Modus guides in PDF format**
  - Location: **<Modus_install_path>/tools/tb/etc/doc/Modus_Portfolio.pdf**

cadence

This page does not contain notes.

Searching the Modus Documentation Using `cdnshelp`



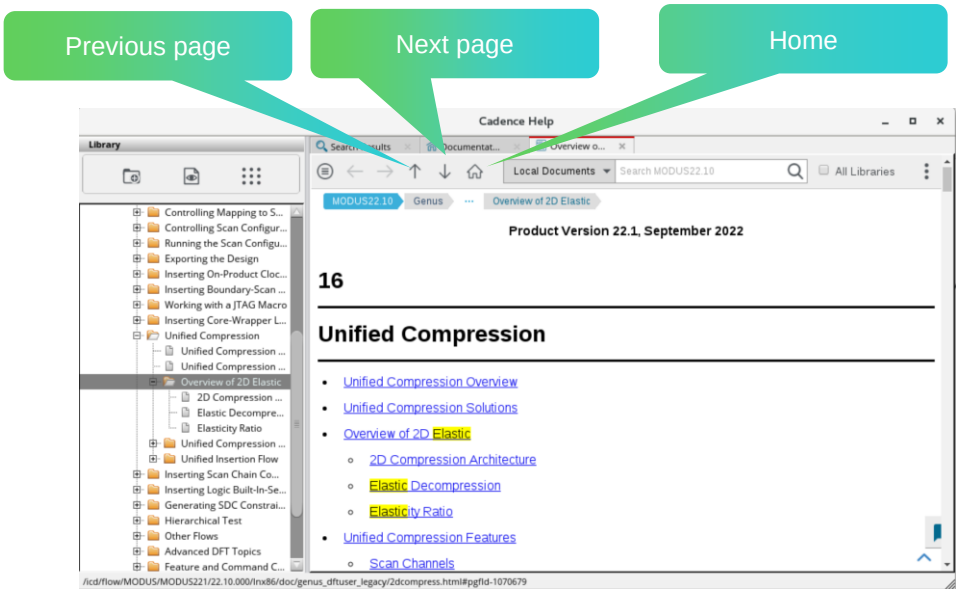
24

The search is fairly intuitive:

Enter the search string and click the search icon and the results come up.

You can select whether to search all the items in the list or just the selected ones.

Searching the Modus Documentation Using `cdnshelp` (continued)



25

When the search results come up; double-click the item to bring up the document viewer.

The document browser provides:

- Where am I – if you are many levels deep in the document it gives you an indication of where you are.
- Previous/Next – moves back and forth through the document
- Home – moves back to the beginning
- Zoom In/Out

Displaying the *help* Command Line Options

help Command Line Option to Display Modus Commands, Objects, and Attributes	Description
• *help –command ** • *help –command re** • *help –command <cmd_name>*	• Display a list of all supporting commands. • List all the commands starting with re. • Display all options for the specified command.
• *help –obj ** • *help –obj c** • *help –obj <obj_name>*	• List all Modus object. • List all objects starting with c. • Displays list of attributes for the specified object.
• *help –attribute ** • *help –attribute c** • *help –attribute <attr_name>*	• List all Modus attributes. • List all attributes starting with c. • Displays description and usage of the specified attribute.

Suppose you use the help command without any parameter; Modus searches and prints the help for all commands, objects, and attributes matching the criteria. For example, help *net* displays help for all commands, objects, and attributes containing the string ‘net’.

cadence

This page does not contain notes.

Querying Attribute

```
get_db <object> .<attribute_name>
```

get_db command also provides help on commands, options, and attributes:

- To print all supported documents
  - **get_db commands .help**
- To display the command description
  - **get_db command:<cmd_name> .help**
- To display the list of options for the specified command
  - **get_db command:<cmd_name> .options**

Example

get_db testmode

```
@modus:root:/ 14> get_db testmode
testmode:FULLSCAN _
```


This page does not contain notes.

Man Command Attribute

```
man <command name> <XXXmessages> <msg help>
```

man command provides a man page with general information about the commands.

- To provide a man page with all help text for the command and its options use,
  - **man <cmd_name>**
- To provide a man page with all the messages generated with an ID starting with XXX. For example, man TSVmessages.
  - **man XXXmessages**
- To provide a man page with general information about the format, severity codes, and return codes for Modus messages use:
  - **man MessageInfo provides**
- To provide help for the specific message use:
  - **msgHelp <msg_number>**

This page does not contain notes.

Quick Reference Guides: Rapid Adoption Kits (RAKs)

- RAKs are a quick source of sample flows and commands for
  - Setting up a Cadence® tool.
  - Demonstrating the flow and scripts.
- Rapid adoption kits are available for Modus on support.cadence.com
  - Modus Test: ATPG and Analysis
  - Modus DFT Software Solution – Advanced ATPG Rapid Adoption Kit (RAK) Overview
  - Modus Test: Diagnostics
  - Modus and Genus: Programmable Memory Built-In-Self-Test (PMBIST)
  - Modus and Genus: Hierarchical Compression RAK - GENUS-DFT and MODUS Test
  - Modus and Genus: Boundary Scan Insertion & Verification (BSV)

This page does not contain notes.

Module Summary

In this module, you learned how to

- Explain the features of the Modus DFT Software Solution and describe how it integrates with the Genus Synthesis Solution
- Describe the ATPG (Automatic Test Pattern Generation) Flow
- Describe the full-chip integration
- Differentiate between traditional flow and full-chip DFT integration flow
- Setup and invoke the Modus DFT tool
- Explain the Modus documentation structure
- Demonstrate how to access Modus documentation and RAKs
- Display Modus commands, objects, and attributes

cadence

This page does not contain notes.

Lab

Lab 2-1 Introducing Modus ATPG Flow Database and Steps to Invoke the Modus Tool

- Bring up and Execute Modus
- Directory Structure
- Tool Versions
- Unzip the Database and Run the Lab

cadence

This page does not contain notes.

Module 3

The ATPG Flow

cadence

This page does not contain notes.

Module Objectives

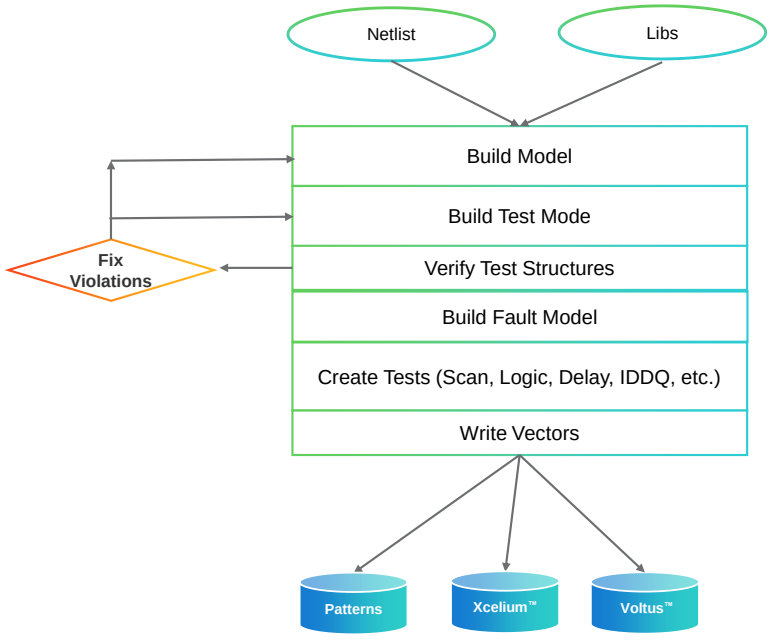
In this module, you

- Identify the steps involved in the ATPG flow
- Build a test model
- Define and create test mode
- Verify the test structure to ensure it is working correctly
- Build a fault model to identify and locate faults
- Perform different types of ATPG tests, such as:
  - Dynamic fault test
  - Scan chain test
  - Sequential test
  - Bridging fault test
  - IDDQ tests
- Write out ATPG vectors to perform tests

cadence

This page does not contain notes.

The ATPG Flow



This page does not contain notes.

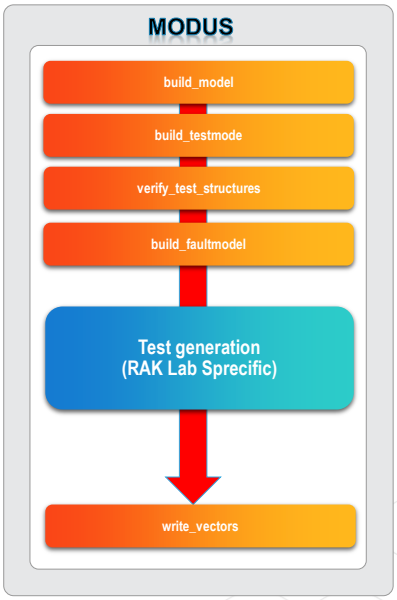
High Level ATPG Phases

Phases	Command/Topics

Model Build and Verification:
• Common to all ATPG flows
• Walk-through in Lab 1
	`build_model`
	`build_testmode`
	`verify_test_strucutures`
	`report_test_structures`
	`build_faultmodel`

Test Generation:
• Sub-flow for generating test vectors
• Customizable to meet needs
	Test Preparation:
	`read_sdc`
	`read_sdf`
	`prepared_timed_sequence`
	`prepare_opcg_test_sequence`
	ATPG:
	Logic tests, Top-off Tests

Vector Processing: • Common to all ATPG • Walk-through in Lab 1	`write_vectors` Vector Simulation (Verilog)



35

cadence

This page does not contain notes.

Modus TCL and GUI Interface

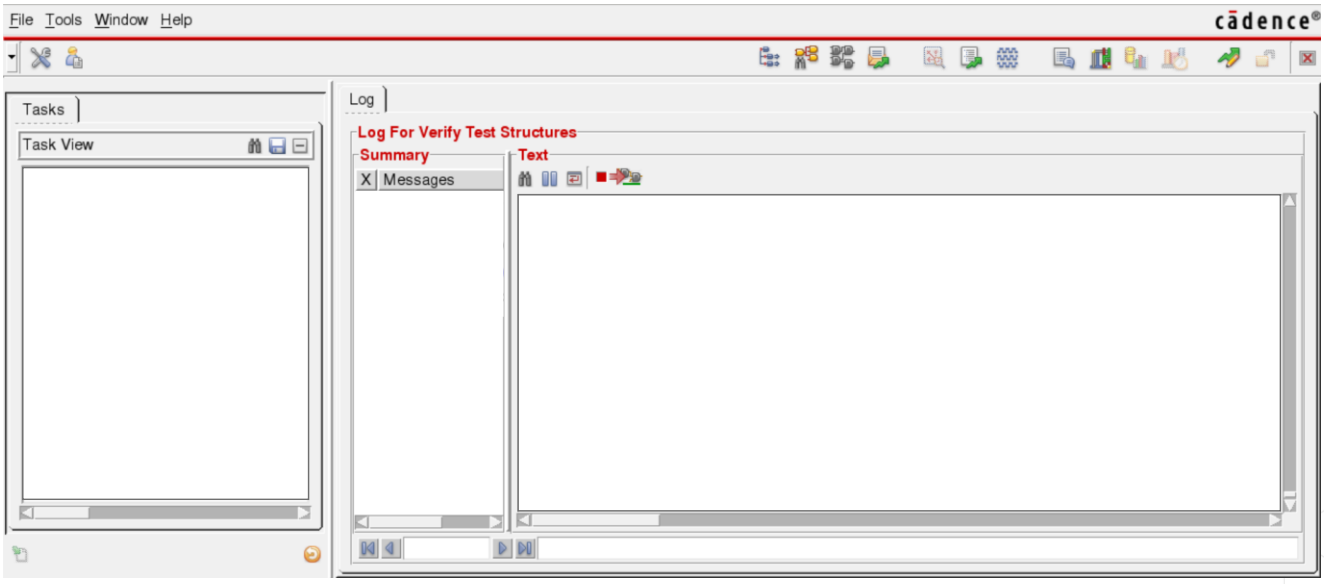
Modus supports two interfaces to the applications.

- Shell environment: The system can be started in the Tcl shell by sourcing a Tcl script or directly without an initial command.
- Design editor: Has **GUI (Graphical User Interface)** mode and **command line interface.**
  - Tcl command line interface: Design Edition has its own Tcl commands as well as interfaces to traditional Unix commands. A command line can accept both commands and scripts, and a graphical user interface (GUI).
    - Bring up the Modus Tcl shell
      - `modus`
  - GUI mode: Provides the steps to run as a push button flow.
    - Bring up the Modus Tcl shell with GUI
      - `modus -gui`

cadence

This page does not contain notes.

Example: Modus GUI Window

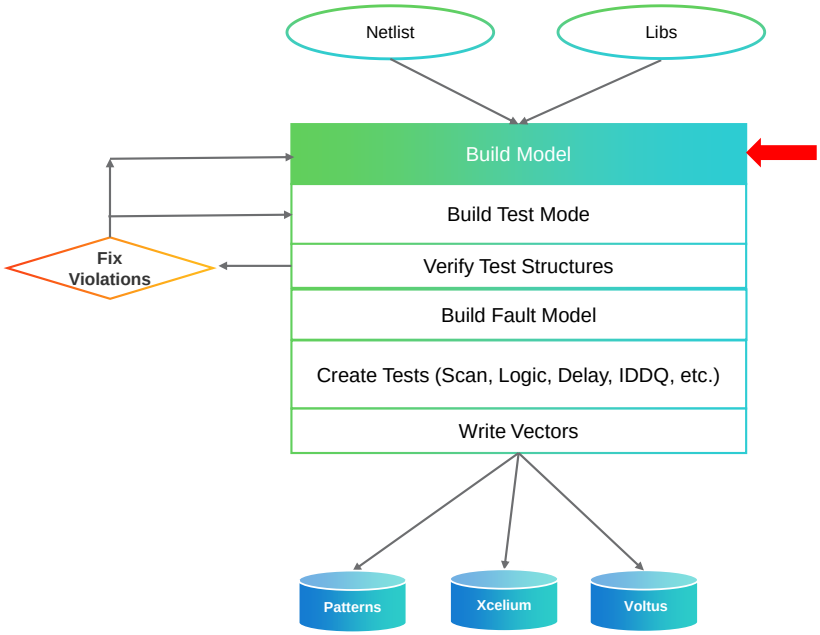


37

cadence

This page does not contain notes.

The ATPG Flow: Build Model

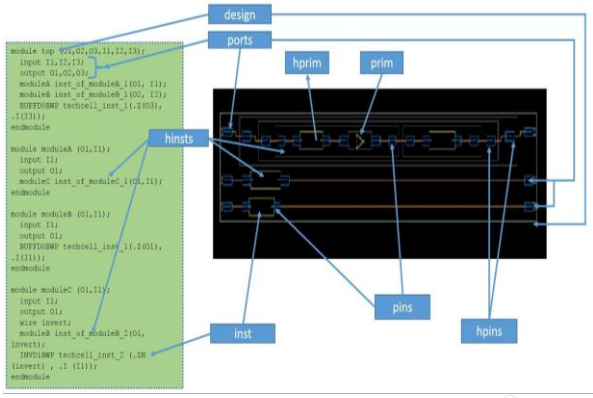


This page does not contain notes.

Build Model Overview

The creation of Modus model of the design is defined as **Build Model**. It does this by reading the design netlist and the structural library files and combining them together to create a complete design image.

- The *build_model* command reads in a netlist and libraries to create an optimized test model.
- This test model is used for all further steps in the ATPG flow.
- Example of a test model from Verilog to schematic is shown here.



cadence

This page does not contain notes.

Build Model: Inputs and Outputs

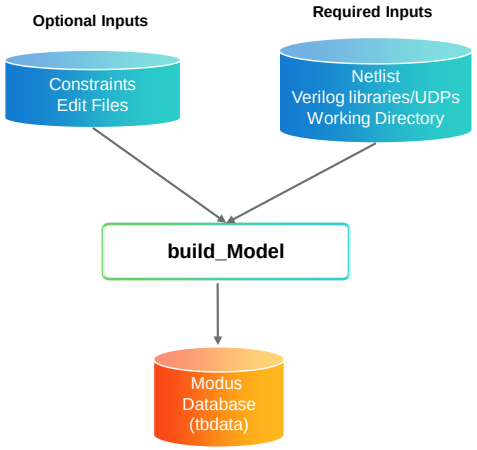
Required inputs:

- Netlist
  - Generally, this is the representation of the design in Verilog.
- Library files listed in the techlib option
  - These are the structural library files that define the technology that is instanced in the design netlist.

Optional inputs:

- Constraints
  - Ability to add logic constraints to avoid certain patterns from being generated.
- Edit files
  - Capability to edit the model from the command line (and GUI).

Output: A binary view of the design for Modus to process and a report of any issues with the process of building that model.



40

cadence

Netlist – This is the representation of the design generally in Verilog.

Library Files – These are the structural library files that define the technology that is instanced in the design netlist.

The netlist and library may consist of one or more files.

Note: Verilog is the most common language used for both Netlist and Library data. However, other structural netlist languages (e.g. MTV, IDM) are also supported.

Optional Inputs

Constraints – These are attributes that identify logic value requirements/restrictions (such as, One Hot Nets). The specified constraints cause special logic to be synthesized and added to the Modus model so that if the constraint is violated, it will appear as a three-state contention. constraints file which can be used to limit generation of certain types of vectors..

Edit Files – This is a file that can be used to delete, add, or change certain objects or attributes from the design (examples are in Additional Information section at the end of this presentation)

Output:

A binary view of the design for Modus to process and a report of any issues with the process of building that model

Required Inputs: Netlist and Libraries

Netlist and library data can be provided in any combination of the following formats:

- Minimal support for assigned statements (and, or, not)
- Structural Verilog format
- Verilog UDP
- Pre-existing Modus models
- Modus Primitives (MUX, LATCH, RAM, ROM)

When building a model, Modus will use the first definition of the cell/module that is found. The search order is:

- Design Source(s) in the order provided
- Technology Libraries in the order provided

cadence

This page does not contain notes.

Required Inputs: Lib Files/Multiple Designs

- Multiple design files and libraries are separated by commas and no spaces:
  - `build_model –cell cellname -workdir <dir>  -designsource design1.v,design2.v  -techlib library1.v,library2.v,library3.v:/dir/`
- When using `define statements that need to apply to multiple libraries or design files, separate them with a colon, not a comma:
  - `build_model –cell cellname –workdir <dir> -designsource include.v:design1.v:design2.v -techlib library1.v:library2.v:library3.v:/dir/`
- Design files and libraries can be listed in a file that must be designated with a .files extension:
  - `build_model -designsource allnetlist.files –techlib alllibs.files`
- definemacro keyword:
  - `build_model –designsource XXX –techlib YYY –definemacro M1,M2`
  - The macro defined on the command line is global (applies to all files)
  - Specifications in the verilog override the command line specification
  - `ifdef M1  in the verilog source file, we process the logic defined inside the `ifdef block.
  - `undef M1 in the verilog source file will remove the definition of M1

cadence

This page does not contain notes.

Optional Inputs: Constraints and Edit Files

- **-constraints** option is used to specify constraints.
  - These constraints are added to the model in order to prevent certain patterns from being generated.
- **-editfile** option is used to specify edits to be applied to the Modus model.
  - This file is used to add/delete/change/connect/disconnect/copy pins, cells, nets, instances, and/or test points to an existing test model.

The format of the edit file is discussed in the Additional Information section.

cadence

This shows the command line invocation to include optional inputs, constraints and edit file.

Building the Modus Model

```
build_model \
-workdir <path of working
directory> \
-cell <design top level module> \
-techlib <comma/colon separated
list of libraries> \
-designsource          <netlist> \
-allowmissingmodules <yes/no> \
-blackboxoutputs      <Z/X/0/1> \
-constraints           <Filename> \
-editfile              <Filename> \
```

Example

```
build_model \
-workdir .    \
-cell DLX_TOP \
-designsource DLX_TOP.v \
-techlib ../TECHLIB/*.v \
-allowmissingmodules yes \
-blackboxoutputs z
```


cadence

This page does not contain notes.

Undefined Cells

By default, if a cell is not defined, an error message is given, and no model is built. Typically, there are three main missing cell types:

- No module definition for the cell.
- Module definition exists, but the cell has empty contents.
- Module definition exists, but Modus does not understand some parts of the function due to RTL description.

Two command-line options control how **build_model** handles modules in the netlist:

- Option **`-allowmissingmodules yes`**
  - If set to no, build_model will error out on missing modules and issue a warning on empty ones.
- Option **`-blackboxoutputs X/Z/0/1`**
  - Allows you to specify the value used to tie blackbox outputs.
  - Default is X.
- It is recommended to run **build_model** the first time with option **`-allowmissingmodules`** no in order to identify if any key modules are missing.

cadence

Build Model will use the first definition of a cell that it finds. This is important because at times the same cell might be defined multiple times in different directories or files. If the desire to have one particular cell used, make sure it is specified first in your lists.

Build Model also has support to create black boxes for undefined cells. The tool does its best to determine what are inputs and outputs and will be pessimistic in the models it creates. User can also create a shell for a cell to make sure the tool gets the correct inputs/outputs. A shell in Verilog defined the inputs and outputs, but there is no contents to the cell.

```
module logic2 (Z,Z1,A,B,C);
input A,B,C;
output Z,Z1;
endmodule
```


Build Model Summary

```
INFO (TLM-055): Design Summary
                --------------

Hierarchical Model:                   Flattened Model:
         68,065  Blocks                        30,230  Blocks
        209,618  Pins                          30,230  Nodes
        116,504  Nets

Primary Inputs:                       Primary Outputs:
         11  Input Only                          45  Output Only
         33  Input/Output                        33  Input/Output
         44  Total Inputs                        78  Total Outputs

Tied Nets:                            Dotted Nets:
         74  Tied to 0                           0  Two-State
         46  Tied to 1                          33  Three-State
          0  Tied to X                          33  Total Dotted Nets
        120  Total Tied Nets

Selected Primitive Functions:
          0  Clock Chopper (CHOP) primitives
          0  RAMs
          0  ROMs
         33  TSDs
          0  Resistors
          0  Transistors
      1,889  MUX2s
          2  Latches


      1,526  Rising  Edge Flop w/Set-Dominant and Reset Port
      1,526  Total Flops

      8,155  Technology Library Cell Instances

[end TLM_055]
Optimization removed logic for 7 of 89 cells in this design.

Optimization removed a total of 5,948 tied Latch Ports.
Optimization removed a total of 7,466 non-controlling inputs.
Optimization removed a total of 24,744 dangling logic nodes.


INFO (TEI-199): Build Model - Flat Model Build completed.  [end TEI_199]
```

```
--------------------------------------------------------------------------------
*                        Message Summary                                       *
--------------------------------------------------------------------------------
 Count  Number             First Instance of Message Text
------- ------             ------------------------------

  INFO Messages...
      1 INFO (TEI-195): Build Model - Controller starting:
      1 INFO (TEI-196): Build Model - Hierarchical Model Build starting:
      1 INFO (TEI-197): Build Model - Hierarchical Model Build completed.
      1 INFO (TEI-198): Build Model - Flat Model Build starting:
      1 INFO (TEI-199): Build Model - Flat Model Build completed.
      1 INFO (TEI-200): Build Model - Controller completed.
      1 INFO (TLM-055): Design Summary

  WARNING Messages...
     11 WARNING (TEI-110): Pin 'NOTIFIER' of 'cell udp_dff' has no external net connection for any
 usage in the design. Cell contents file: '/home/shubham8/work/JumpStart_LABS/TECHLIB/pads.v'.
      1 WARNING (TEI-275): Mixed signal strengths not supported on line 3763.

 For a detailed explanation of a message and a suggested user response execute 'msgHelp <message i
 d>'.  For example: msgHelp TDA-009

--------------------------------------------------------------------------------
```


cadence

This page does not contain notes.

Reporting Logic Model Information

```
report_model_statistics –workdir <work directory>
```

- Model statistics indicates the number of logic gates in the model and the counts for other significant model structures.
- The **report_model_statistics** command prints basic structural information about the logic model, including:
  - Circuit Statistics
  - ATPG Constraints in the model
  - RAM/ROM data
  - Primary inputs

```
report_model_statistics -workdir .
```


cadence

This page does not contain notes.

report_model_statistics Summary

The number of primary I/O pins on the original design (these are the pins on Block 0 of the Hierarchical Model) can be calculated by adding the number of Input Only Primary Inputs, the number of Output Only Primary Outputs, and the number of Input/Output pins from either of the two columns.

Adding the Total Inputs and Total Outputs gives the number of primary I/O pins in the Flattened Model, and this counts the Input/Output (bi-directional) pins twice.

The numbers of various kinds of Tied Nets refer to flat model tie blocks. With respect to the hierarchical model, this is the addition of the number of logical nets that are tied and the number of unused pins that become tied in the flat model.

```
INFO (TLM-055): Design Summary
                --------------

Hierarchical Model:                  Flattened Model:
         68,065  Blocks                      30,230  Blocks
        209,618  Pins                        30,230  Nodes
        116,504  Nets

Primary Inputs:                      Primary Outputs:
         11  Input Only                          45  Output Only
         33  Input/Output                        33  Input/Output
         44  Total Inputs                        78  Total Outputs

Tied Nets:                           Dotted Nets:
         74  Tied to 0                           0  Two-State
         46  Tied to 1                          33  Three-State
          0  Tied to X                          33  Total Dotted Nets
        120  Total Tied Nets

Selected Primitive Functions:
          0  Clock Chopper (CHOP) primitives
          0  RAMs
          0  ROMs
         33  TSDs
          0  Resistors
          0  Transistors
      1,889  MUX2s
          2  Latches
  1,526  Rising  Edge Flop w/Set-Dominant and Reset Port
      1,526  Total Flops

      8,155  Technology Library Cell Instances

[end TLM_055]
Optimization removed logic for 7 of 89 cells in this design.

Optimization removed a total of 5,948 tied Latch Ports.
Optimization removed a total of 7,466 non-controlling inputs.
Optimization removed a total of 24,744 dangling logic nodes.
```


cadence

This page does not contain notes.

report_model_statistics Summary (continued)

The Flattened Model section lists a number of blocks and nodes. The flat model blocks include each primitive, dot, primary input, and primary output. The nodes include each block (primitive or dot) output, primary input, and primary output. Note that the difference between the number of blocks and the number of nodes is only in the multiple-output primitives.

Dotted Nets are the number of dot blocks in the flat model, which is the same as the number of multi-source logical nets in the hierModel, including nets connected to bi-directional I/O pins.

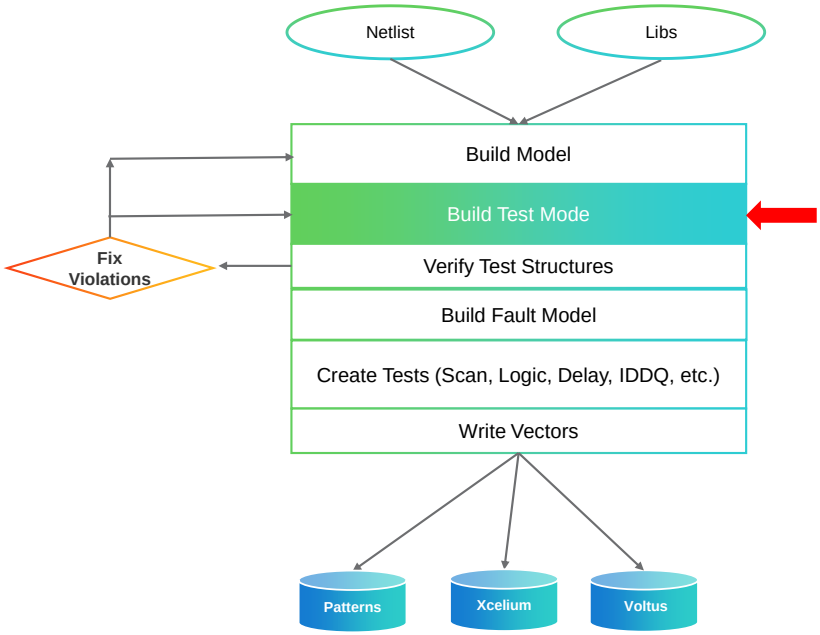
The total number of RAM outputs and ROM outputs is calculated by subtracting the number of flat model blocks from the number of flat model nodes and adding the number of RAMs and ROMs.

```
INFO (TLM-055): Design Summary
                --------------

                                        Flattened Model:
        68,065  Blocks                          30,230  Blocks
       209,618  Pins                            30,230  Nodes
       116,504  Nets

Primary Inputs:                         Primary Outputs:
          11  Input Only                        45  Output Only
          33  Input/Output                      33  Input/Output
          44  Total Inputs                      78  Total Outputs

Tied Nets:                              Dotted Nets:
          74  Tied to 0                          0  Two-State
          46  Tied to 1                         33  Three-State
           0  Tied to X                         33  Total Dotted Nets
         120  Total Tied Nets

Selected Primitive Functions:
           0  Clock Chopper (CHOP) primitives
           0  RAMs
           0  ROMs
          33  TSDs
           0  Resistors
           0  Transistors
       1,889  MUX2s
           2  Latches
   1,526  Rising  Edge Flop w/Set-Dominant and Reset Port
       1,526  Total Flops

       8,155  Technology Library Cell Instances

[end TLM_055]
Optimization removed logic for 7 of 89 cells in this design.

Optimization removed a total of 5,948 tied Latch Ports.
Optimization removed a total of 7,466 non-controlling inputs.
Optimization removed a total of 24,744 dangling logic nodes.
```


cadence

This page does not contain notes.

The ATPG Flow: Build Test Mode



cadence

This page does not contain notes.

What Is Test Mode?

A test mode specifies how the device should be set up for testing. It includes the pins with special functions, the sequences needed for initialization and scanning, as well as the logic that will be active during testing. It also outlines the requirements and restrictions on the tester that will process the data.

- Any change in a test configuration requires the definition of a new test mode.
- The default test modes can be found in:
  - `<$Install_Dir>/tools/tb/defaults/rules/modedef`
- Specific configuration of a design consisting of:
  - How the test structures are accessed and how clocking is controlled
  - Scan chains, clocks, and control pins
  - Initialization, scan, and test sequences (can be customized)
  - Fixed-Value registers, inactive logic, etc.
  - Any change in a test configuration requires the definition of a new test mode

cadence

Let's understand the following information:
- What is a test mode and why do you need one?
- What information do you need to provide to define the test mode?
- What is the syntax of that information?

A Test Mode defines a specific configuration of a design consisting of:
How the test structures are accessed and how clocking is controlled
Scan chains, clocks, and control pins
Initialization, scan and test sequences (can be customized)
Fixed-Value registers, inactive logic, etc.
Any change in a test configuration requires the definition of a new test mode

We start with “What is a test mode and why do I need one?” (aka “What does a testmode do?”)
A test mode defines the test setup for the device. It defines:
- The pins that have special function for test (for example, scan clocks)
- The sequences that are required for initialization and scanning; and
- The logic that will be observable (active) during test (or conversely, the logic that will NOT be observable (inactive) during test).
- The requirements/restrictions on the tester that will process the data

In addition, it may define special types of tests that are required (for example IDDq), or other requirements of the test methodology (for example, True-Time test information).

Test Mode Features

Default test modes include:

- Elastic, Compression, and OPMISR are the Modus compression test modes.
- FULLSCAN test mode: The traditional scan test with scan-in and scan-out pairs.
- Each test mode type can have an OPCG (On-product clock generation) version to allow on-chip clocking.
- 1149 is a scan mode using the IEEE 1149.1 Standard scan protocol:
  - Used for our MBIST, LBIST, and/or Boundary Scan in chip manufacturing.
  - Often used as a parent mode to set up other test modes.

cadence

This page does not contain notes.

Build Test Mode Overview

```
build_testmode -testmode <Name of testmode> -assignfile <pin assign file
with testfunction definitions> -modedef <Test mode definition file name>
-seqdef <sequence definition file>
```

- The `build_testmode` command creates a particular test configuration for use with ATPG.
- The test configuration is based on user inputs.
  - There can be many different test configurations.

```
build_testmode -testmode FULLSCAN  \
-assignfile DLX.working_assignfile
```


cadence

This page does not contain notes.

Build Test Mode Summary

The summary at the end of the building the test mode is show below:

```
--------------------------------------------------------------------------------
*                     Message Summary                                          *
--------------------------------------------------------------------------------
 Count  Number             First Instance of Message Text
------- ------             ------------------------------

  INFO Messages...
      1 INFO (THM-814): Testmode contains 93.47% active logic,  6.53% inactive logic and  0.00% constraint logic.
      1 INFO (TTM-357): There are 16 scan chains which are controllable and observable.
      1 INFO (TTM-387): A default scanop sequence will be generated.
      1 INFO (TTM-391): A default modeinit sequence will be generated.

  WARNING Messages...
      1 WARNING (TTM-347): There is less than 96 percent active logic in this test mode.  Global fault
      1 WARNING (TTM-809): Test mode FULLSCAN has been created, WARNINGs have been generated -

 For a detailed explanation of a message and a suggested user response execute 'msgHelp <message id>'.  For example: msgHelp TDA-009

--------------------------------------------------------------------------------
```


cadence

This page does not contain notes.

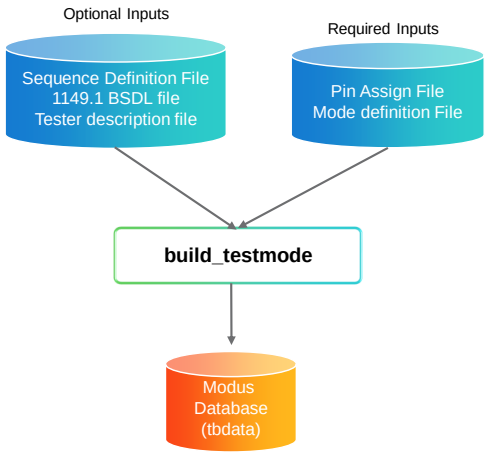
Build Test Mode: Inputs

Required inputs:

- Pin assign file
- Mode definition file

Optional inputs:

- Sequence definition file
- 1149.1 BSDL(Boundary Scan Description Language) file
- Tester description file



The features of the 1149.1 BSDL file are discussed in the Additional Information section

This page does not contain notes.

Required Inputs: Pin Assign File

Pin assignment needs three pieces of information for every pin assigned for ATPG: pin name, pin functionality, and polarity.

- Syntax of the assign file **`assign pin=<pin name> <pin functionality>= <polarity of the pin>;`**
  - The first is the pin name, the second is the pin functionality, and the third is the pin's polarity.
- Example:

```
/************************************************/
/*            Pin assign file file format       */
/************************************************/
assign pin=DLX_CHIPTOP_TEST_ENABLE test_function= +TI;
assign pin=DLX_CHIPTOP_TEST_CLOCK test_function= -ES;
assign pin=DLX_CHIPTOP_TDI test_function= -TI;
assign pin=DLX_CHIPTOP_TCK test_function= +TI;
assign pin=DLX_CHIPTOP_TMS test_function= -TI;
assign pin=DLX_CHIPTOP_TRST test_function= -TI;
assign pin=DLX_CHIPTOP_SE test_function= +SE;
assign pin=DLX_CHIPTOP_RESET test_function= +SC;
assign pin=DLX_CHIPTOP_RESET2 test_function= -SE;
assign pin=DLX_CHIPTOP_SYS_CLK test_function= -ES;
assign pin=DLX_CHIPTOP_DATA[0] test_function=  SI;
assign pin=DLX_CHIPTOP_DATA[1] test_function=  SI;
assign pin=DLX_CHIPTOP_DATA[2] test_function=  SI;
assign pin=DLX_CHIPTOP_DATA[3] test_function=  SI;
```

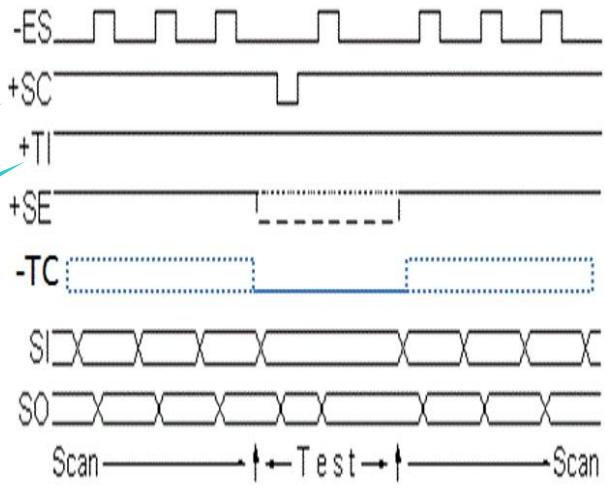

This page does not contain notes.

Example: Polarity and Test Functions

Clocks may or may not be pulsed during capture as determined by ATPG.

SC type clock is selectively pulsed ONLY during capture (async. resets, RAM clocks, etc.)

Control polarity is a value held while active; it may be any value when not active.



57

cadence

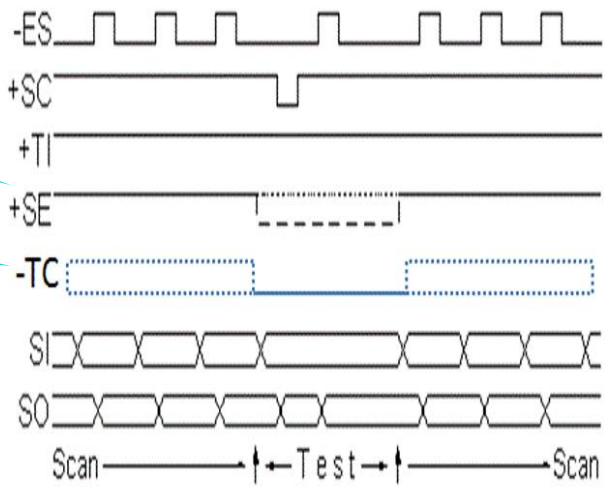
Here is an example of the polarity and test functions when looking at the scan and test states. The scan state is defined as when the design under test is in the process of being scan loaded or unloaded. The test state (sometimes called Test Constraint state) is after scan has been loaded and system level clocks can be used.

Example: Polarity and Test Functions (continued)

Control polarity is a value held while active; it may be any value when not active.

To hold a control value during capture, use the TC (Test Constraint) flag.

You can have multiple test states applied to a pin. For example: test_function=-TC,+SE; This says to set the pin to 1 during shift and 0 during capture.



58

cadence

This page does not contain notes.

Required Inputs: Mode Definition File

- Mode Definition (ModeDef) file defines the scan style, type of testing, and optionally the test pin assignments
- Default modedef files provided (without pin assignments) for default test modes.
  (<$Install_Dir>/tools/tb/defaults/rules/modedef)

```
/************************************************/
/*    Default Modedef file for FULLSCAN Mode      */
/************************************************/
Tester_Description_Rule = dummy.tdr
                ;
scan            type = gsd
                boundary=no
                in = pi
                out = po
                ;
test_types      dynamic timed early (0,0,1) late (0,0,1)
                logic signatures no
                dynamic scan shift_register
                static macro
                ;
faults          static, dynamic
                ;
test_function_pin_attributes C_TEST, C_MODE, CT_TEST, CT_MODE
                ;
```


cadence

In this example, lets look at the mode definition for the FULLSCAN testmode. This is one of the testmodes included with the product.

The tester description rule is named dummy.tdr

The scan type is Generalized Scan Design – our tool supports both edge-sensitive scan and level-sensitive scan, or a mixture of the two. GSD is the type of scan used for edge-sensitive scan or a mixture of scan.

Boundary=no means we did not select Reduced Pin Count Test (a technique for testing a device that has more pins than the tester).

IN = PI and OUT=PO means we are doing normal scan from a scan-in primary input to a scan-out primary output. If the PI/PO was set to “ON_BOARD” it would indicate we were doing BIST where the input stimulus comes from a Pseudo-Random Pattern Generator (PRPG) on the product and the responses are gathered by a Multiple Input Signature Register (MISR) on the product. Setting PO=TO_MISR would indicate we were using test compression where the input stimulus comes from primary inputs, but the responses are collected by a MISR embedded in the design.

Optional Inputs: Sequence Definition File

TBDpatt files contain test patterns (experiments) in ASCII form. **TBDseqPatt** files are a specialized form of TBDpatt files containing sequence definitions in ASCII form and a subset of the statements used in TBDpatt files.

- To produce a TBDpatt file from an existing Modus experiment, use the report_vectors command.
- Sequence definition files have two major functions.
  - Define the mode initialization to be applied to the design.
    - Usually, a design under test requires an initialization sequence; such a sequence can be defined with a seqdef file, also referred to as a **Modeinit** sequence.
  - Defining the scan protocols.
    - The Sequence Definition (**SeqDef**) file can also be used to override default clock/control/data sequences (scan entry, scan cycle, scan exit, test, etc.)
      - Override depends on sequence naming conventions.
      - Uses readable test pattern (**TBDPatt**) format.

cadence

This page does not contain notes.

Example: Sequence Definition File

Common Events

- Stim_PI – stim a primary input
- Pulse – pulse a clock pin
- Force – place an internal value on something
- Release – release forced value

```
TBDpatt_Format (mode=node, model_entity_form=name);
[Define_Sequence test_setup (modeinit);
// set all pins to initial states
   [Pattern ();
      Event Stim_PI ():
         "DLX_CHIPTOP_TEST_ENABLE" = 1
         "DLX_CHIPTOP_TCK"         = 0 ;
   ]Pattern;
// set TMS low and clock 1 time
   [Pattern ();
      Event Stim_PI (): "DLX_CHIPTOP_TMS" = 1 ;
      Event Pulse   (): "DLX_CHIPTOP_TCK" =+ ;
   ]Pattern;
]Define_Sequence;
```

Single Semi-colon ends Events

Pulse clock rising

cadence

This page does not contain notes.

Optional Inputs: Tester Description Rule (TDR)

The Tester Description Rule (TDR) is used to identify the capabilities of the manufacturing tester.

- Defines the characteristics of the ATE necessary to generate patterns that will work within the constraints of the ATE:
  - Number of tester pins available
    - Full Function
    - Scan Clock
    - Parametric Measurement Units
  - Memory depth behind tester pins
  - Timing constraints of pins (Frequency, Pulse width)
  - Handling of tester and on-chip termination
  - HighZ and IddQ measurement
- The default TDR files provided are in `<$Install_Dir>/tools/tb/defaults/rules/tdr`

The Tester Description Rule (TDR) is used to identify capabilities of the manufacturing tester. Tester features can be specified; for example, number of pins, size of scan buffer, tester termination of outputs and bidirectional pins. These characteristics are observed by Modus when building the test patterns from test generation. This tester-specific knowledge allows Modus to both effectively utilize the capabilities of the test equipment and to supply some of the necessary tester set-up information.

Example: Dummy Tester Description File (TDR)

Generic (“dummy.tdr”) supplied with installation of Modus, example:

```
TEST_PINS
        FULL_FUNCTION_PIN_LIMIT = 4096
        STORED_PATTERN_DEPTH = 16000000
        CLOCK_PINS = 64
        SCAN_IN_PINS = 256
        OSCILLATORS = 4
        STORED_PATTERN_SCAN_DEPTH =
1000000000
        PARAMETRIC_MEASURE_UNITS = 4100;
```

```
PIN_TIMING
        TIMING_RESOURCE = SHARED_RESOURCE
        CLOCKS=5 NON_CLOCKS=13 PO_STROBES=7
        MAX_PULSES = 1
        MAX_STIMS = 1
        MAX_MEASURES = 1
        MAX_CYCLE_TIME = 1 MS
        MIN_CYCLE_TIME = 2500 PS
        MAX_PULSE_WIDTH = 1 MS
        MIN_PULSE_WIDTH = 1 NS;
```


cadence

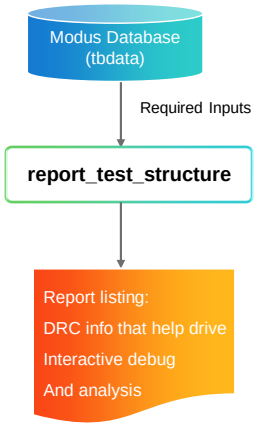
This page does not contain notes.

Reporting the Test Structures

```
report_test_structures -testmode <Name of testmode> -reportscanchain
<all/summary/no> \ ... Other report options
```

- The **report_test_structures** command reports test and circuit information based on a particular test mode.
- Some examples: Scan-chain structure, fixed-value FFs, pipelines, test pin info, clock affiliations, and design info.
- **Example**

```
report_test_structures \
  -testmode FULLSCAN \
  -reportscanchain all
```



This page does not contain notes.

Report Test Structures Log Outline

report_test_structures Log Outline	Information
Circuit Summary	● Model Hierarchical / Flattened ● Primary Inputs / Outputs ● Nets Tied / Dotted ● Selected Primitive Functions
Information for Test Mode	● Scan Type

Controllable/Observable Scan Chain Information for Test Mode	● Number of Scan Changes Controllable/Observable
	● Longest Scan Chain Length
	● Average Scan Chain Length
	● For each scan chain
	  ○ Controllable scan chain identifying information
	  ○ Observable Scan Chain identifying information

cadence

This page does not contain notes.

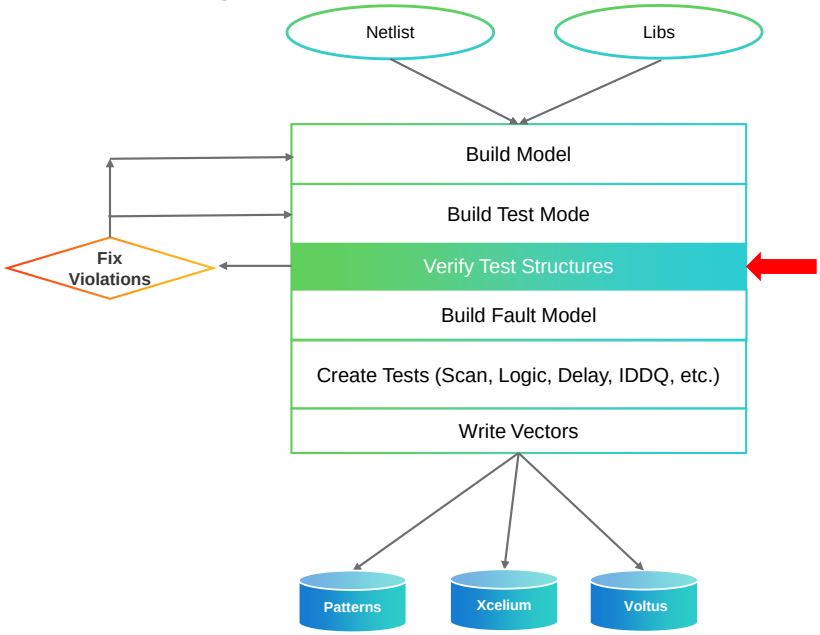
Additional Help and Topics

- Build Test mode Concepts:
  - Modus: Guide2: Testmodes
- Pin Assignment descriptions can be found in:
  - Modus: Guide2: Testmodes –> Identifying Test Function Pins
- Sequence Definition descriptions can be found in:
  - Modus: Guide2: Testmodes -> Sequence Definition File
  - Modus: Reference: Test Pattern Formats -> Sequence Definition Application Objects
- TDR Details:
  - Modus: Guide2: Testmodes -> Tester Description Rule File Syntax

cadence

This page does not contain notes.

The ATPG Flow: Verify Test Structure



cadence

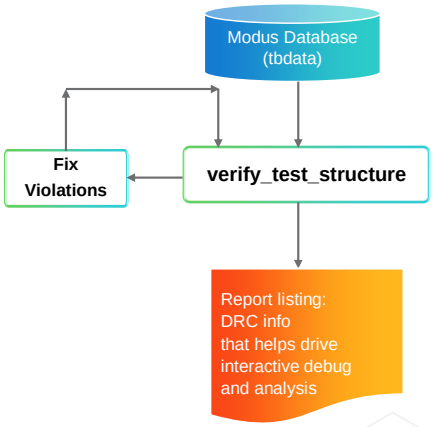
This page does not contain notes.

Verify Test Structure Overview

This step will run the entire design rule checking for your circuit. The types of checks you run are totally user-controlled.

The **`verify_test_structures`** command analyzes scan-based design rules for proper shifting and capturing. Verify test structures:

- Identify scan elements and verify their controllability and observability.
- Identify logic structures that may cause manufacturing test problems.
- Identify logic structures that cause test generation problems.



This page does not contain notes.

verify_test_structures

```
verify_test_structures -testmode <Name of testmode> -workdir <dir>
... Other verification options
```

The **verify_test_structures** command checks everything from:

- clock race conditions
- tri-state contention
- broken scan chains
- X-state propagation

Example

```
verify_test_structures -testmode
FULLSCAN -workdir .
```


cadence

This page does not contain notes.

Verify Test Structures Summary

```
--------------------------------------------------------------------------------
*                     Message Summary                                          *
--------------------------------------------------------------------------------
 Count  Number             First Instance of Message Text
------- ------             ------------------------------

  INFO Messages...
      1 INFO (TLM-055): Design Summary
      1 INFO (TSV-068): The length of the longest scan chain is 85 bit positions, which is 101% of the average scan chain len
gth 85 (based on 1348 total scan chain bits and 16 valid scan chains).
     16 INFO (TSV-378): Scan chain beginning at 'pin DLX_CHIPTOP_DATA[0]' and ending at 'pin DLX_CHIPTOP_DATA[16]' is control
lable and observable. The length of the scan chain is 85 bit positions.
      1 INFO (TSV-567): There are 16 controllable scan chains fed by Scan In (SI) primary inputs.
      1 INFO (TSV-568): There are 16 observable scan chains feeding to Scan Out (SO) primary outputs.
      1 INFO (TSV-569): There are 0 controllable scan chains fed by on-product Pattern Generator(s).
      1 INFO (TSV-570): There are 0 observable scan chains feeding to on-product Multiple-Input Signature Register (MISRs).

      1 INFO (TSV-900): verify_test_structures processing has started Fri Nov 17 19:53:31 2023
      1 INFO (TSV-908): verify_test_structures processing complete.

  WARNING Messages...
      1 WARNING (TSV-163): Scan chain flop or latch block DLX_CORE.PC_REG.STORED_VALUE_reg23_24.__iNsT2.dff_primitive changes
 value during the same scan cycle as scan chain flop or latch block DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_907.__iNsT1.
dff_primitive. The upstream and downstream flops or latches are changing on the same scan cycle but the clocks are in differe
nt domains which may cause the scan chain to fail to shift properly.
      1 WARNING (TSV-390): There are 316 inactive (non-scan) latches.

 For a detailed explanation of a message and a suggested user response execute 'msgHelp <message id>'.  For example: msgHelp
TDA-009

--------------------------------------------------------------------------------
```


cadence

This page does not contain notes.

verify_test_structures Log Outline

verify_test_structures Log Outline	Information
Verify Test Structures Process Preview	• Key process features enabled/disabled for this run.
Circuit Statistics	• Circuit Summary • Hierarchical and Flat Model block, pin, net, and primitive counts • Test function pin information each test mode
Details for Each Check	• Check performed, its status, and CPU/actual time taken

This page does not contain notes.

Test Performed During verify_test_structure Step

Default Tests	Optional Tests
Clock control of memory element	PRPD and MISRs
3-State contention	Channel inputs
Feedback and Keepers	X-State sources are not observable*
Scan chains	Verify Parent/Child mode compatibility
Clock/data races without mutually exclusive gating	Analyze clock domains
Clock choppers	Analyze On-product controls
Fixed value memory elements	Verify non-contacted PIs are not observable
Latch characteristics	Analyze electronic chip ID registers
	Clock/data races with mutually exclusive gating

All tests can be turned ON or OFF.

* default in compression modes.

cadence

This page does not contain notes.

Additional Help and Topics

Help References

- Verify Test Structures, Boundary Scan, Core Isolation, and Fault Analysis
  - Modus: Guide 3: Test Structures
  - Modus: Guide 4: Faults
- Message descriptions
  - Modus: Reference: Messages
    - TSV -> Test Structure Verification Messages
    - TMI -> Verify Core Isolation Messages
    - TJC - IEEE 1149.1 Boundary Scan Verification Messages
- Using the interactive Schematic Viewer
  - Modus: Reference : GUI

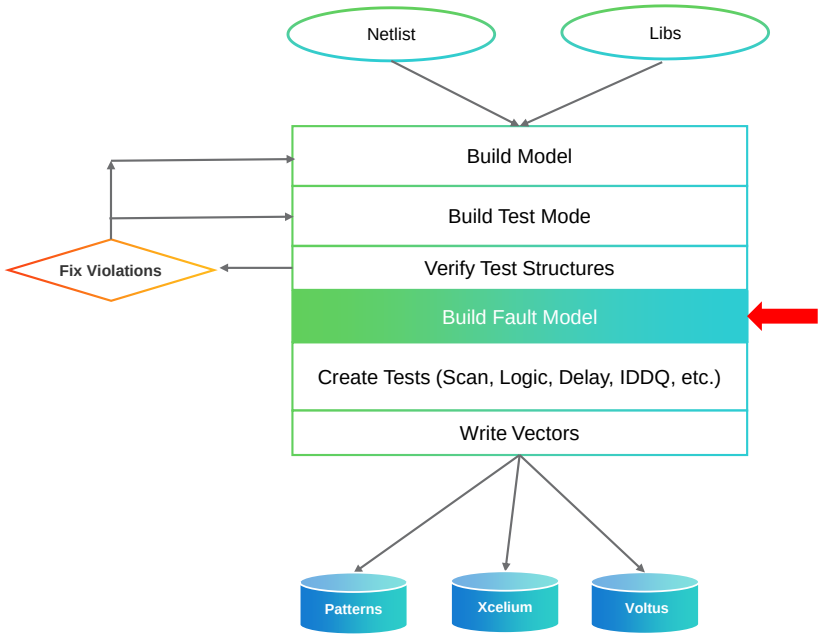
Additional information Section:

- Boundary scan verification

cadence

This page does not contain notes.

The ATPG Flow: Build Fault Model



This page does not contain notes.

Building the Fault Model

```
build_faultmodel -workdir <dir> -includedynamic <yes/no> ... Other fault
modeling options
```

- The **build_faultmodel** command creates a fault database for ATPG processing.
- Some examples of fault types are static, dynamic(transition), iddq, pattern, pin, path, gate-exhaustive, and cell aware faults.
- The recommended flow is to have the test modes already built before building the fault model.

```
build_faultmodel -workdir . \
-includedynamic no
```


cadence

This page does not contain notes.

Build Fault Model Summary

The Summary at the end of **build_faultmodel** in the log file is shown below:

```
--------------------------------------------------------------------------------
*                      Message Summary                                         *
--------------------------------------------------------------------------------
 Count  Number             First Instance of Message Text
------- ------             ------------------------------

  INFO Messages...
      1 INFO (TFM-099): Build Fault Model started.
      1 INFO (TFM-102): Creating faultModel file /home/shubham8/work/JumpStart_LABS/LAB1/tbdata/faultModel.
      1 INFO (TFM-103): Creating faultStatus file /home/shubham8/work/JumpStart_LABS/LAB1/tbdata/faultStatus.
      1 INFO (TFM-109): Build Fault Model has completed with highest level severity message of INFO.
      1 INFO (TFM-704): Maximum Global Test Coverage Statistics:

 For a detailed explanation of a message and a suggested user response execute 'msgHelp <message id>'.  For example: msgHelp
TDA-009

--------------------------------------------------------------------------------
```


This page does not contain notes.

Build Fault Model: Inputs and Outputs

Required inputs:

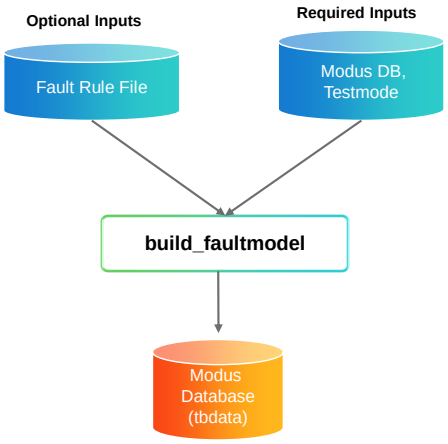
- Modus Database
  - The model needs to have been built first.
- Test Mode
  - The recommended flow is to build all test modes first. Then, build the fault model. There is an update and exchange of information so you can rebuild a test mode after you have built the fault model and still be fine.

Optional inputs:

- Fault Rule File
  - This file is a way of bringing user-customized fault models into the overall fault model. A great example is the bridging fault model.

Output:

- A binary file in the tbdata directory that contains all of the fault information for the design.



cadence

This page does not contain notes.

Optional Input: Fault Rule File Keywords

Keyword Used Inside Fault Rule File	Type of Fault `build_faultmodel -includedynamic yes -faultrulefile myfaultrule1`

NOFAULT keyword:
• Exclude all faults from the specific module using the fault rule file.
• Removes faults entirely from the fault universe.
	Excludes only STATIC faults. Example
	
	`Instance in Module:`
	
	`NoFault STATIC Instance sp_*_reset_reg_0 In Module sys_reset_block`
	`Instance in Cell:`
	
	`NoFault STATIC Instance sp_*_reset_reg_0 In Cell latch_*_upd`
	
	NoFault **STATIC** Block dig_top.u_sys_reset_block.reset_reg_0
	
	`NoFault STATIC Cell SDFFRX1`
	
	`NoFault STATIC Module SDFFRX1`

cadence

This page does not contain notes.

Optional Input: Fault Rule File Keywords (continued)

Keyword Used Inside Fault Rule File	Type of Fault `build_faultmodel -includedynamic yes -faultrulefile myfaultrule1`

NOFAULT keyword:
• Exclude all faults from the specific module using the fault rule file.
• Removes faults entirely from the fault universe.
	Excludes only DYNAMIC faults. Example
	`Instance in Module:`
	`NoFault DYNAMIC Instance sp_*_reset_reg_0 In Module sys_reset_block`
	`Instance in Cell:`
	`NoFault DYNAMIC Instance sp_*_reset_reg_0 In Cell latch_*_upd`
	`NoFault DYNAMIC Block dig_top.u_sys_reset_block.reset_reg_0`
	`NoFault DYNAMIC Cell SDFFRX1`
	`NoFault DYNAMIC Module SDFFRX1`

cadence

This page does not contain notes.

Optional Input: Fault Rule File Keywords (continued)

Keyword Used Inside Fault Rule File	Type of Fault `build_faultmodel -includedynamic yes -faultrulefile myfaultrule1`

FAULTONLY keyword:
• To include faults only for that particular BLOCK or MODULE.
• Will include fault for both the global and test mode coverage.
	Example
	`Instance in Module:`
	`FAULTONLY Instance q_reg[0] In Module five_ff`
	`Instance in Cell:`
	`FAULTONLY Instance sp_*_reset_reg_0 In Cell latch_*_upd`
	`FAULTONLY Block dig_top.u_sys_reset_block.reset_reg_0`
	`FAULTONLY Cell SDFFRX1`
	`FAULTONLY Module SDFFRX1`

cadence

This page does not contain notes.

Optional Input: Fault Rule File Keywords (continued)

Keyword Used Inside Fault Rule File	Type of Fault `build_faultmodel -includedynamic yes -faultrulefile myfaultrule1`

IGNORE keyword:
• To ignore some of the faults modeled on a pin.
• Faults are moved to the ignored list.
	Example
	`IGNORE { PATTERN 2 BLOCK LATCH_p_BLT_DATAI_OUT$OMX104.0000100.master }`
	`IGNORE { SA0 PIN BLTI_p_DRV103.0000101.01 }`
	`IGNORE { SA1 PIN BLTI_p_DRV106.0000101.01 }`

This page does not contain notes.

Build Fault Model Features

Building the fault model is a step that initializes the test generation portions of the Modus database. The features of building the fault model inside Modus include:

- **Single fault model** throughout the ATPG process and prepare a fault list for Modus ATPG fault processing for every type of fault, including:
  - Stuck-at (static) pin faults (stuck-at 1, stuck-at 0)
  - Transition (dynamic) pin faults (slow to rise, slow to fall)
  - Pattern faults (static and dynamic)
  - Path delay faults
- **Modus fault model** optimizes the fault list. Faults
  - Are collapsed
    - Faults that have the same effect on the circuit
  - Are grouped for more efficient test generation
    - For example, Register Arrays where there is known regularity in the logic structure
- **Modus fault model** prepares:
  - Repository of all fault status information
  - Mechanism to "keep score" (multi-mode fault coverages)
  - Permanent pattern set

cadence

This page does not contain notes.

Faults Marking Representation During build_faultmodel

Fault classes are attributed during:

- The process of building the fault model based on design or
- While initializing the fault status for a test mode.

The following table shows the fault classifications possible during the Build fault model.

Fault Class	Fault Representation	Fault Classification During BTM	Collapsed Faults on Parent Classification
Ignored Fault	I	Ignored – unconnected – (Iu) Ignored – Blocked logic (Ib) Ignored – Tied Logic (It) Ignored – dynamic SE/SR (Id) Ignored – User fault rule (If) Ignored – Unclassified (I)	cIu (collapsed Ignored unconnected) cIb (collapsed Ignored blocked) cIt (collapsed Ignored tied) cId (collapsed Ignored dyn SE/SR) cIf (collapsed Ignored user FRULE) cI (collapsed Ignored Unclassified)
Active Fault	a	a	ca (collapsed active)

cadence

This page does not contain notes.

Faults Marking Representation During build_faultmodel (continued)

Fault Class	Fault Representation	Fault Classification During BTM	Collapsed Faults on Parent Classification
Inactive faults	i	i	ci (collapsed inactive)
Collapsed fault	C	C	cC (collapsed)
Possibly testable at best faults (PTAB)	3 P3 X PX C PC	3 = PTAB 3-state: Untested P3 = PTAB 3-state: Possibly Tested X = PTAB X-state: Untested PX = PTAB X-state: Possibly Tested C = PTAB Clock Stuck Off: Untested PC = PTAB Clock Stuck Off: Possibly Tested	c3 (PTAB 3-state: Untested) cP3 (PTAB 3-state: Possibly Tested) cX (PTAB X-state: Untested) cPX (PTAB X-state: Possibly Tested) cC (PTAB Clock Stuck Off: Untested) cPC (PTAB Clock Stuck Off: Possibly Tested)
Grouping	& I	AND Grouping OR Grouping	

cadence

This page does not contain notes.

Standard Design Constraints SDC

```
read_sdc -testmode <testmode name> -sdc <sdc file_name>
```

- **read_sdc** is used to read and verify design constraints from an SDC file.
- The SDC file supplements or replaces the SDF for delay tests.
  - It is expected that you will get the SDC from your timing tool rather than trying to create one manually.

```
read_sdc -testmode FULLSCAN \
-sdc mysdc
```


This page does not contain notes.

Important Information Messages After Reading the SDC

Here are a few important messages to check the status of your SDC file, whether it got accepted or rejected:

- **`INFO (TDC-001): Read and Verify SDC Process Started.`**
  - Explanation: Indicates that Read and Verify SDC process has started.
- **`INFO (TDC-005): Done reading the SDC file(s).<verificationString>`**
  - Explanation: Reading of SDC file was successful, and now the verification process will start if verification is turned on.
- **`INFO (TDC-006): <verificationString>A total of <numConstraints> design constraints<middleString> will be honored in ATPG.`**
  - Explanation: Verification process is finished. Only the verified constraints will be honored during ATPG. If verification is turned off, all constraints will be honored during ATPG.
- **`INFO (TDC-017): No design constraints are presently registered.`**
  - Explanation: No SDC constraints were found. Run read_sdc to register constraints.

cadence

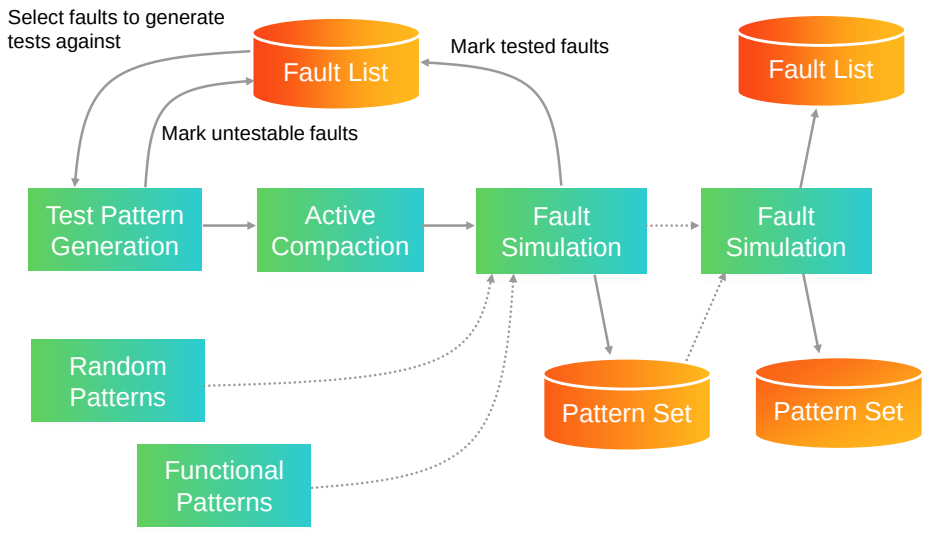
This page does not contain notes.

Design Constraints Attributes Description

SDC Constraints	Description
set_case_analysis	Identifies points in the design to hold at a known state.
set_false_path	Identifies paths that are expected to be inactive in the functional mode; checking is not required. There are two types of false paths: ● Paths through which transitions (including glitches) cannot propagate. ● Paths for which tests are not to be created. A common use is to remove inter-domain paths.
set_multicycle_path	Identifies logic that exceeds one cycle to complete.
set_disable_timing	Removes timing arcs and indicates the logic can be don't care, set to X, for ATPG and simulation.
set_clock_groups	Identifies clocking groups that should not be proceeded together.

This page does not contain notes.

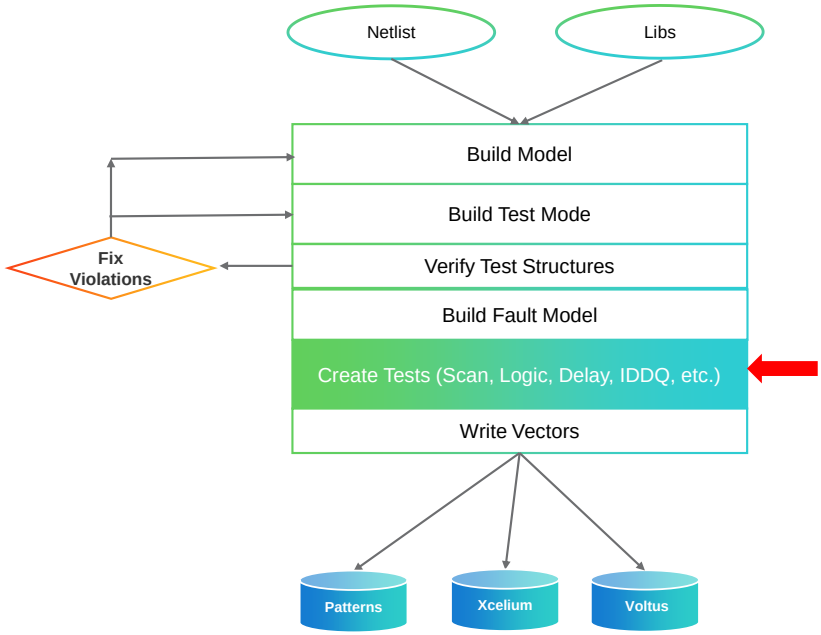
ATPG Test Pattern Creation Flow



cadence

This page does not contain notes.

The ATPG Flow: Create Tests



cadence

This page does not contain notes.

How to Perform Dynamic Fault Testing

To perform dynamic fault testing, we need to excite the transition on the fault and capture it downstream.

Dynamic fault testing is more complex and processing-intensive than stuck-at-fault tests. This test is performed in two time frames.

1. **First time frame:** Generate transition
2. **Second time frame:** Capture results

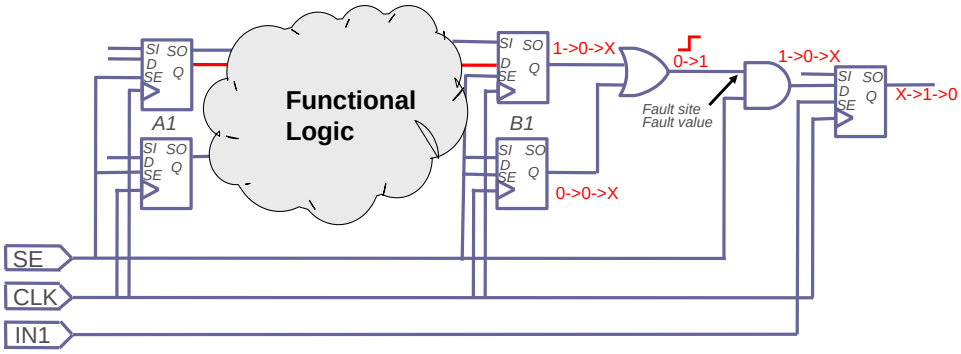
Command line example:

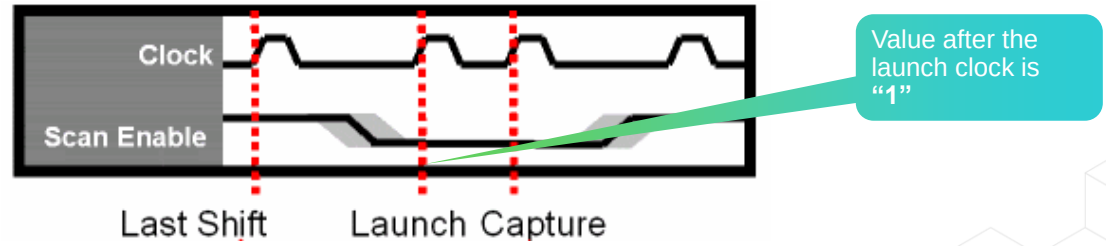
```
create_logic_delay_tests            \
-testmode        <testmode name>     \
-experiment      <an experiment name> \
    ... Other test generation options
```


cadence

Transition test or delay test consists of exciting a transition on the fault site and capturing it downstream. Exciting the transition requires that values are set up in two time frames. This results in expanding the cone of logic that needs to be set up by approximately 2x. Delay test ATPG software will have to make more decisions and keep track of more circuit values than a standard stuck fault ATPG tool.

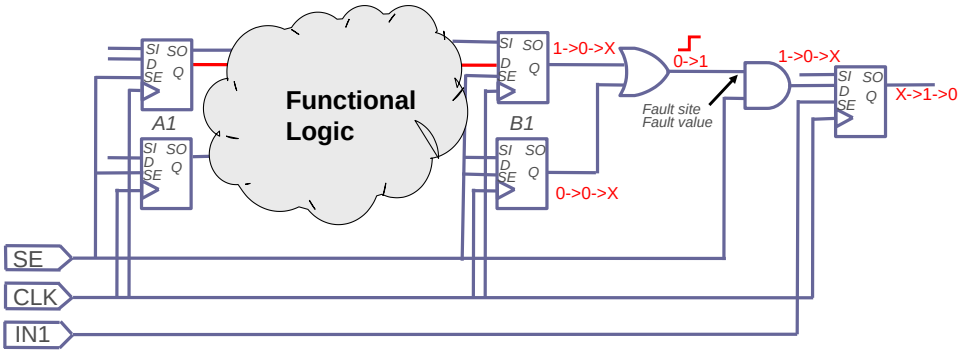
Step 01: Generate Transition

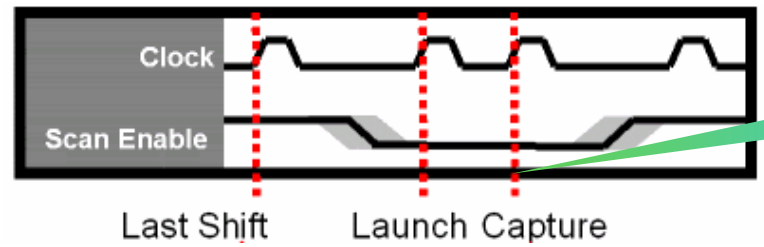




This page does not contain notes.

Step 02: Capture Result





92

cadence

This page does not contain notes.

Create Other Types of ATPG Tests

Command	Description
create_scanchain_tests	• Scan chain tests • Scan chain tests are automatically generated while running *create_logic/logic_delay_tests*
create_sequential_tests	• Sequential tests • Generate tests with sequential depths, especially for designs with some non-scan flops
create_bridge_tests	• Bridging fault static tests • Requires specific fault model
create_bridge_delay_tests	• Bridging fault dynamic tests • Requires specific fault model
create_iddq_tests	• IDDQ tests • Tests for IDDQ faults, targeting high-quiescent current
prepare_cellaware_faults	• Cell-aware fault model to model potential open, short, and transistor-specific defect sites and identify cell-level test patterns

cadence

This page does not contain notes.

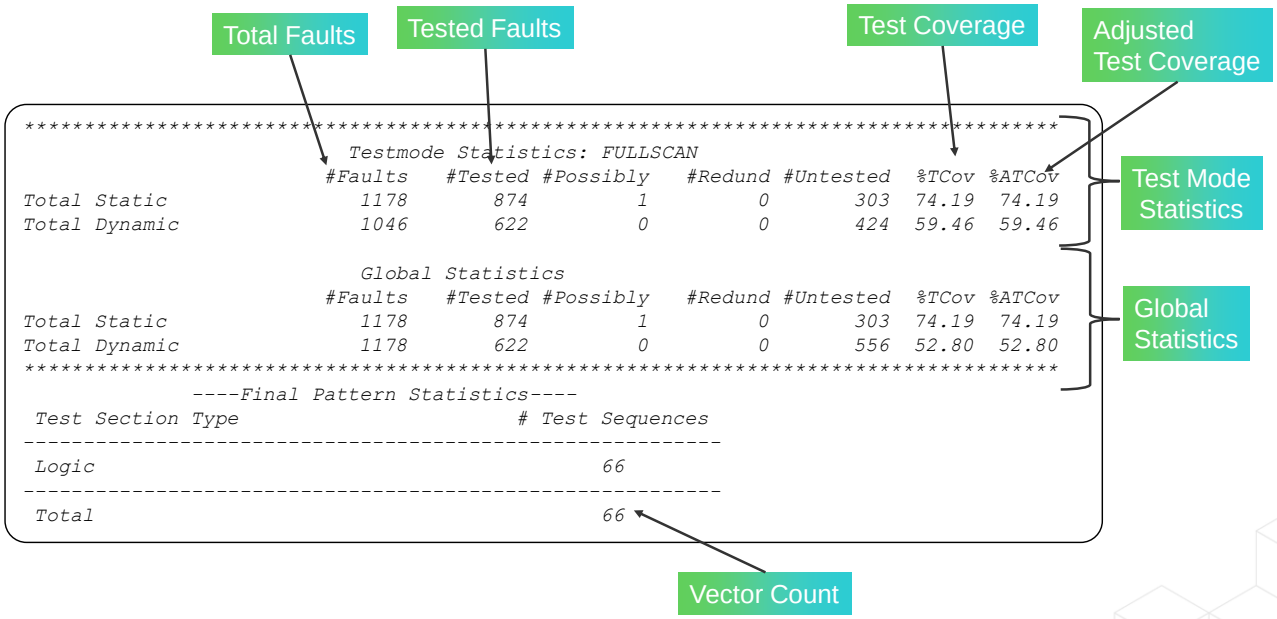
Create Other Types of ATPG Tests (continued)

Command	Description
create_logic_tests	• Modus uses ATPG to create static tests. • In Modus, we generate a scan chain test, to verify the scan chains are working prior to applying tests and logic tests, which are traditionally known as stuck-at fault tests.
create_logic_delay_tests	• Generate delay tests with or without actual timings.

cadence

This page does not contain notes.

ATPG Statistics: Fault Coverage and Pattern Count




```
*******************************************************************************
                        Testmode Statistics: FULLSCAN
                      #Faults   #Tested #Possibly   #Redund #Untested  %TCov  %ATCov
Total Static             1178       874         1         0       303  74.19  74.19
Total Dynamic            1046       622         0         0       424  59.46  59.46

                        Global Statistics
                      #Faults   #Tested #Possibly   #Redund #Untested  %TCov  %ATCov
Total Static             1178       874         1         0       303  74.19  74.19
Total Dynamic            1178       622         0         0       556  52.80  52.80
*******************************************************************************
             ----Final Pattern Statistics----
 Test Section Type                      # Test Sequences
-------------------------------------------------------
 Logic                                  66
-------------------------------------------------------
 Total                                  66
```


This page does not contain notes.

Test Coverage Calculation

There are different types of calculation by which we calculate the test coverage and global test coverage:

- Standard Modus calculations:
  - $Test\ Coverage = \frac{Number\ of\ detected\ Faults}{Total\ \ Faults}$
  - $Adjusted\ Test\ Coverage = \frac{Number\ of\ detected\ Faults}{Total\ \ Faults - Redundant\ faults}$
- Global vs. Testmode Test Coverage
  - Global test coverage includes all faults in the design minus the ignored faults
  - Test mode test coverage includes only global faults that are active in the test mode
- Non-standard calculations can be made:
  - Extract data from the report logfiles
  - TCL scripts

cadence

Example of the types of test coverage statistics that are provided.

Line Hold a Pin/Net During Test Generation

```
<Line Hold Type> <object> "object_Name" = <value"
```

- To control a specific pin or net of a design, a line hold file should be created and used when running ATPG.
- Logic is back-traced, and values are justified to meet the line-hold constraint.

```
# Examples of Line Holds
HOLD PIN "Pin.f.l.newpart.nl.lat1.LATCH_2.P01DATA"=1 ;
HOLD PIN"lat1.LATCH_4.P01DATA"= 1;
HOLD BLOCK "lat1.LATCH_6"=0;
HOLD NET "ioblock.weaken.output" = L ;
```

When running ATPG, add **-linehold <filename>**
Example:

```
create_logic_tests \
        -EXPERIMENT    <name> \
        -TESTMODE      <name> \
        -linehold     <linehold file name>
```


cadence

This page does not contain notes.

Ignore Flip-Flops and Primary Inputs During Test Generation

To ignore certain flip-flops and primary inputs, a text file with their names should be created and used when running ATPG.

- The flip-flop name can be the full or short name.
- Just a simple list of elements where no measures
- Flip-flops can still be scan loaded to a value.
- **Example**

```
#Ignore Measures File
# List of Flops / Primary Inputs.

Block.f.l.topcell.nl.c1.b4.f2
c1.b4.f2
PinName
```

- When running ATPG add **-ignoremeasures <filename>**
  - **Example**

```
create_logic_tests \
        -EXPERIMENT    <name> \
        -TESTMODE      <name> \
        -ignoremeasures <ignore_measure_file name>
```


cadence

This page does not contain notes.

Reporting Fault List

```
report_faults -testmode <testmode name> -experiment <an experiment name> \
-faultstatus <all/tested/possible/redundant and so on>  \
-hierstart <top/string/integer>
```

The **report_faults** command lets the user report the faults with the test status.

- Global fault list and testmode/experiment-specific fault list can be reported.
  - Specify a testmode and experiment to report faults of an experiment.
- By default, untested faults are reported.
  - Use **-faultstatus** option to list faults with a specific status.
- Specify **-hierstart** to report faults of a specific block.

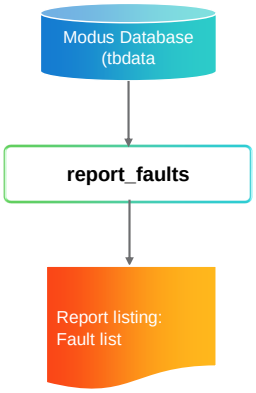
```
report_faults -testmode FULLSCAN \
-experiment LOGIC -faultstatus tested \
-hierstart top
```


cadence

This page does not contain notes.

Reporting Fault List (continued)

- Reporting of ignored/inactive fault categories:
  - Use "`-ignored yes`"
  - Use "`-inactive yes`"



cadence

This page does not contain notes.

Reporting Fault Statistic

Display fault statistics for a given experiment or pattern set.

- Type of report
  - static, dynamic, parametric, or Iddq
- Coverage credit
  - tested, redundant, possibly, etc.
- Figures can be reported hierarchically
  - `-hierstart top|<string><integer> blocks to be reported`

```
report_fault_statistics\
-testmode       <testmode name> \
-experiment     <an experiment name>\
-hierstart      <top/string/integer>
```


cadence

Here is the command line and location of report_fault_statistics . This command can list the coverage statistics for a given set of patterns.

Faults Categorization of ATPG

Fault Classification Category After ATPG	Sub-categories
Tested (T)	Tested (T), Tested by Simulation (T), Tested by Implication (Ti), Tested by Possibly Detected Limit (Tpdl), Tested in Another Mode (Tm), Tested User Specified (Tus), Possibly Tested (P)
Untested (u)	Untested – Not Processed (u), Untestable- Undetermined (Ud), Untestable- User Specified (Uus), Untestable- Linehold inhibits fault control/observe (Ulh), Untestable- X-sourced or sinked (Uxs), Untestable- Seq or control not specified to target fault (Unio , Unil, Unlo, Unra, Uner), Untestable- Testmode inhibits fault control/observe (Utm), Untestable- Sequential depth (Usd), Untestable- Global termination (Ugt), Untestable: Constraint (Ucn)

cadence

This page does not contain notes.

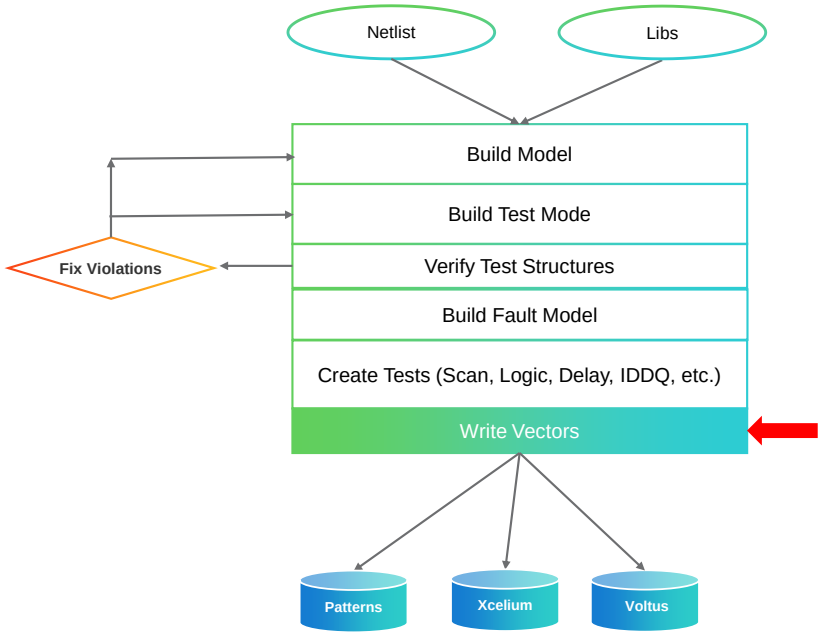
Faults Categorization of ATPG (continued)

Fault Classification Category After ATPG	Sub-categories
Redundant (R)	Redundant (R), Redundant: User Specified (Rus)
Aborted (A)	Atpg –untestable (A)

cadence

This page does not contain notes.

The ATPG Flow: Write Vectors



cadence

This page does not contain notes.

Writing Vectors

```
write_vectors -testmode <testmode name>   \
-inexperiment <experiment if uncommited> -language <verilog/stil/wgl/tdl> \
-scanformat <serial/parallel(applicable to verilog)
... Other write_vctors options
```

The **write_vectors** command writes out industry standard formats of the patterns previously generated.

```
write_vectors -testmode FULLSCAN \
-language verilog -scanformat parallel\
-testrange 1:100
```

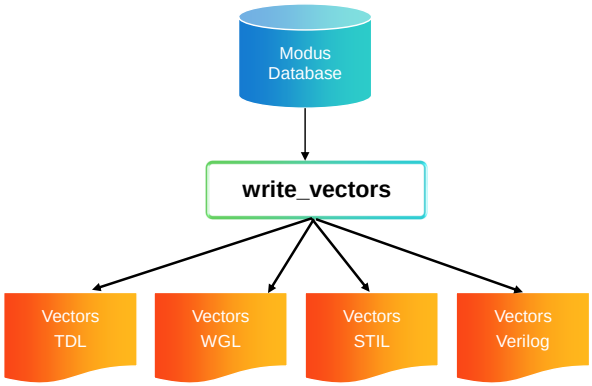

cadence

This page does not contain notes.

Write Vectors Directory

Modus, by default, outputs the Verilog test patterns into the **<WORKDIR>/testresults/verilog directory**

- Directory created by **`write_vectors`** command contains the testbench file (which must be compiled), one scan pattern file, logic test files (needed during simulations), and cyclemap file.
- Output formats of the generated vectors are Verilog, WGL, STIL, and TDL.



cadence

This page does not contain notes.

Example: Write Vector Pattern Files

All the pattern file names begin with "`VER.<testmode>.data.<pattern_type>.`" Pattern types can be either scan or logic.

- Scan confidence patterns are placed into a file with "scan" as part of the name.
- Logic patterns are placed into a file with "logic" as part of the name.
- **Example:** `write_vectors -testmode FULLSCAN -scanformat parallel -inexperiment logic`
  - VER.FULLSCAN.data.scan.ex1.ts1 // Scan chain test patterns
  - VER.FULLSCAN.data.logic.ex1.ts2 // First logic test pattern set
  - VER.FULLSCAN.data.logic.ex1.ts3 // Second logic test pattern set
  - VER.FULLSCAN.mainsim.v // testbench
  - Cyclemap.FULLSCAN.data // Cyclemap file

A complete set of the scan, logic, mainsim testbench file, and cycle map file is generated for each test mode.

cadence

This page does not contain notes.

Write Vectors: Pin Timing Attributes

Tester specific timings are specified in the **write_vectors** command. However,

- WGL, STIL, TDL, and Verilog vectors will have default timings based on test and scan cycles. Modifying testperiod/scanperiod will auto-adjust the other default timings.
- Users can modify the default timings through the GUI or by using the following options:

Scan Cycle Timing Options	Capture Cycle Timing Options
-scantimeunits	*-testtimeunits*
-scanperiod	*-testperiod*
-scanpioffset	*-testpioffset*
-scanpulsewidth	*-testpulsewidth*
-scanstrobeoffset	*-teststrobeoffset*
-scanstrobetype	*-teststrobetype*
-scanbidioffset	*-testbidioffset*
-scanpioffsetlist	*-testpioffsetlist*
-releasebidioffset	*-capturebidioffset*

cadence

The vectors for use in manufacturing are written in industry standard formats. In addition, the vectors may be written in proprietary formats used by certain partner foundries.

What Is Experiment “Commit”?

The commit experiment updates the global fault list to mark as detected fault if any faults detected in the design.

- All test generation runs are made as uncommitted tests in a test mode.
- Experiment commit moves the uncommitted experiment test results into the committed vectors test data.
- Marks faults as tested are updated in the global status.
  - No further ATPG runs will target the faults that are marked tested.
- Once the experiment is done, patterns are saved in testmode data, and the experiment is removed.

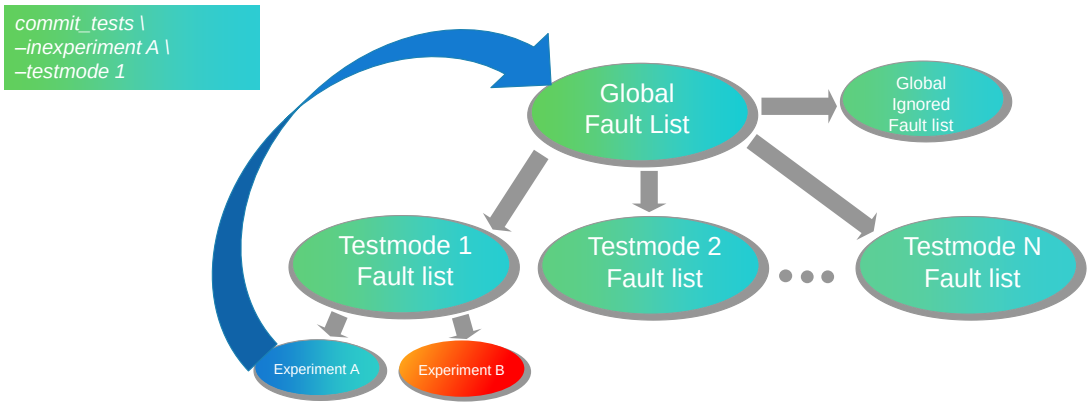
cadence

This page does not contain notes.

Experiment “Commit” Flow

As shown below,

If Experiment A runs first and is committed, Experiment B will start with a detected coverage that includes Experiment A.



cadence

This page does not contain notes.

commit_tests

```
commit_tests\
-testmode          <testmode name> \
-inexperiment        <an experiment name>
```

- The **commit_test** command updates the global fault list to mark as “detected” any faults detected by the experiment.
- Subsequent ATPG runs will not target the detected marked-off faults.
- The global coverage will be calculated to include the results of all committed experiments, potentially across multiple test modes.

```
commit_tests -inexperiment A \
-testmode 1
```


cadence

This page does not contain notes.

Additional Help and Topics

Help Reference

- Writing and reporting test data
  - Modus: Guide 6 : Test Vectors
- Test generation techniques
  - Modus: Guide 5 : ATPG
- Command line help can be found in:
  - Modus: Reference: Test Pattern Formats

cadence

This page does not contain notes.

Module Summary

In this module, you learned how to

- Identify the steps involved in the ATPG flow
- Build a test model
- Define and create test mode
- Verify the test structure to ensure it is working correctly
- Build a fault model to identify and locate faults
- Perform different types of ATPG tests, such as:
  - Dynamic fault test
  - Scan chain test
  - Sequential test
  - Bridging fault test
  - IDDQ tests
- Write out ATPG vectors to perform tests

cadence

This page does not contain notes.

Labs

Lab 3-1 Perform the Modus ATPG (Automatic Test Pattern Generation) Flow

- Building the Modus Test Model
- Review the Errors and Warnings Inside Modus GUI
- Build the Test Mode
- Report Test Structures
- Verify Test Structures
- Build the Fault Model

Lab 3-2 Generating ATPG Vectors

- Generating ATPG Vectors
- Committing the ATPG Vectors
- Writing Vectors

cadence

This page does not contain notes.

Module 4

Debug Scan Chains with GUI and Tcl Interface

cadence

This page does not contain notes.

Module Objectives

In this module, you will be able to

- Trace the broken scan chains with GUI and Tcl interface
- Debug the different errors and warnings, such as TSV 385, TSV 40X, and TSV 093
- Analyze the Faulty Instance

cadence

This page does not contain notes.

Scan Chains Debugging

There are multiple methods to debug the scan chains. Modus traces scan chains both forward from **scan-in** and backward from **scan-out**. A few recommended steps need to be followed to debug the scan chains backward since it is easier.

- A broken scan chain normally produces two error messages.
  - WARNING **(TSV-384)**: `Controllable scan chain beginning at {objectName} is not a observable scan chain.`
  - WARNING **(TSV-40X for example TSV-401, 402)**: `Observable scan chain ending at {objectName} is not a controllable scan chain.`
  - It is usually easier to debug scan chains working **backward**. Since it is recommended to debug the **TSV-40X** warnings before **TSV-384**.
  - Log file will report the last good Observation flop in the **TSV-40X** message.
- During the `build_testmode` command summary, it will give an indication of broken chains in the following info and warning messages:
  - `INFO (TTM-357): There are <number> scan chains which are controllable and observable.`
  - `WARNING (TTM-358): There are <number> scan chains which are only controllable.`
  - `WARNING (TTM-359): There are <number> scan chains which are only observable.`

cadence

This page does not contain notes.

Submodule 4.1

Scan Chains Debugging with GUI

cadence

This page does not contain notes.

Submodule Objectives

In this submodule, you will be able to

- Identify the different shortcuts of keyboard/mouse to interact with Modus GUI
- Analyze the circuit state, TSV messages analysis, and fault status using different color coding in the schematic view
- Identify the last good flop
- Trace the faulty instance
- Find and analyze the faulty instance in Modus GUI
- Identify the TSV-093 (Three-state contention) error messages

cadence

This page does not contain notes.

Mouse/Keyboard Shortcuts to Interact with Modus GUI

Keyboard Shortcuts	Function
Ctrl+b	View Block
Ctrl+a	View Fault
Ctrl+c	View Core
Ctrl+n	View Net
Ctrl+p	View Pin
Ctrl+I	View PPI
Ctrl+h	View Path
Ctrl+o	View Snapshot
f	Trace forward from a selected block/pin
b	Trace backward from a selected block/pin

cadence

This page does not contain notes.

Mouse/Keyboard Shortcuts to Interact with Modus GUI (continued)

Keyboard Shortcuts	Function
Numpad-5	Center selected or highlighted object, as per the center on object setting
Arrow keys	Scroll the circuit display in the direction of the arrow key
Ctrl+arrow	Scroll the display faster. The arrow keys from the numeric keypad perform in the same manner
Spacebar + Spacebar -	Zoom in Zoom out
Ctrl+x	Display the context window tear-off
Ctrl+v	Display the information window tear-off

cadence

This page does not contain notes.

Mouse/Keyboard Shortcuts to Interact with Modus GUI (continued)

Mouse + Keyboard Shortcuts	Function
Ctrl+wheel	Zoom in/out
Alt+Ctrl+wheel	Zoom in/out Faster
Shift+wheel	Scroll Horizontally
Alt+Shift+Wheel	Scroll Horizontally Faster
Wheel	Scroll Vertically
Alt+Wheel	Scroll Vertically Faster
Ctrl+LMB+Drag (Left to right)	Select area to enlarge (Zoom in)
Ctrl+LMB+Drag (Right to Left)	Fit the entire display (Zoom out)
Ctrl+LMB	Toggle object selection
Shift+LMB+Dral or CTRL+MMB+Drag	Drag the display
Ctrl+Shift+LMB+Drag or Shift+MMB+Drag	Select multiple objects within the selection box

cadence

This page does not contain notes.

Schematic Color Coding in Modus GUI

Normal Circuit States (scan, TC, clocks-off, etc) Color Coding	Color Description
	Inactive Logic
	Inactive logic due to cut points
	Clocks
	Normal data pin
	PPI data pin
	Pin on scan path
	Pin selected
	Block selected

cadence

This page does not contain notes.

Schematic Color Coding in Modus GUI (continued)

Fault Status	Color Description
	Tested
	Aborted
	Untestable
	Unprocessed
	Inactive

cadence

This page does not contain notes.

Schematic Color Coding in Modus GUI (continued)

Diagnostic Analysis Mode	Color Description
	Faults that best explains the failures
	Faults that do not explain failure but possible counter examples
	Some faults explained but with many possible counter examples

TSV Messages Analysis Modes	Color Description
	Pins in violation
	Clock pins in violation

cadence

This page does not contain notes.

Analyzing the TSV Warnings

Let's debug the broken scan chains in the Interactive GUI.

We first need to analyze the TSV warning to trace the broken scan chains so that the last good flop appears in the schematic window.

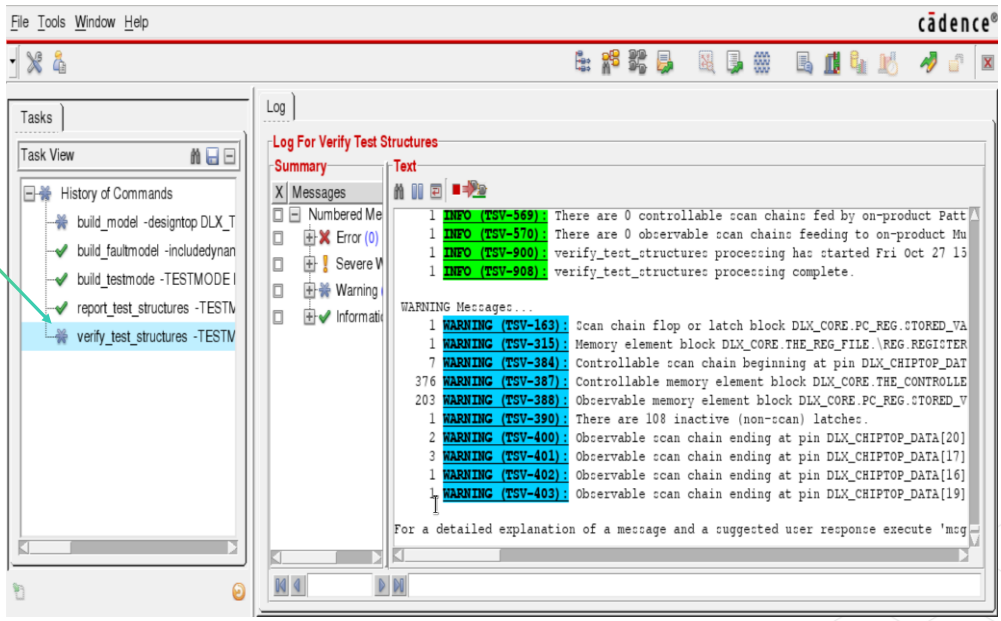
1. Review **`verify_test_structure`**
2. Set analysis context
3. Review the errors and warning messages
4. Bring up the last correct flop

cadence

This page does not contain notes.

Step 01: Reviewing the `verify_test_structure` Command Summary

Click on the verify_test_structures run in the modus GUI and review the Error and Warning Messages.

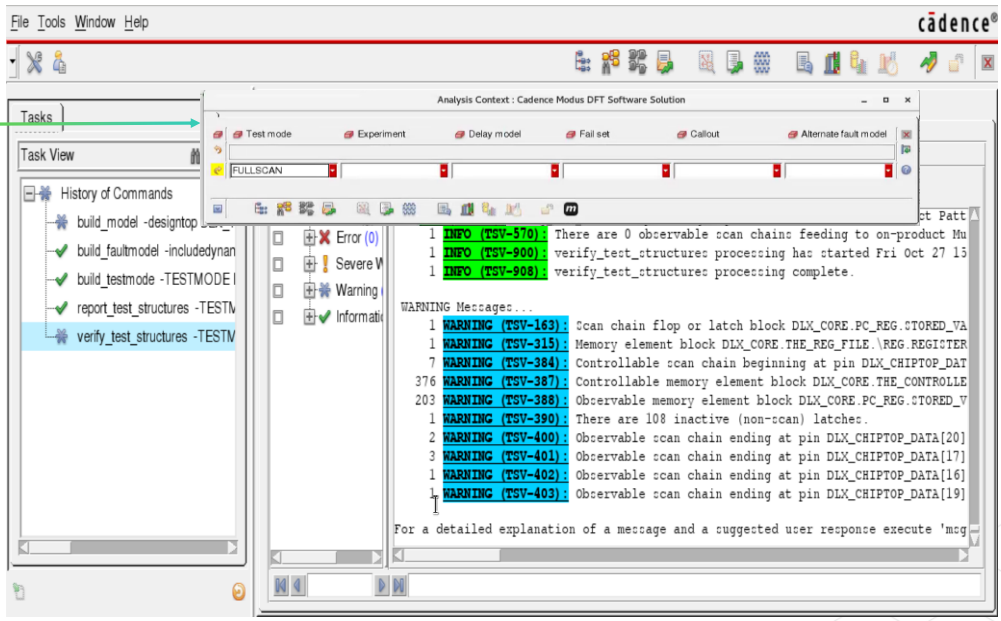


cadence

This page does not contain notes.

Step 02: Set Analysis Context

You can see what the context is by clicking on the “Arrow/Pencil” icon in the upper right-hand corner of the GUI or the pulldown menu **Window** -> **Analysis Context**.



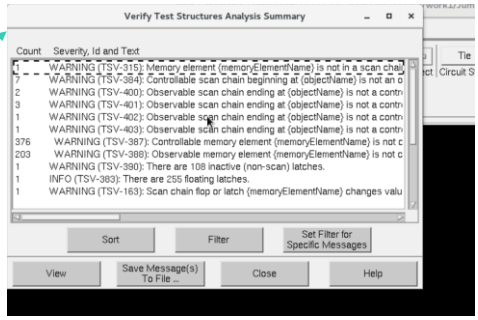
cadence

This page does not contain notes.

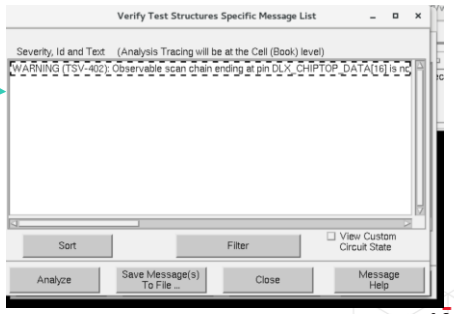
Step 03: Reviewing the Errors and Warnings Messages

Once the context is set, click the view messages icon. This will open the new messages window, containing the TSV message summary.

Search for TSV-384 and several TSV-40X (for exampleTSV-401 TSV-402) messages; these messages indicate the broken chains.



Select any of the messages in the **Verify Test Structures** Summary window and click **View**. This should bring up another window with 1 individual instance of message.



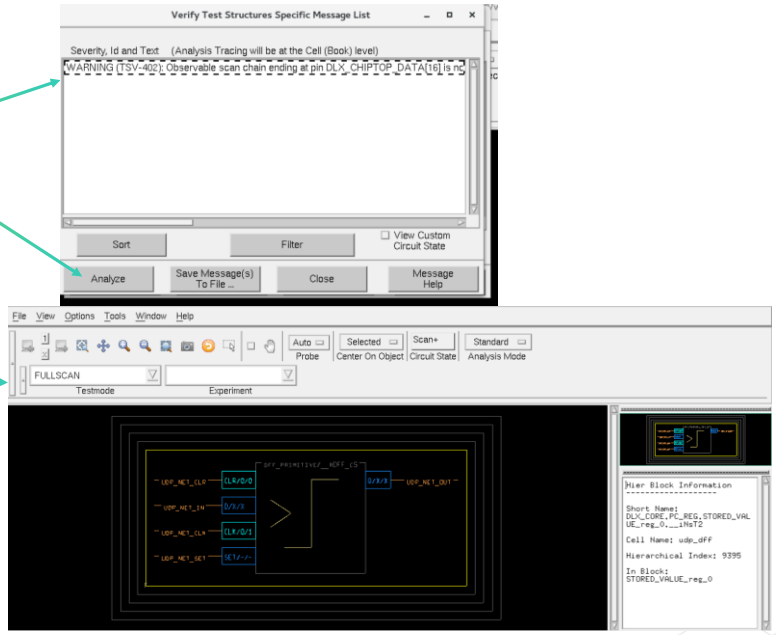
cadence

This page does not contain notes.

Step 04: Bring Up the Last Correct Flop

Select the message in the Verify Test Structure Specific Message List and click **Analyze**. This will bring the schematic window to the top and will navigate to the last good cell it encountered in the analysis.

Schematic window with last good cells in the broken scan chain in schematic window.



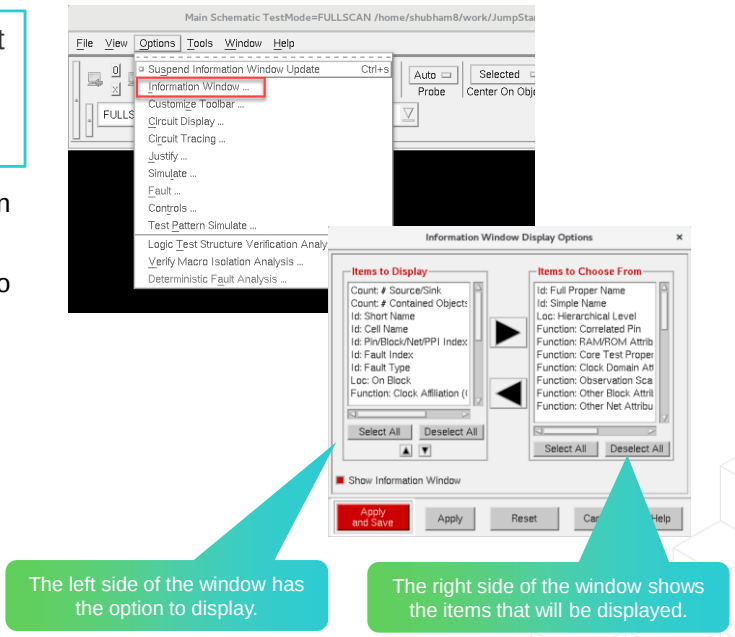
cadence

This page does not contain notes.

Recommended Items for Information Window

In the main Schematic Window (circuit display window), choose **Options –> Information Window**.

- The information window in the Circuit Display can be customized and reordered.
- Recommended Items to choose from right side to left side:
  - **Function: Scan Path Data**
    - Shows info on scan chain path
  - **Function: Flop/Latch Scan Data**
    - Shows flops scan bit position
  - **Function: Clock Affiliation**
    - Clock information



The left side of the window has the option to display.

The right side of the window shows the items that will be displayed.

cadence

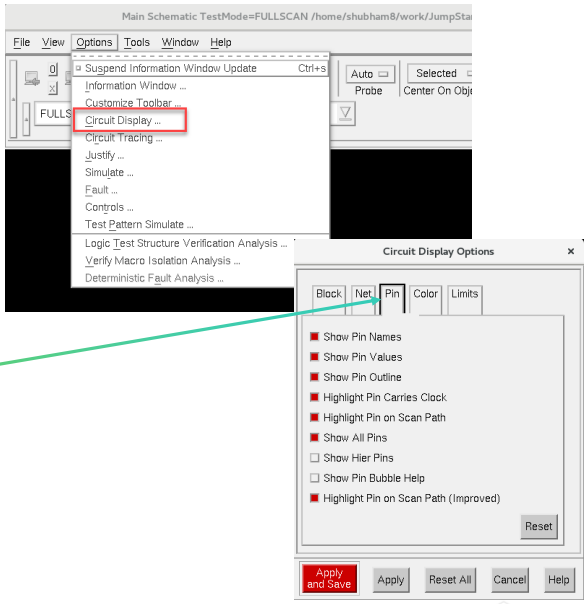
This page does not contain notes.

Circuit Display Window in GUI

In the main Schematic Window (circuit display window), choose **Options –> Circuit Display**.

The way the net/pin/hierarchy information is displayed in the Circuit Display can be customized.

- Options for blocks, nets, and pins can be set.
- Recommendation:
  - Apply all options under the Pin Tab.

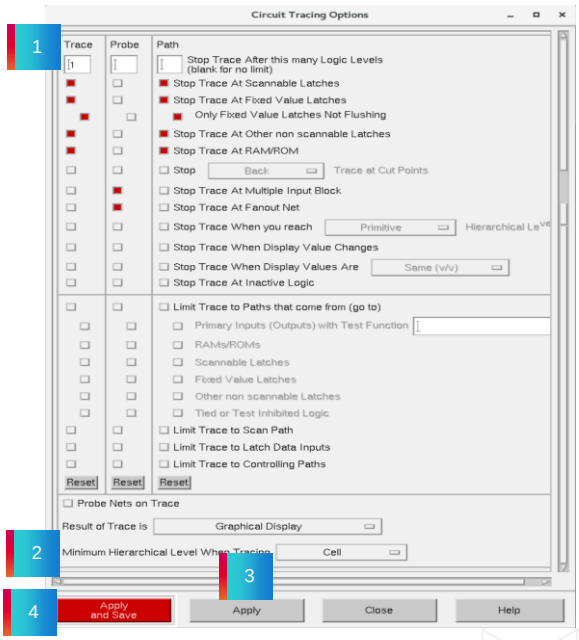


The look of the blocks and cells in the circuit display can be customized. Here are the steps to do this and some recommend items.

Circuit Tracing Options in GUI

To customize some of the circuit tracing options, in the schematic window from the pulldown menu, choose **Options -> Circuit Tracing…**

1. Trace Option
   - Control the # of levels of logic to trace back and what logic the tool stops at.
2. Minimum hierarchical level when tracing option
   - Control tracing level to primitive, cell, or macro level.
3. Apply
   - Apply" changes for the current session.
4. Apply and Save
   - "Apply and Save" changes for all future sessions.



cadence

This page does not contain notes.

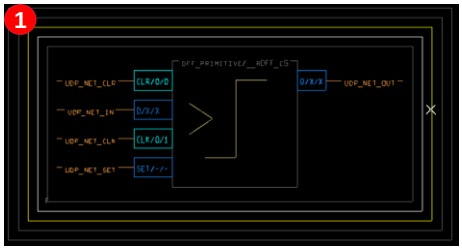
How to Move Up to the Next Hierarchy Level

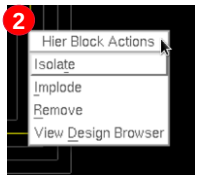
For broken chain debug, the instance displayed is the last correct scan flop before the break in the chain.

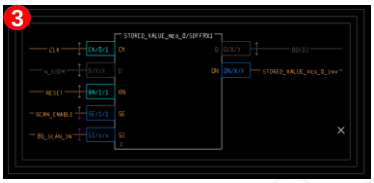
- If only the top-level pin pops up, then tracing never returned to the scan flip flop driving the pin.
- If it is a flop, it is the last working flop.

You can “collapse” the instance to move up to the next hierarchy level by using these steps:

1. **Left**-click to select the block.
   - Rectangle highlighted in white is selected.
2. **Right**-click to open the “Block Actions” menu.
3. Click **Implode**.
   - These above steps might need to be done more than once until the library-level cell is found.
   - The instance cell type and instance name are shown in the right-hand “Information Window.”



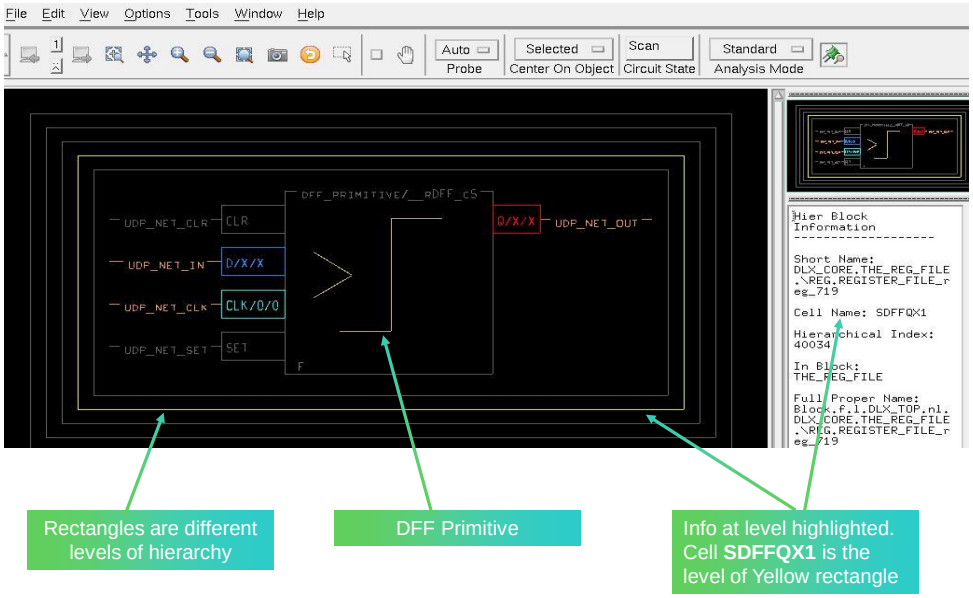




cadence

This page does not contain notes.

Instance Hierarchy and Information Window Display



This page does not contain notes.

Correct Values of Flop in Schematic and Information Window

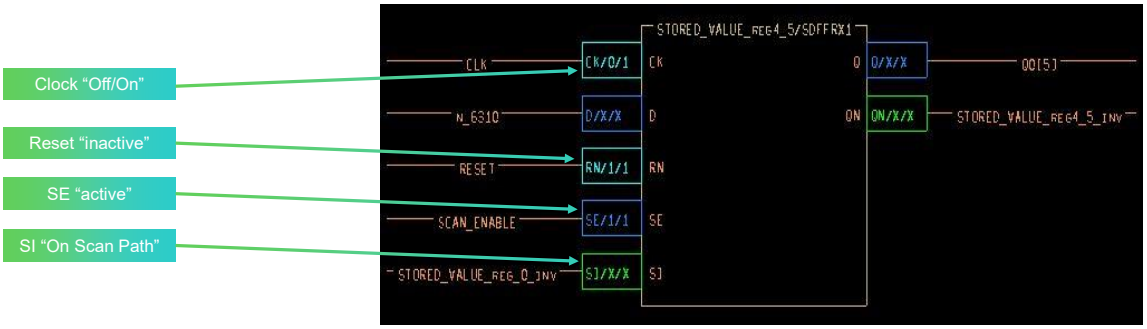
While tracing the scan chain, we should always observe the typical pin values for a correctly functioning scan flop in the schematic window and the information window. The correct pin values of each pin for a scan flop are mentioned as follows.

- The **clock pin** will show a defined off-state.
  - 0 or 1 is acceptable; X is an error.
- The **asynchronous set** or **reset** pin will show a valid off-state.
  - 1 in the case of active low reset, X would be an error.
- The **scan-enable** pin is active.
  - 1 in the case of an active high enable (+SE).
- The **data-in** and **scan-in** pins are undefined.
  - This is normal during scan chain trace and is not an error.
  - They will typically show an X value, which means it can be any value.
  - If this pin or any pin shows a lowercase x, that means it is undriven.

cadence

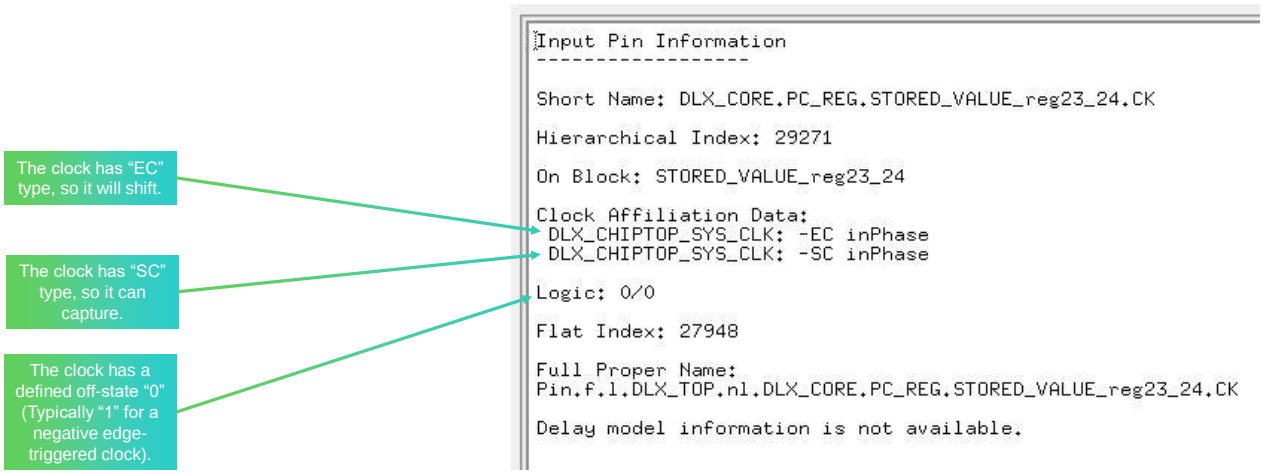
This page does not contain notes.

Correct Scan Flop Values in Schematic Window



This page does not contain notes.

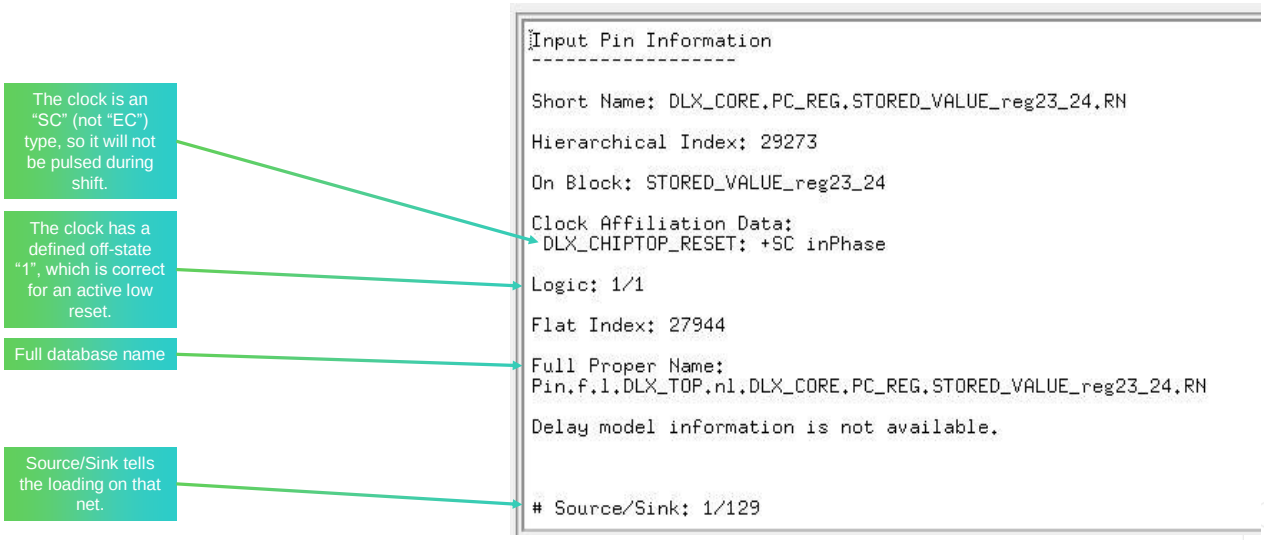
Correct Scan Clock Values in the Information Window



cadence

This page does not contain notes.

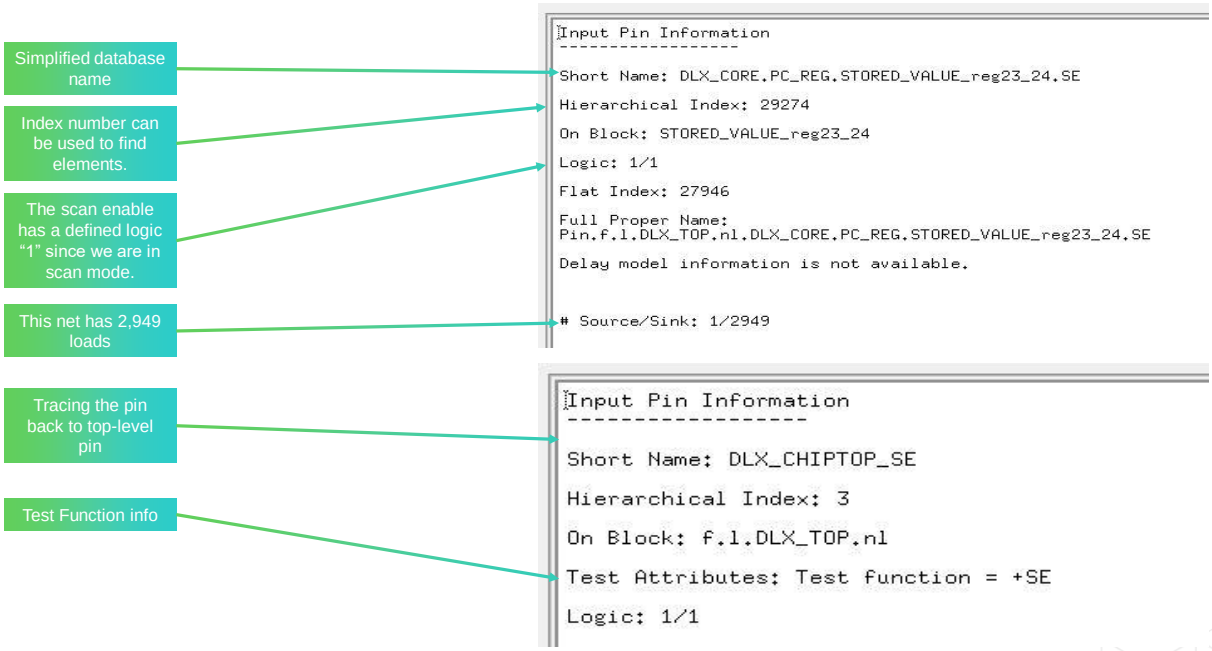
Correct Asynchronous Reset Values in the Information Window



cadence

This page does not contain notes.

Correct Scan Enable Values in the Information Window



cadence

This page does not contain notes.

Finding the Faulty Instance

The analysis of the TSV message will show the last “correct” instance the tool found:

- If tracing a broken Observable scan chain, you need to trace backward from the SI pin of the flop, usually just to the next scan flop in the chain.
- If tracing a broken Controllable chain, you need to trace forward from the SO pin of the flop to the next scan flop in the chain.

1. Selecting the SI/SO pin
2. Opening the Pin Action menu
3. “Trace Backward” or “Trace Forward”

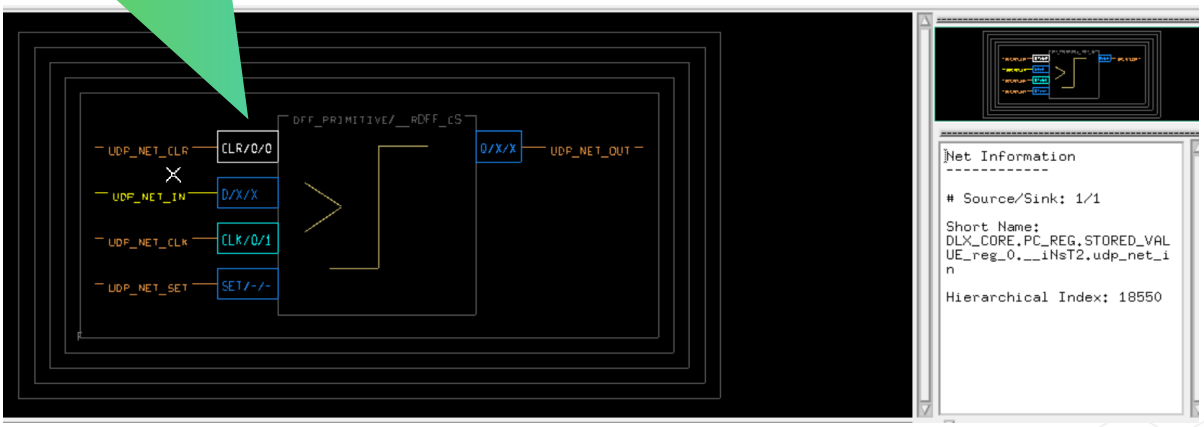
cadence

This page does not contain notes.

Step 01: Selecting the SI Pin of the Last Correct Flop

SI pin has been selected by clicking the “left mouse button” on the pin of the last good observed flop.

1. Selecting the SI/SO pin
2. Opening the Pin Action menu
3. “Trace Backward” or “Trace Forward”



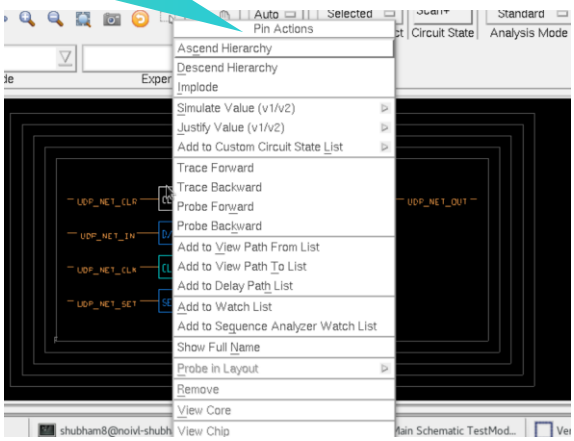
cadence

This page does not contain notes.

Step 02: Opening the Pin Action Menu

"Pin Action" menu to trace forward or backward; in this case, we will trace backward in the next step.

1. Selecting the SI/SO pin
2. Opening the Pin Action menu
3. "Trace Backward" or "Trace Forward"

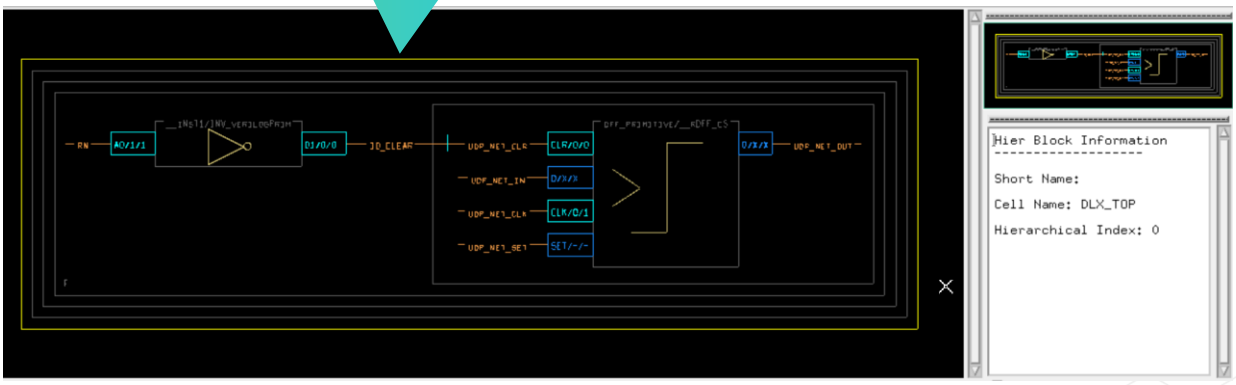


This page does not contain notes.

Step 03: Tracing Backward to Reach to the Faulty Instance

After “Tracing Backward”, we reached the faulty instance and now have to analyze it.

1. Selecting the SI/SO pin
2. Opening the Pin Action menu
3. “Trace Backward” or “Trace Forward”



This page does not contain notes.

How to Analyze the Faulty Instance

While debugging the broken scan chain, either forward or backward tracing, you reach the faulty instance. Now, it's time to analyze the faulty instance and know the fault reason.

"Implode" the library cell level

Place the cursor over each input pin and check the status

cadence

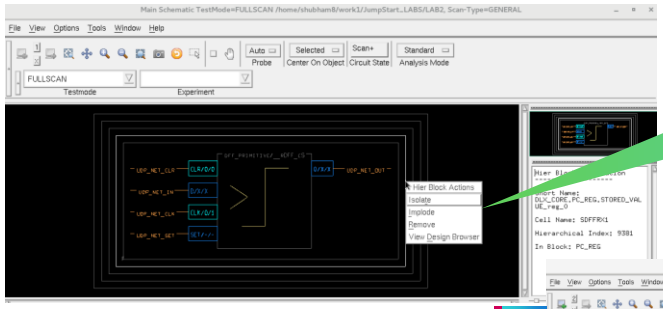
This page does not contain notes.

Step 01: Imploding to the Library Cell

"Implode to the library cell"

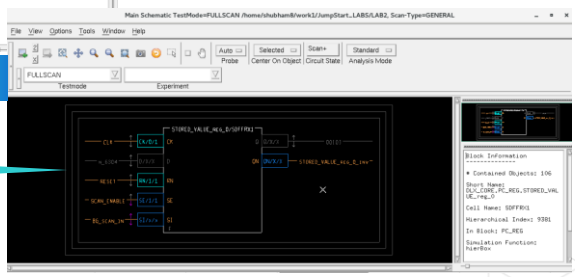
Place the cursor over each input pin and check the status

1



1. Imploding the instance by selecting the rectangular box, opening the "Block Action" Menu, and clicking "**Implode**".

2



2. Instance after imploding.

146 © Cadence Design Systems, Inc. All rights reserved.

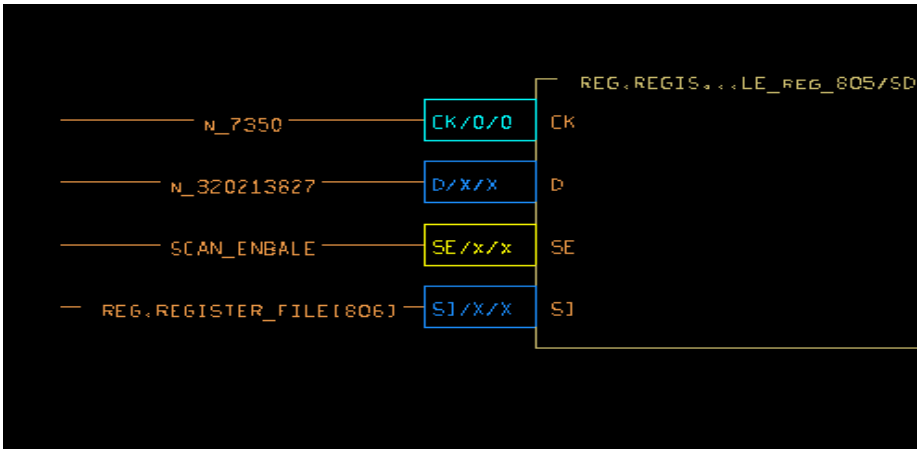
cadence

This page does not contain notes.

Step 02: Placing the Cursor Over Each Input Pin of Instance

"Implode to the library cell"

Place the cursor over each input pin and check the status



cadence

This page does not contain notes.

Analyzing the Faulty Instance Input Pins

Once you find the faulty instance, it's time to analyze and debug it by moving the cursor over each input pin of the instance and observing the values of each pin. The pin values of the instance are discussed below.

- Is **Scan** related pin values okay?
  - Is the scan-enable pin at the correct logic state for shifting?
  - Does the scan-in pin show an "X" state? (This is correct).
  - Does the scan-in pin show an "x" (lower case).? If so, then the input is undriven and faulty.
  - Does the scan-in pin show a fixed value: "0", "1", "+," or "-"? If so, then this will break the chain.
- Are **Clock Pin** values okay?
  - Is the pin set to it's defined "off-state"?
  - Is the pin set to an "X" or "x"?
  - Does the pin have a proper "Clock Affiliation" in the information window?

cadence

This page does not contain notes.

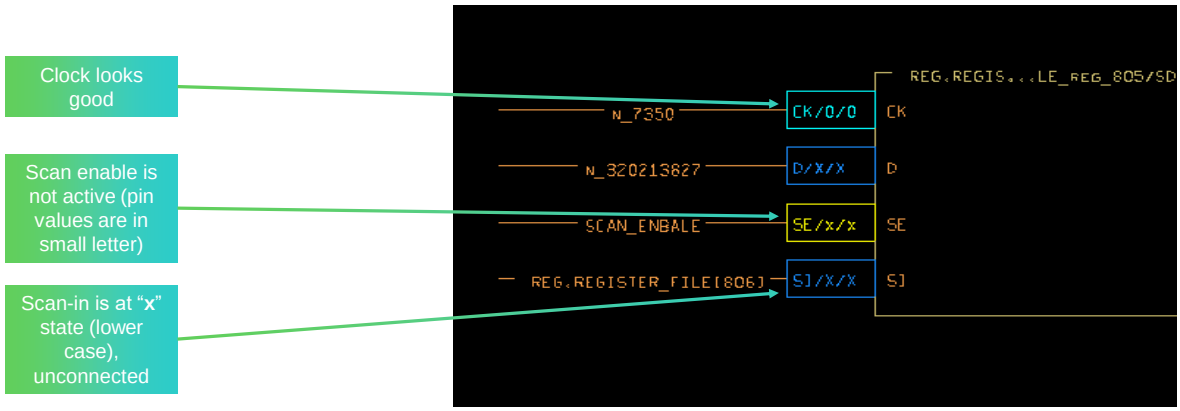
Analyzing the Faulty Instance Input Pins (continued)

- **Set/Reset pin(s)** values are okay?
  - Is the pin tied to its proper inactive value?
  - If defined as an SC clock, is it set to the proper “off state”?

cadence

This page does not contain notes.

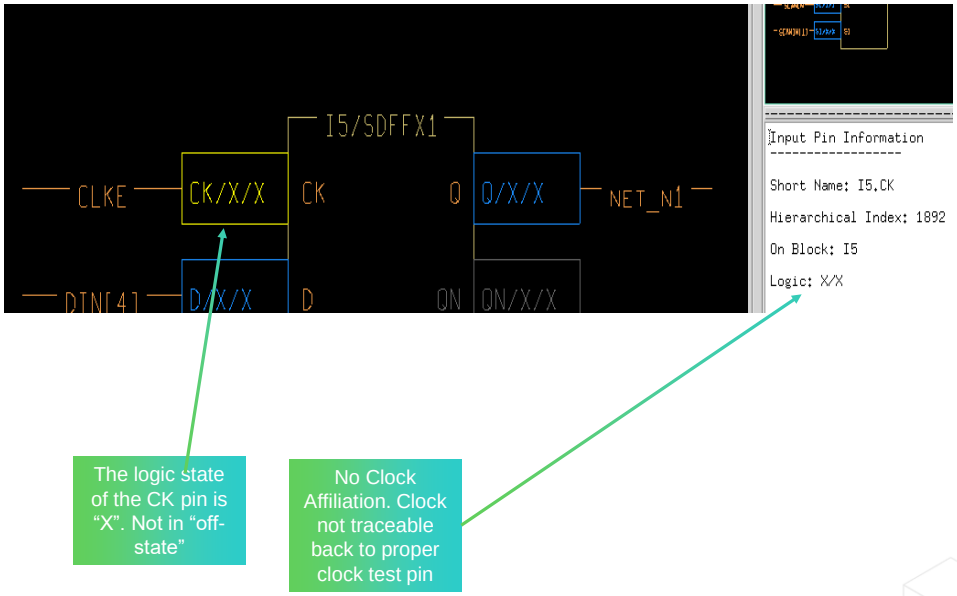
Bad Scan Enable Pin in Faulty Instance



cadence

This page does not contain notes.

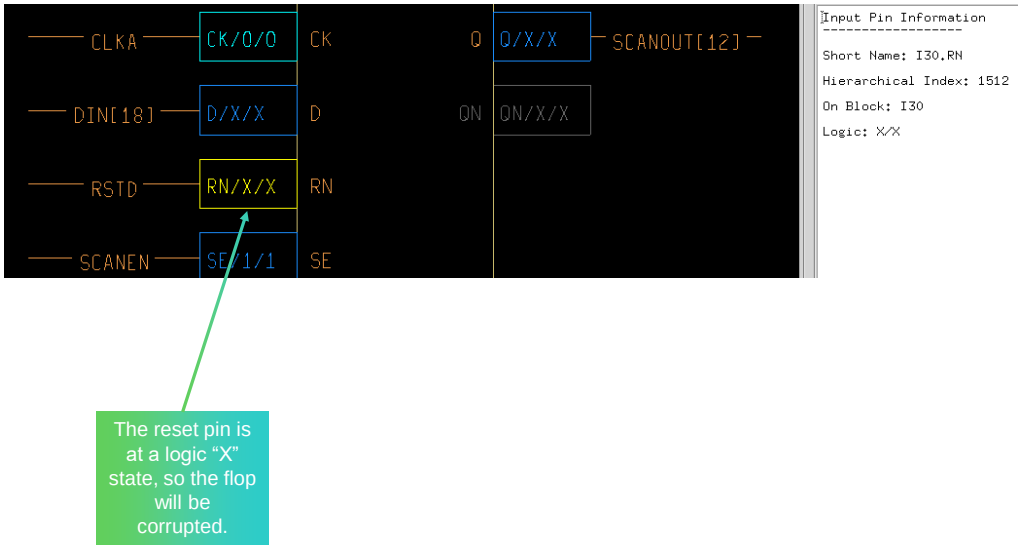
Undefined Clock in Faulty Instance



cadence

This page does not contain notes.

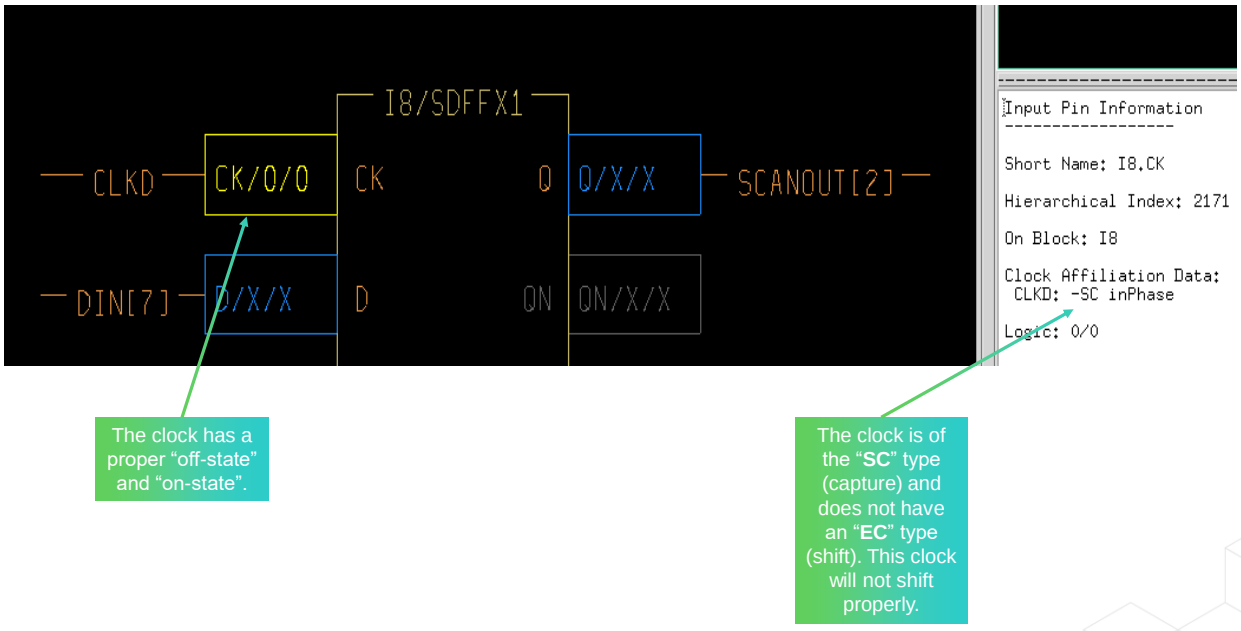
Uncontrolled Reset in Faulty Instance



cadence

This page does not contain notes.

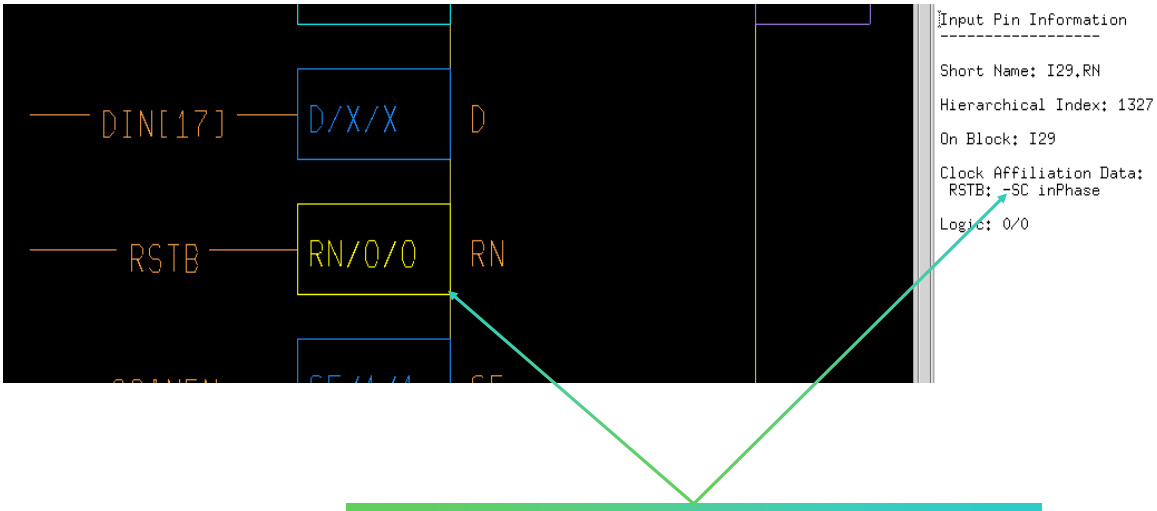
Incorrect Clock Type in Faulty Instance



cadence

This page does not contain notes.

Wrong Reset Clock Polarity in Faulty Instance

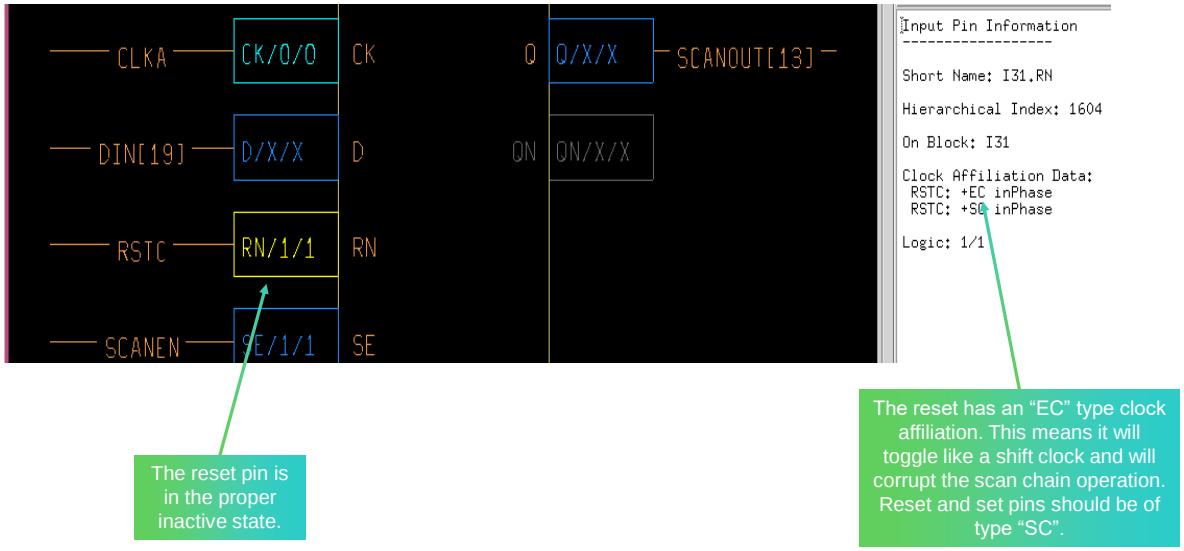


The RSTB signal is active low at 0, which means reset is active in this case because of the "**–SC flag**" in the Pin assign file.

cadence

This page does not contain notes.

Incorrect Clock Type on Asynchronous Reset in Faulty Instance



cadence

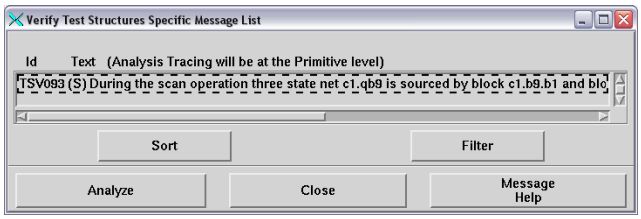
This page does not contain notes.

TSV-093 Warning

Verify Test Structures → WARNING (TSV-093): `During the scan operation scanSectionName three state netName is sourced by sourceName1 and sourceName2, which may simultaneously drive opposite values, resulting in 3-state contention`

TSV-093 says that contention occurs during the scan operation. A very serious problem with the possibility of creating burn-out patterns.

- Three-state contention can also result in excess power consumption as well as possible damage to the product and invalid signatures.



Solution: Analyze the message and use the mouse pointer to identify the individual blocks, pins, and nets comprising the paths.

- To correct the deviation, install logic that prevents both blocks feeding the net from being simultaneously driven to opposite values.

cadence

This page does not contain notes.

TSV-193 Warning

WARNING (TSV-193): [severity] In the Test Constraint state three state netName is sourced by sourceName1 and sourceName2 which may simultaneously drive opposite values resulting in 3-state contention

TSV-193 says that contention occurs during the capture operation. This could create a damaging pattern.

- The Modus applications assume that the scan operation is free of three-state contentions.

Solution: Analyze the message and use the mouse pointer to identify the individual blocks, pins, and nets comprising the paths.

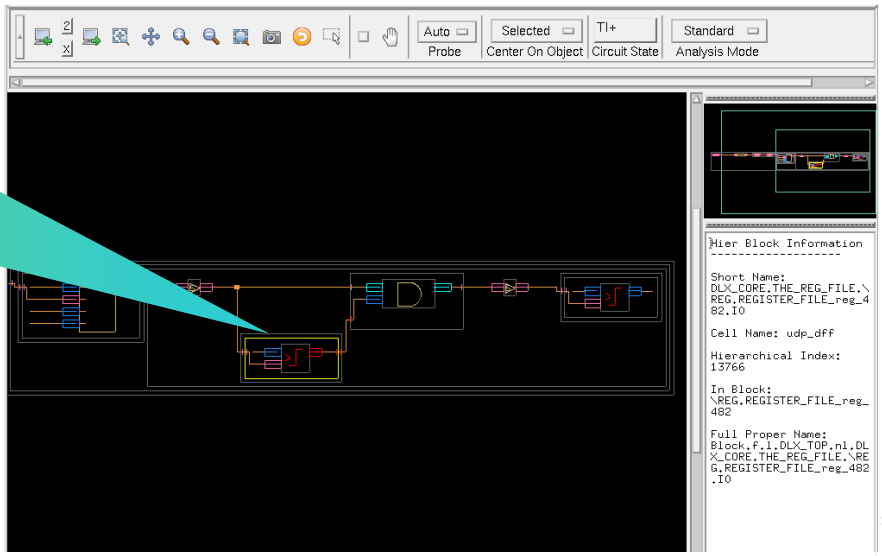
- To correct the deviation, install logic that prevents both blocks feeding the net from being simultaneously driven to opposite values.

cadence

This page does not contain notes.

Highlighted Contention Message in Schematic Window

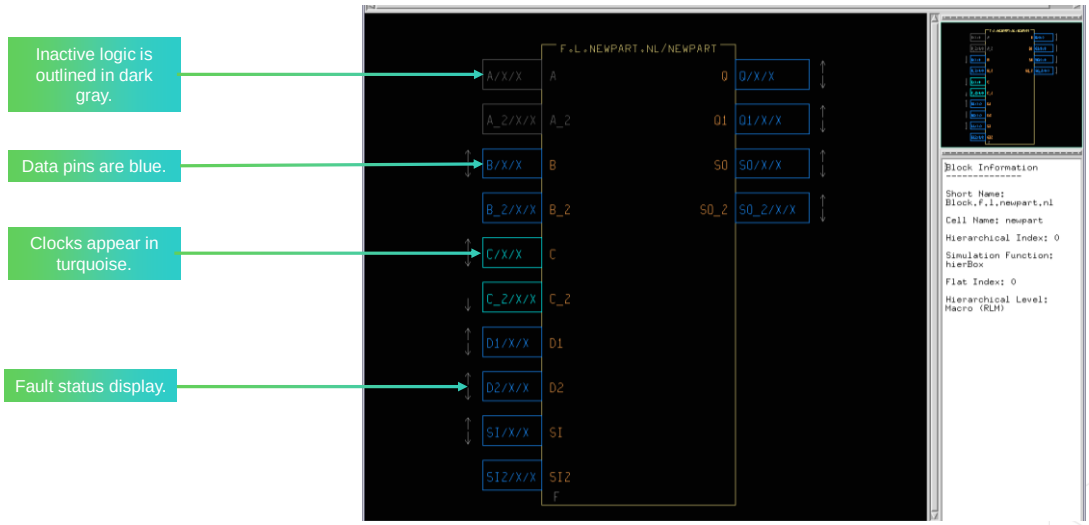
Highlights problem pins in red and pink (clocks) show sources and sinks of problem areas.
Verify that the correct net names or logic is in the design.



cadence

This page does not contain notes.

Logic Color Coding in Schematic



cadence

This page does not contain notes.

Other Tips for Broken Scan Chain Tracing and Debugging

Previously, we discussed the techniques outlined in the debugging techniques for a mux-scan design:

- We showed working at the library level. You can alternatively work at the primitive level. This helps find problems in the cell library.
- Sometimes, tracing forward can be useful using the **TSV-384** messages and not always tracing backward using the **TSV-40X** messages.
  - No single method is completely guaranteed to work in every case.
  - Try other approaches as well to debug the broken scan chains.

cadence

This page does not contain notes.

Submodule 4.2

Scan Chain Debugging with Tcl Interface

cadence

This page does not contain notes.

Submodule Objectives

In this submodule, you

- Identify the correct and incorrect flops by looking at their I/O values using Tcl interface
- Verify the polarity of the clocks, scan enable, and reset pins with Tcl interface
- Analyze the “Faulty Instance” in Tcl interface
- Evaluate the scan flops

cadence

This page does not contain notes.

Terminology and Commands to Interact with Modus

Terminology	Commands
Application Commands	Shell commands available in Modus Common UI (ex:- ***build_model, build_testmode, create_logic_tests***, etc.)
To set/clear context in the test mode/experiment in Modus Common UI	Example: • ***set_context -testmode FULLSCAN -experiment logic*** • ***set_context -clear***
Interactive/Analysis Commands operate on the context set by **set_context** command	Newly added Tcl commands in Modus Common UI (ex:- **set_context, simulate_state, trace_circuit, get_db, set_db, report_chains, vls, vcd**, etc.)
Commands to set and get information on Modus Objects/Attributes	• ***set_db*** • ***get_db***

cadence

This page does not contain notes.

Terminology and Commands to Interact with Modus (continued)

Terminology	Commands
To report information about: • Test functions • Scan chains • Pins • Instances	Use the below commands: • **report_testfunctions** • **report_chains** • **report_pins** • **report_instances**
To set/get/unset options for application commands: • Set the option • Get the option • Unset the option	Use the below commands: • ***set_option*** (ex:- ***set_option stdout summary***) • ***get_option*** (ex:- ***get_option stdout***) • ***unset_option*** (ex:- ***unset_option stdout)***
To set the circuit state in the Modus Tcl interface	***simulate_state scan***
To get the circuit state by simulate_state	***get_db circuit_state***

cadence

This page does not contain notes.

Terminology and Commands to Interact with Modus (continued)

Terminology	Commands
To view the values on pin objects in the circuit state	`get_db <pin_object> .value`
To simulate values on pins (custom circuit state)	`simulate_value <pin_object> -value <val1>/<val2>`
To view updated values on pin objects with custom state simulation	`get_db <pin_object> .value`
To perform forward or backward tracing while debugging the scan chains	Example: `trace_circuit -maxdepth 3 NPO[9]`
To redirect output to log or Tcl variable	Example: `redirect report_chains_log {report_testfunctions}`

cadence

This page does not contain notes.

Reporting the Scan Chains

```
report_chains -controlchain/-observechain <specific chain with chain ID>
-summary
```

- The **report_chains** command is used to report the scan chains.
- This requires the **analysis_context** to be set.
- To report a specific chain, mention the specific chain with specify the chain ID,
  - **report_chains -controlchain/-observechain**

Example

```
report_chains -summary
```


cadence

This page does not contain notes.

Report Chain Summary Result

```
@modus:root:/ 4> report_chains -summary
-----------------------------------------------------------------
Testmode        : FULLSCAN
Circuit State   : {TIE ONLY}
Experiment      : <not set>
-----------------------------------------------------------------

INFO (TUI-722): Start of Chain Summary Report

-----------------------------------------------------------------
                      Scan Chains Report
-----------------------------------------------------------------

Number of controllable and observable scan chains: 9
-------------------------------------------------------------------------------------------------------------
| Observe Chain Id | Control Chain Id | Length | Load Pin               | Unload Pin              |
-------------------------------------------------------------------------------------------------------------
| 7                | 13               | 85     | {DLX_CHIPTOP_DATA[6]}  | {DLX_CHIPTOP_DATA[22]} |
| 9                | 15               | 85     | {DLX_CHIPTOP_DATA[8]}  | {DLX_CHIPTOP_DATA[24]} |
| 10               | 16               | 85     | {DLX_CHIPTOP_DATA[9]}  | {DLX_CHIPTOP_DATA[25]} |
| 11               | 2                | 85     | {DLX_CHIPTOP_DATA[10]} | {DLX_CHIPTOP_DATA[26]} |
| 12               | 3                | 85     | {DLX_CHIPTOP_DATA[11]} | {DLX_CHIPTOP_DATA[27]} |
| 13               | 4                | 85     | {DLX_CHIPTOP_DATA[12]} | {DLX_CHIPTOP_DATA[28]} |
| 14               | 5                | 85     | {DLX_CHIPTOP_DATA[13]} | {DLX_CHIPTOP_DATA[29]} |
| 15               | 6                | 85     | {DLX_CHIPTOP_DATA[14]} | {DLX_CHIPTOP_DATA[30]} |
| 16               | 7                | 85     | {DLX_CHIPTOP_DATA[15]} | {DLX_CHIPTOP_DATA[31]} |
-------------------------------------------------------------------------------------------------------------
```


cadence

This page does not contain notes.

What Is Modus Data Model?

The Modus Data Model is the representation of Verilog netlist to the schematic form of the design in the Modus schematic viewer.

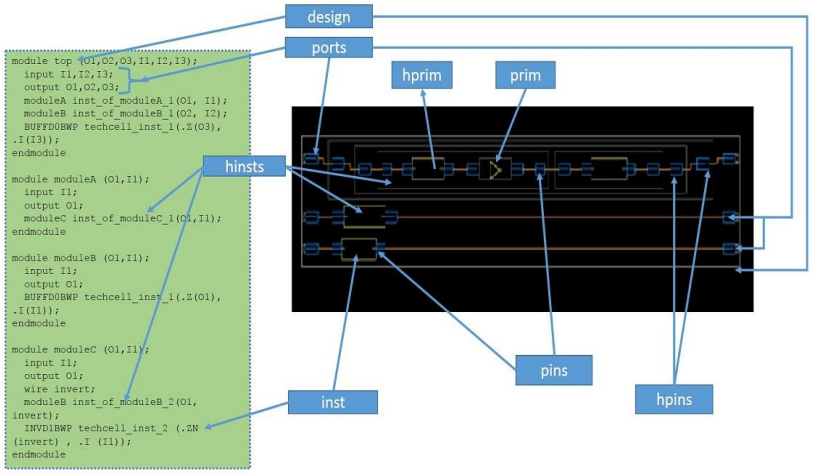
- Modus Common UI follows a slightly different naming convention for Verilog Netlist contents.
- You will need some time to get up to speed on the Verilog Data Model. To speed up on the Verilog Data Model, you can:
  - Invoke schematic from Modus Common UI and highlight a few netlist objects; the object representation is reported in Console View in Schematic window.
  - Take a small design and try running **`get_object`** command on various pins, blocks, and nets.

cadence

This page does not contain notes.

Example: Modus Data Model (Verilog to Schematic)

The following figure shows a sample Verilog of the Modus Data Model that lists the data objects along with the graphical representation of the Verilog in the Modus Schematic Viewer.



cadence

This page does not contain notes.

Modus Data Model Attributes

Take a small design and try running the **get_object** command on various pins, blocks, and nets. For example:

- To convert a Verilog name such as pin, block, net, use the **`get_object`** command. The steps to convert are:
  1. Get the netlist object name you want to determine the object type
  2. Find out if it is a Pin or Block or Net (object_type)
  3. Run **`get_object –<pin|net|block> <netlist object name>`**
  - Example: ***`get_object –pin DFT_sdi_2 \`***

    `port:DFT_sdi_2`
- To add an object in Schematic Viewer:
  - To add/replace Modus objects in Schematic Viewer, use **`gui_view_objects <list of objects>`**
    - **`gui_view_objects port:DFT_sdi_1`**
    - **`gui_view_objects {port:DFT_sdi_1 fault:366}`**
  - Schematic viewer opens up in the same test mode, circuit state, and experiment that was set using **`set_context/simulate_state`** attributes.
- To close Schematic Viewer, use **`gui_close_schematic`** command.

cadence

This page does not contain notes.

Modus Data Model Attributes (continued)

- To highlight the required instance/pins/object and its features, use Modus schematic attributes:
  - You always have access to the highlighted object in the schematic in the Modus Console window by using:
    - `get_db gui_highlighted_object`
  - You can register your scripts to allow custom information to be reported in the Schematic Console window:
    - `gui_register_highlight_script`
    - Schematic tells you what object is highlighted in the GUI.
    - Schematic tells you what script to call when an object is highlighted.
    - The user can register a script that processes the highlighted object and reports useful information.

cadence

This page does not contain notes.

Interactive Data Access with get_db Attribute

```
get_db <object/object_list> attribute_chain <pattern>
```

Use the **get_db** command to fetch information about data model objects.

- The property of a data model object is referred to as an attribute.

- `get_db fault:300 .type`
- `get_db observe_chain:1 .unload_pin`
- `get_db test_functions .flags`
- `get_db insts *DFT_ieee1500_cell_wbm_err_i*`
- `get_db ports -if {.direction == in}`

cadence

This page does not contain notes.

Tips to Access the Data from Modus Data Model

- To accept multiple objects as a list, use:
  - `get_db {{port:NPO[0]} {port:NPO[1]}} .value`
- Use foreach to print nice reports. $object is a special variable which contains the returned object:
  - `get_db test_functions -foreach {puts "[get_db $object .pin.name] [get_db $object .flags]"}`
- Use –unique option to print all possible values for the attribute.
  - `get_db faults .type -unique \`

```
ISA0 ISA1 ISR ISF OSA0 OSA1 OSR OSF \
get_db faults -if {.type == OSR}
```

- Use .? or .* to report all attributes for an object.
  - `get_db fault:300 .? \`

```
---- fault:300 ---- \
category : dynamic \
index : 300  \
is_representative : true
```


cadence

This page does not contain notes.

Tips to Access the Data from Modus Data Model (continued)

- You can chain attributes as long the return value of the attribute is an object.
  - `get_db observe_flop:(1,1) .observe_chain.unload_pin.name`
- You can get help on objects and attributes from get_db itself.
  - `get_db obj_type:fault .help \`

```
get_db  attribute:fault/category .help
```


cadence

This page does not contain notes.

Set the Analysis Context

```
set_context {{[-testmode <string>] [-experiment <string>]} | -clear}
```

- **set_context** is used to set or clear the analysis context in the Tcl environment.
- All subsequent Tcl commands will operate in the analysis context set by this command.
- When the test mode is set, the default circuit state is set to “Test Constraint”.
- This command is equivalent to setting analysis context in GUI.

Example

- **set_context -testmode FULLSCAN**
- **set_context -clear**

```
@modus:root:/ 1> set_context -testmode FULLSCAN
INFO (TUI-361): Initializing Model. This may take some time to load. [end TUI_361]
INFO (TUI-766): Initialized Model. CPU time:  0:00:00 Elapsed time:  0:00:00.02 [end TUI_766]
INFO (TUI-727): Successfully set the 'testmode' with value 'FULLSCAN'.  [end TUI_727]
INFO (TUI-726): The design is currently in circuit state 'tie_only'. [end TUI_726]
```


cadence

This page does not contain notes.

Get the Analysis Context

```
get_context {-testmode | -experiment}
```

- ***get_context*** is used to get the analysis context information set by ***set_context*** command.

```
get_context -testmode
```

Example

```
@modus:root:/ 4> get_context -testmode
FULLSCAN
```


cadence

This page does not contain notes.

Setting/Resetting the Circuit State

```
simulate_state {<state_name>}
```

- **simulate_state** command is used to set or reset the circuit state.
- This command should be invoked after setting a valid test mode using the **set_context** command.
- Users are required to simulate a state before analyzing the circuit.
- This helps report the values that are present on a circuit in a specific state like scan, **test_constraint**, **test_inihibit** etc.
- This command is equivalent to setting the circuit state in the GUI.

- **simulate_state scan**
- **simulate_state test_constraint**

Example

```
@modus:root:/ 5> simulate_state scan
INFO (TUI-726): The design is currently in circuit state 'tie_only'. [end TUI_726]

INFO (TUI-798): Circuit state 'scan' is set successfully with section as 'Scan_Section_Sequence'.
  [end TUI_798]
```


cadence

This page does not contain notes.

Circuit State Definitions

State of the Circuit	Definition
Tie only	Applies only the VDD, GND and other tied values in the circuit
Test Inhibit	Applies tie only and test inhibits from test mode definition
Test Constraint	Applies tie only, test inhibits, and test constraints
Test Constraints and Clocks Off	Applies test constraint state and puts the clocks to their stability state
Scan	Applies values to the circuit after the scan precondition state
Test Mode Initialization	Applies simulation state at the end of the mode initialization state
Scan Corrupted	Applies state at the end of scan

cadence

This page does not contain notes.

Tracing the Circuit

```
trace_circuit [-backward | -forward] [-maxdepth <integer> | -continue] [-reportprimitives] [-maxlimit <integer>] {<pin_name>}
```

- **`trace_circuit`** command is used to trace the circuit in the **forward** or **backward** direction starting from a specified pin name.
- By default, tracing proceeds **backward** and stops after tracing one level.
- The tracing happens in the circuit state set by the **`simulate_state`** command.
- If **`-continue`** option is specified, tracing proceeds in the specified direction until it hits a stopping condition.
  - Following are the stopping conditions:
    - PI, PO, RAM/ROMs, Cut Points, Tied Nodes, Sequential Blocks (Flops, Latches) and inactive pins

Example

- `trace_circuit top.core1.instA.A` #(Trace back one level from pin)
- `trace_circuit -continue top.core1.instA.A` #(Trace back from pin until stop conditions are reached.)

cadence

This page does not contain notes.

Example: Tracing Backward Output Report

The output report contains the pins identified during trace, along with the value of the pin in the circuit state.

The hierarchy of reported pins is marked using a double-spaced indentation.

The stopping conditions are reported next to the pins when the trace stops.

```
@modus:root:/ 7> trace_circuit -backward DLX_CORE.PC_REG.STORED_VALUE_reg_0.SI
----------------------------------------------------------------
Testmode        : FULLSCAN
Circuit State   : SCAN
Experiment      : <not set>
----------------------------------------------------------------


INFO (TUI-720): Start of Trace Report for 'Pin DLX_CORE.PC_REG.STORED_VALUE_reg_0.SI'

DLX_CORE.PC_REG.STORED_VALUE_reg_0.SI (x/x)
  DLX_CORE.PC_REG.BG_scan_in (x/x)
    DLX_CORE.THE_CONTROLLER.MAR_OUT_EN1 (x/x)
    [STOP: No_Next_Pins ]
```


cadence

This page does not contain notes.

Reporting the Test Functions

```
report_testfunctions {-names <test_function_name>+ | -all}
```

- **report_testfunctions** command is used to report the information about pins associated with the specified test function names.
- The command accepts the test function names below and reports the pins associated with these test functions.
  - shiftclock, captureclock, shiftandcaptureclock
  - shiftconstraint, testconstraint, captureconstraint, compressionconstraint
  - scanin, scanout, opcgconstraint, ppi, boundaryscan

report_testfunctions -names {scanin scanout}

report_testfunctions -all

```
@mbdus:root:/ 8> report_testfunctions
-------------------------------------------------------------------------------
Testmode          : FULLSCAN
Circuit State     : SCAN
Experiment        : <not set>
-------------------------------------------------------------------------------

INFO (TUI-722): Start of test functions Report

-------------------------------------------------------------------------------
                        Test Function Pins Report
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
| Test Function          | Flag     | Pin Name                  | Value |
-------------------------------------------------------------------------------
| captureclock/reset     | +SC      | DLX_CHIPTOP_RESET         | 1/1   |
|                        | -SC -TI  | DLX_CHIPTOP_TCK           | -/-   |
| shiftandcaptureclock   | -ES      | DLX_CHIPTOP_SYS_CLK       | 0/0   |
|                        | -ES      | DLX_CHIPTOP_TEST_CLOCK    | 0/0   |
| shiftconstraint        | SO ZSE   | DLX_CHIPTOP_DATA[16]      | X/X   |
|                        | SO ZSE   | DLX_CHIPTOP_DATA[17]      | X/X   |
|                        | SO ZSE   | DLX_CHIPTOP_DATA[18]      | X/X   |
|                        | SO ZSE   | DLX_CHIPTOP_DATA[19]      | X/X   |
|                        | SO ZSE   | DLX_CHIPTOP_DATA[20]      | X/X   |
|                        | SO ZSE   | DLX_CHIPTOP_DATA[21]      | X/X   |
|                        | SO ZSE   | DLX_CHIPTOP_DATA[22]      | X/X   |
|                        | SO ZSE   | DLX_CHIPTOP_DATA[23]      | X/X   |
|                        | SO ZSE   | DLX_CHIPTOP_DATA[24]      | X/X   |
|                        | SO ZSE   | DLX_CHIPTOP_DATA[25]      | X/X   |
|                        | SO ZSE   | DLX_CHIPTOP_DATA[26]      | X/X   |
|                        | SO ZSE   | DLX_CHIPTOP_DATA[27]      | X/X   |
|                        | SO ZSE   | DLX_CHIPTOP_DATA[28]      | X/X   |
|                        | SO ZSE   | DLX_CHIPTOP_DATA[29]      | X/X   |
```


cadence

This page does not contain notes.

Reporting the Pins

```
report_pins {<pin_name>+}
```

report_pins command is used to report the information about the specified pin(s).

- The command reports key information about the pin(s). Some of the fields include:
  - Direction
  - Parent Net
  - Child Net
  - Active/Inactive
  - Test Function Flag
  - On Scan Path Information
  - Clocks Affiliated information
  - Value of the pin in the current circuit state.

report_pins DLX_CORE.PC_REG.STORED_VALUE_reg_0.SI

```
@modus:root:/ 10> report_pins DLX_CORE.PC_REG.STORED_VALUE_reg_0.SI
-----------------------------------------------------------------
Testmode        : FULLSCAN
Circuit State   : SCAN
Experiment      : <not set>
-----------------------------------------------------------------

INFO (TUI-720): Start of Pin Report for 'DLX_CORE.PC_REG.STORED_VALUE_reg_0.SI'

-----------------------------------------------------------------
                    Pin Report
-----------------------------------------------------------------
  Direction             : Input
  Instance              : DLX_CORE.PC_REG.STORED_VALUE_reg_0
  Module                : SDFFRX1
  Parent Net            : DLX_CORE.PC_REG.BG_scan_in
  Child Net             : DLX_CORE.PC_REG.STORED_VALUE_reg_0.SI
  Active                : true
  Test Flags            : None
  On Scan Path          : false
  Clocks Affiliated     : None
  Value                 : x/x
-----------------------------------------------------------------

End of Pin Report [end TUI 720]
```


cadence

This page does not contain notes.

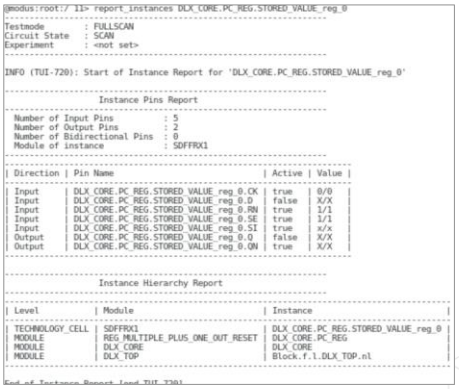
Reporting the Instances

```
report_instances {<instance_name>+}
```

report_instances command is used to report the information about the specified instances (i.e. blocks).

- The command reports all the pins on the instance and their direction.
- Parent hierarchy of the instance till the top module is reported.
- The pins reported for the instance can be specified to the **report_pins** command for reporting information on the pins.

```
report_instances
DLX_CORE.PC_REG.STORED_VALUE_reg_0
```



cadence

This page does not contain notes.

Reporting the Scan Chains

```
report_chains {-summary | {[-controlchain <index>] | [-observechain
<index>]} [-bits <bit>+]}
```

report_chains command is used to report the information about the scan chains and scan flops in the design.

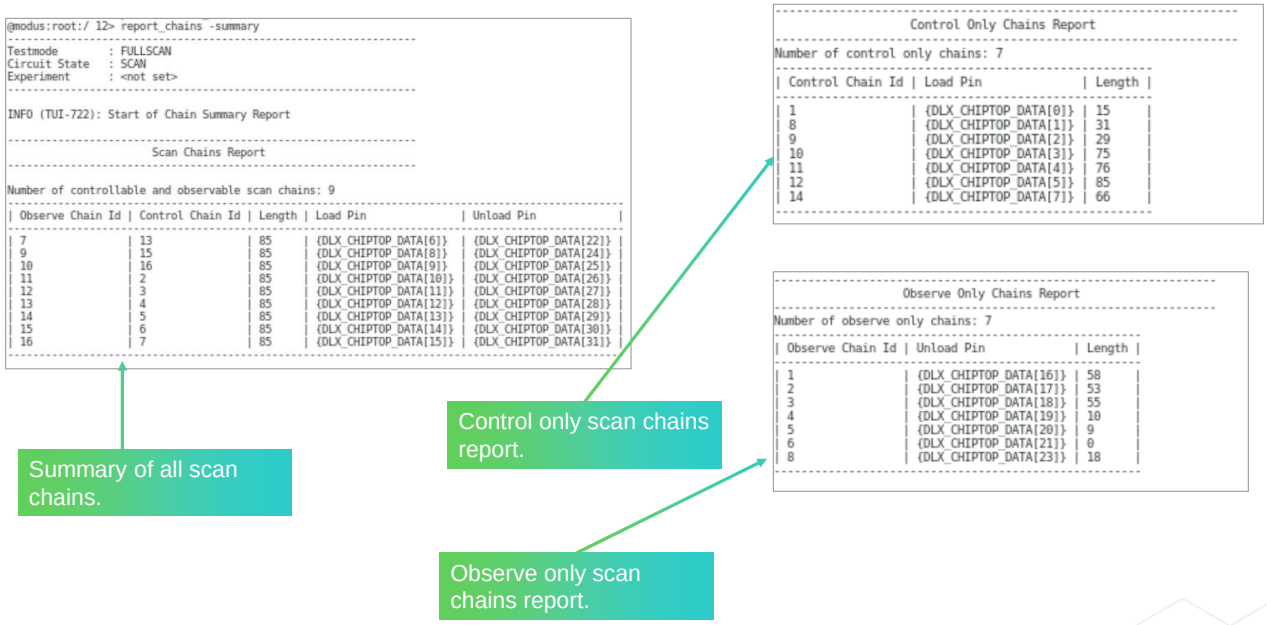
- This command helps to report the following:
  - The summary of all scan chains
  - A controllable or observable scan chain
  - Specific flop(s) in a scan chain

```
report_chains -summary
report_chains -observechain 1 -bits {1 2 4}
```


cadence

This page does not contain notes.

Example: Report Scan Chains Summary



185

cadence

This page does not contain notes.

Debugging the Broken Scan Chain with Tcl Command Line

To debug the broken scan chain in the Tcl command line interface, report the chain summary to see the broken scan chain, and then debug it by either tracing forward or backward, depending on the type of scan chain (controllable only or observable only).

1. Identify the broken scan chain
2. Report the last good flop
3. Find the faulty instance
4. Tracing the faulty instance

cadence

This page does not contain notes.

Step 01: Identifying the Broken Scan Chains

1. Use "**`report_chains`**" to report the scan chains.
2. Look for control only and observe only chain reports.

From the TSV – 40X messages in the verify_test_structure summary, we got to know that {DLX_CHIPTOP_DATA[16]} scan chain is broken.

Scan chain ID – 1

Chain length – 58 bit

```
------------------------------------------------------------------
                    Control Only Chains Report
------------------------------------------------------------------
Number of control only chains: 7
-----------------------------------------------------
| Control Chain Id | Load Pin               | Length |
-----------------------------------------------------
| 1                | {DLX_CHIPTOP_DATA[0]}  | 15     |
| 8                | {DLX_CHIPTOP_DATA[1]}  | 31     |
| 9                | {DLX_CHIPTOP_DATA[2]}  | 29     |
| 10               | {DLX_CHIPTOP_DATA[3]}  | 75     |
| 11               | {DLX_CHIPTOP_DATA[4]}  | 76     |
| 12               | {DLX_CHIPTOP_DATA[5]}  | 85     |
| 14               | {DLX_CHIPTOP_DATA[7]}  | 66     |
-----------------------------------------------------

------------------------------------------------------------------
                    Observe Only Chains Report
------------------------------------------------------------------
Number of observe only chains: 7
-----------------------------------------------------
| Observe Chain Id | Unload Pin             | Length |
-----------------------------------------------------
| 1                | {DLX_CHIPTOP_DATA[16]} | 58     |
| 2                | {DLX_CHIPTOP_DATA[17]} | 53     |
| 3                | {DLX_CHIPTOP_DATA[18]} | 55     |
| 4                | {DLX_CHIPTOP_DATA[19]} | 10     |
| 5                | {DLX_CHIPTOP_DATA[20]} | 9      |
| 6                | {DLX_CHIPTOP_DATA[21]} | 0      |
| 8                | {DLX_CHIPTOP_DATA[23]} | 18     |
-----------------------------------------------------
```


cadence

This page does not contain notes.

Step 02: Reporting the Last Good Flop

The broken scan chain has been identified; now focus on the observe chain only to report the last good flop:

- Use the **`-observechains`** option to report the details
  1. **`report_chains -observechain 1`**
  2. Look for the last good flop

The analysis stopped at Observe Flop 58. This is the last Flop that Modus recognizes as a valid scan flop in the chain. This is the flop to start tracing backward for debugging.

```
-----------------------------------------------------------------
                  Observe Chain Id 1 Report
-----------------------------------------------------------------
Number of flops in chain: 58

-------------------------------------------------------------------------------------------------------------------
| Instance                                                                  | Bit Position | Inverted from unload |
-------------------------------------------------------------------------------------------------------------------
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_966.__iNsT1.dff_primitive | 1            | true                 |
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_967.__iNsT1.dff_primitive | 2            | true                 |
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_968.__iNsT1.dff_primitive | 3            | true                 |
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_969.__iNsT1.dff_primitive | 4            | true                 |
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_970.__iNsT1.dff_primitive | 5            | true                 |
<<<<<Lines have been removed >>>>>
| DLX_CORE.PC_REG.STORED_VALUE_reg30_31.__iNsT2.dff_primitive               | 52           | true                 |
| DLX_CORE.PC_REG.STORED_VALUE_reg26_27.__iNsT2.dff_primitive               | 54           | true                 |
| DLX_CORE.PC_REG.STORED_VALUE_reg1_2.__iNsT2.dff_primitive                 | 55           | true                 |
| DLX_CORE.PC_REG.STORED_VALUE_reg21_22.__iNsT2.dff_primitive               | 56           | true                 |
| DLX_CORE.PC_REG.STORED_VALUE_reg4_5.__iNsT2.dff_primitive                 | 57           | true                 |
| DLX_CORE.PC_REG.STORED_VALUE_reg_0.__iNsT2.dff_primitive                  | 58           | true                 |
-------------------------------------------------------------------------------------------------------------------
```


cadence

This page does not contain notes.

Step 03: Finding the Faulty Instance

The command "**report_instances**" reports the information about the specified instances.

- It fetches the structural and simulation information of the instance in the state simulated using the "**simulate_state**" command.
- Report the last good scan element after setting the circuit state using:

```
1. simulate_state scan
2. report_instances
   DLX_CORE.PC_REG.STORED_VALUE_reg_0
```

- Verify that the various pins are at the required values or not.

```
Testmode        : FULLSCAN
Circuit State   : SCAN
Experiment      : <not set>
------------------------------------------------------------------

INFO (TUI-720): Start of Instance Report for 'DLX_CORE.PC_REG.STORED_VALUE_reg_0'

------------------------------------------------------------------
                  Instance Pins Report
------------------------------------------------------------------
  Number of Input Pins          : 5
  Number of Output Pins         : 2
  Number of Bidirectional Pins  : 0
  Module of instance            : SDFFRX1
------------------------------------------------------------------
------------------------------------------------------------------------
| Direction | Pin Name                                  | Active | Value |
------------------------------------------------------------------------
| Input     | DLX_CORE.PC_REG.STORED_VALUE_reg_0.CK | true   | 0/1   |
| Input     | DLX_CORE.PC_REG.STORED_VALUE_reg_0.D  | false  | X/X   |
| Input     | DLX_CORE.PC_REG.STORED_VALUE_reg_0.RN | true   | 1/1   |
| Input     | DLX_CORE.PC_REG.STORED_VALUE_reg_0.SE | true   | 1/1   |
| Input     | DLX_CORE.PC_REG.STORED_VALUE_reg_0.SI | true   | x/x   |
| Output    | DLX_CORE.PC_REG.STORED_VALUE_reg_0.Q  | true   | X/X   |
| Output    | DLX_CORE.PC_REG.STORED_VALUE_reg_0.QN | true   | X/X   |
------------------------------------------------------------------------
```

Noticed that the SI pin is at the small 'x' value. This can be an issue; a small x means it is not driven by any logic.

cadence

This page does not contain notes.

Step 04: Tracing the Faulty Instance

`report_chains` will show the last “correct” instance the tool found. Use command trace_circuit to trace back or forward from a specific pin.

- If tracing a broken Observable chain, you need to trace backward from the SI pin of the flop, usually just to the next scan flop in the chain.
- If tracing a broken Controllable chain, you need to trace forward from the SO pin of the flop to the next scan flop in the chain.
- We are tracing backward by using:
  - **`trace_circuit -backward DLX_CORE.PC_REG.STORED_VALUE_reg_0.SI`**

```
INFO (TUI-720): Start of Trace Report for 'Pin
DLX_CORE.PC_REG.STORED_VALUE_reg_0.SI'

DLX_CORE.PC_REG.STORED_VALUE_reg_0.SI (x/x)
  DLX_CORE.PC_REG.BG_scan_in (x/x)
    DLX_CORE.THE_CONTROLLER.MAR_OUT_EN1 (x/x)
    [STOP: No_Next_Pins ]
```

Trace circuit report summary

cadence

This page does not contain notes.

Additional Help and Topics

Help References

- General Information
  - Modus: An Overview and Quick Start
- TSV Debug
  - Modus: Guide 3: Test Structures -> Verify Test Structures -> Analyzing Test Structure Problems in the Design
- Message descriptions
  - Modus: Reference: Messages
    - TSV -> Test Structure Verification Messages
    - TMI -> Verify Core Isolation Messages
    - TJC - IEEE 1149.1 Boundary Scan Verification Messages
- Using the Interactive Schematic Viewer
  - Modus: Reference: GUI
  - Modus: Reference: Stylus Common UI

cadence

This page does not contain notes.

Module Summary

In this module, you learned how to

- Trace the broken scan chains with GUI and Tcl interface.
- Debug the different errors and warnings, such as TSV 385, TSV 40X, and TSV 093
- Analyze the Faulty Instance

cadence

This page does not contain notes.

Labs

Lab 4-1 Debugging the Broken Scan Chains with Modus GUI

- Setting the Analysis Context
- Customizing the Schematic Window
- Interact with Schematic Display Window
- Debugging Broken Scan Chains

Lab 4-2 Debugging Broken Scan Chains with Tcl Command Line Interface

- During Debugging, Command Line Utilities to Use
- Invoking Modus and Set Context
- Finding a Broken Scan Chain
- Debugging Observe Scan Chain 1
- Debugging Observe Scan Chain 4
- Exiting the Modus

cadence

This page does not contain notes.

Module 5

Analyze Initialization Sequence Using Modus GUI

cadence

This page does not contain notes.

Module Objectives

In this module, you will be able to

- Simulate the scan sequences in Modus GUI
- Analyze the sequences using Modus and SimVision™

cadence

This page does not contain notes.

How to Simulate Scan Sequences in Modus Sequence Analyzer

Modus sequence analyzer is a very powerful tool to analyze the sequence in GUI. The process of sequence analysis is explained here.

1. Bringing the Sequence Analyzer Window
2. Bringing the logic to Modus Schematic Viewer
3. Adding a pin to the watch list
4. Initializing Analyze Sequences

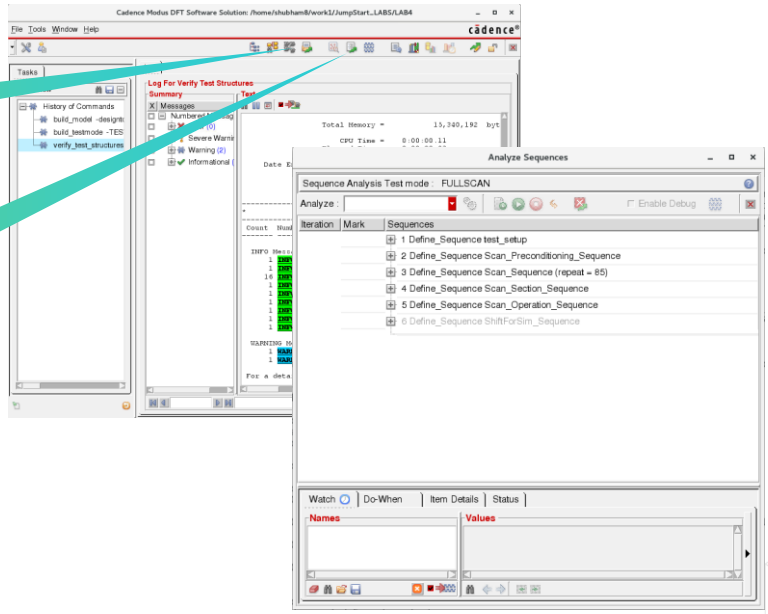
cadence

This page does not contain notes.

Step 01: Bringing up the “Analyze Sequences” Window

Firstly, from the main GUI window, ensure the analysis context is set to the test mode you care about.

Click the **Analyze Sequence** icon to bring up the “Analyze Sequences” window.



197

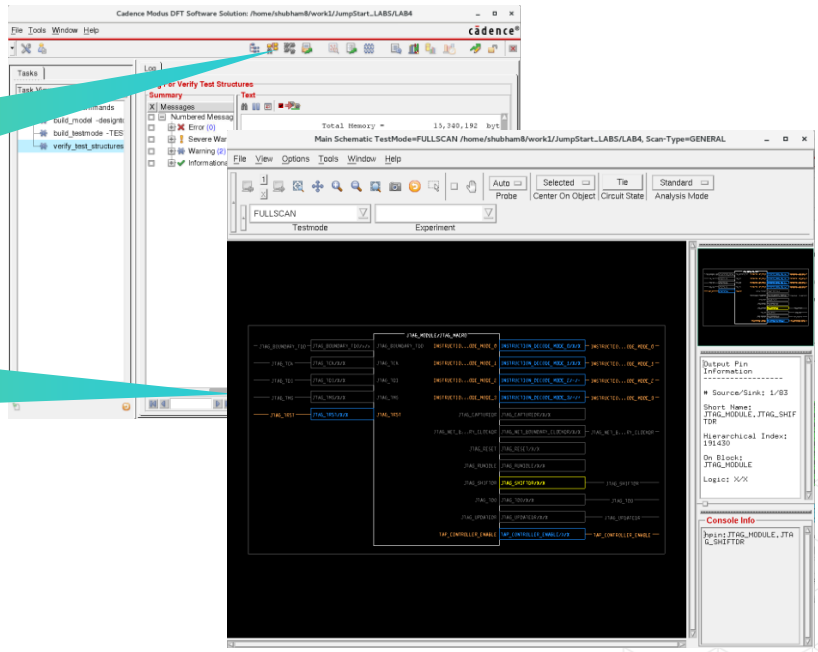
cadence

This page does not contain notes.

Step 02: Bringing Logic to Modus Schematic Viewer

Go to the schematic GUI and find signals to trace and look at the inside window. In the main GUI, click View DesignSearch (Block) icon and mouse over it.

Type in JTAG_MODULE into the search field and hit return. This will bring up the JTAG module into the schematic view. Capture all of the pins into a watch list to track them in the analyze sequence GUI.

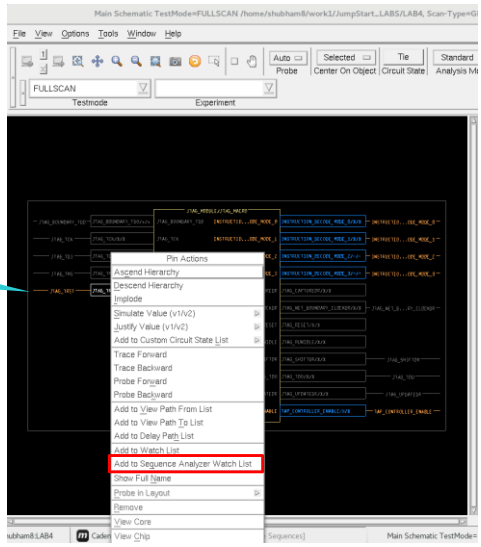


This page does not contain notes.

Step 03: Adding a Pin to a Watch List

Left-click on the JTAG_TRST pin. **Right**-click and select **Add to Sequence Analyzer Watch List** from the pulldown menu. If you go back and look at the Analyze Sequences window, you will see this signal added in the lower left corner.

Go ahead and add JTAG_TRST, JTAG_TDI, and JTAG_TMS using the same method.



cadence

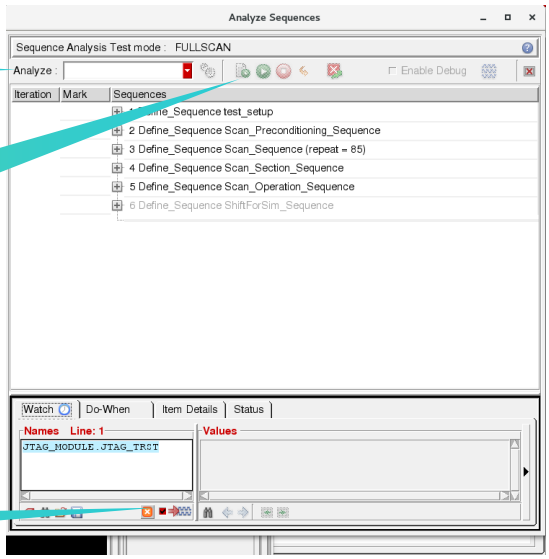
This page does not contain notes.

Step 04: Initializing Analyze Sequences

Select **test_setup** in the Analyze pulldown. This selects the Initialization sequence that we used in the DLX.seqdef file.

Select the **Initialize Simulation** icon (small green play button). You will see values in the bottom pane and a blue Mark Arrow, showing you where you are in the sequence. SimVision will also start up with the watch signals in a Waveform window.

Make sure the icon on the right side is activated. This will enable waveform recording of the simulation of these pins.



cadence

This page does not contain notes.

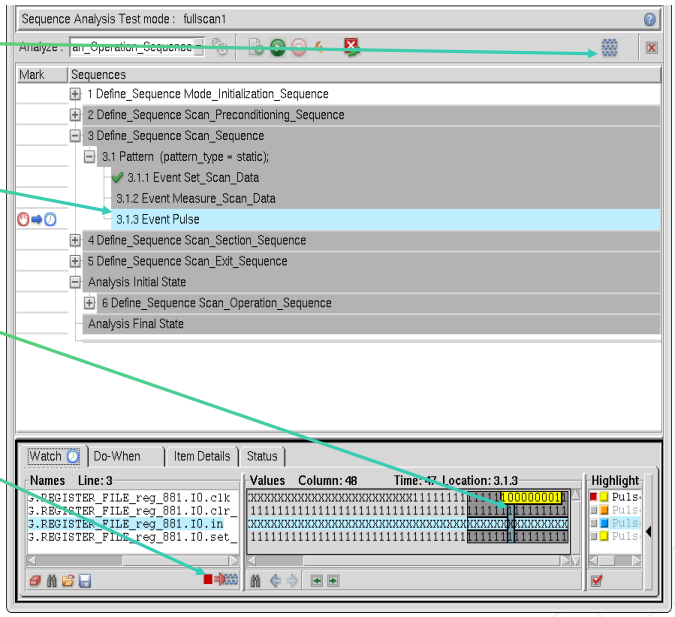
Step 04: Initializing Analyze Sequences (continued)

Click the icon to bring up the waveform display

Current event

Click the column and use right mouse button to view values on the schematic

Enable waveform data



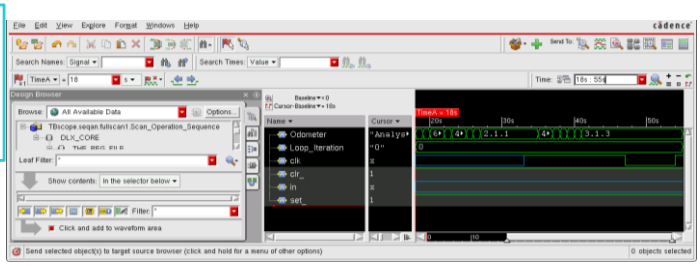
201 © Cadence Design Systems, Inc. All rights reserved.

cadence

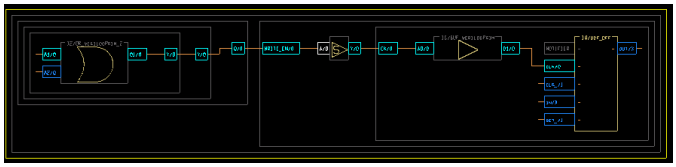
This page does not contain notes.

Analyzing Scan Sequences

After clicking the waveform icon in the "**Analyze Sequences**" window bring up the waveform display in SimVision.



Simulation values are shown in the waveform (in SimVision) window and schematic viewer (in Modus schematic) GUI.



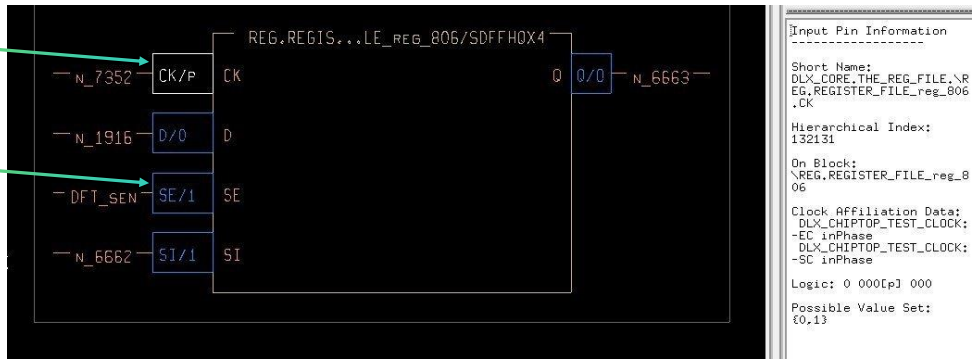
cadence

This page does not contain notes.

Correct Scan Clock During Scan Sequence

A clean pulse “p” is seen on the clock pin.

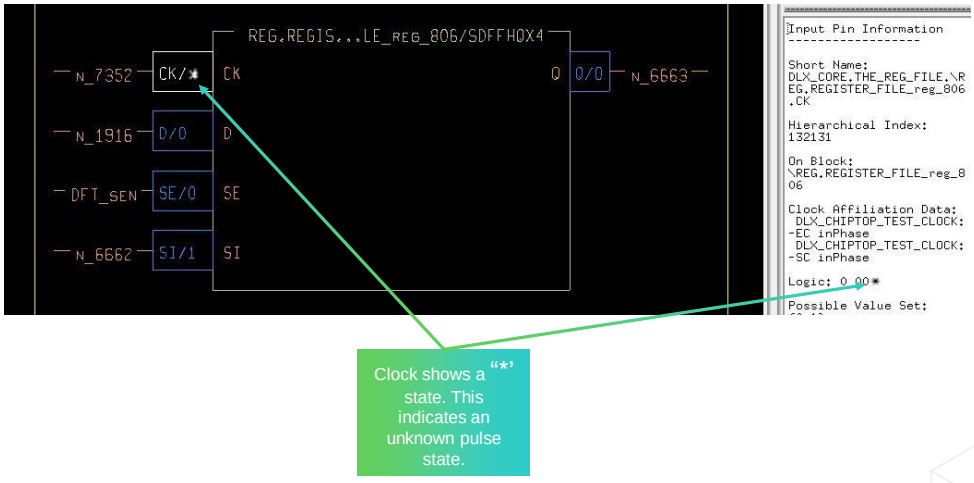
“SE” is active during the sequence.



cadence

This page does not contain notes.

Corrupt Scan Clock During Scan Sequence



Clock shows a "*" state. This indicates an unknown pulse state.

cadence

This page does not contain notes.

Module Summary

In this module, you learned how to

- Simulate the scan sequences in Modus GUI
- Analyze the Sequences using Modus and SimVision

cadence

This page does not contain notes.

Lab

Lab 5-1 Analyzing Initialization Sequence Using Modus GUI

- Invoking Modus
- Setting the Analysis Context and Opening the Sequence Analyzer
- Opening the Schematic and Creating a Watch List
- Analyzing Sequences
- Exiting the Modus

cadence

This page does not contain notes.

Module 6

Debugging the Test Patterns

cadence

This page does not contain notes.

Module Objectives

In this module, you

- Debug the test patterns with Modus diagnostics tool
- Apply and simulate the pattern values
- Debug the waveforms with SimVision™

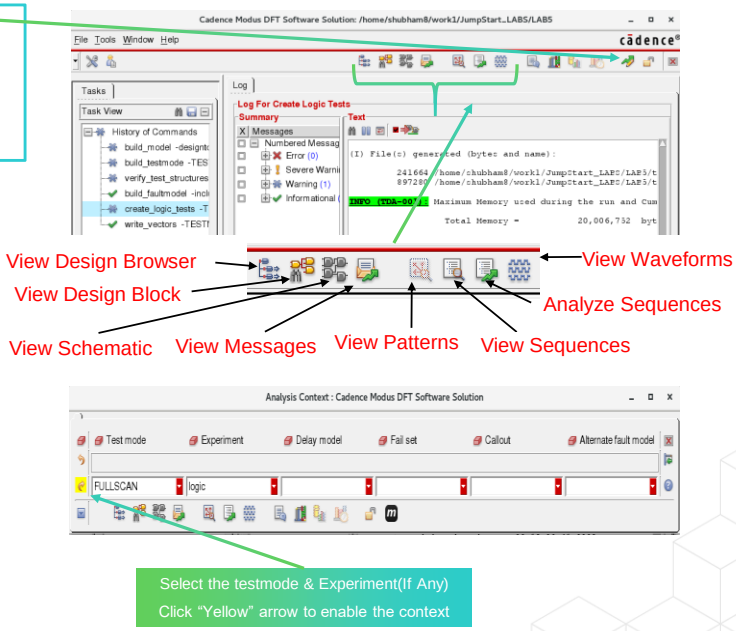
cadence

This page does not contain notes.

Loading Circuit Analysis Context

Click on the "**View Analysis**" icon to view the analysis and load the context.

- Modus allows test mode and pattern data values to be displayed in the circuit display window.
- The test modes and experiments are referred to as context and can be set using GUI and command line interface.
- The Values used are:
  - Test mode name
    - Allows viewing of scan chain information, clocks, and other mode specific information
  - Experiment name
    - Allows viewing of pattern data in the logic

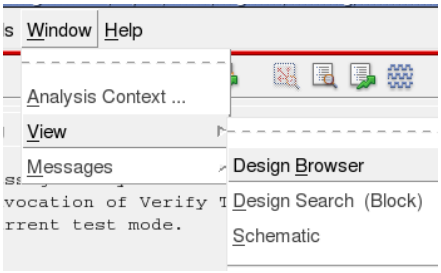


Select the testmode & Experiment(If Any)
Click "Yellow" arrow to enable the context

This page does not contain notes.

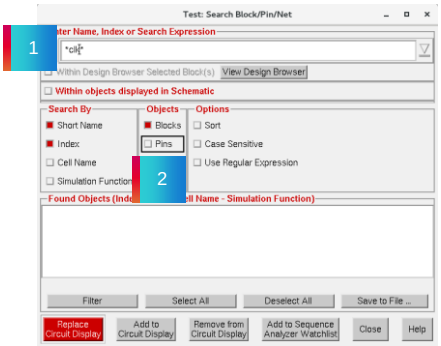
Circuit Display

To access the circuit display, use the pull-down **Window –> View –> Design Search** or click one of these icons in the main Modus GUI.



Modus has a built-in tool for circuit display/tracing. Once a model has been built, circuit tracing can be performed.

- Clicking the middle button brings up a window to enter objects to search for:
  1. Enter the object name to place in the GUI.
  2. Select from Block, Pin, and/or Net.
  3. Allows wildcard searches.
  4. Enter “0” to display the top level of the design.



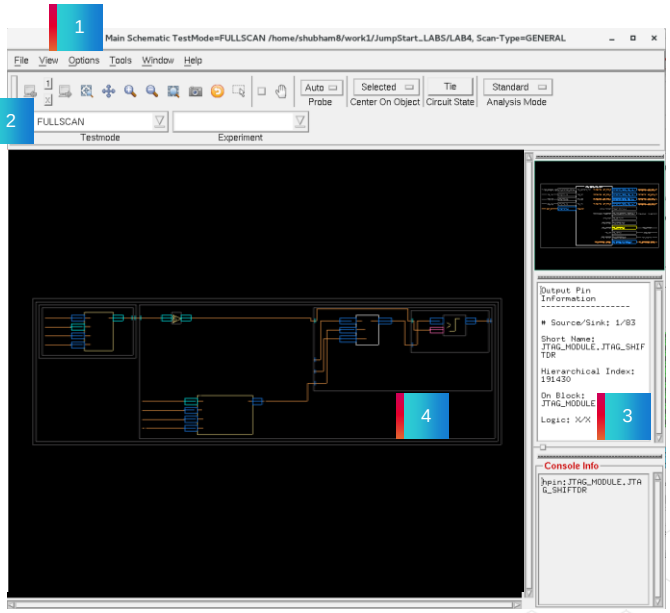
cadence

This page does not contain notes.

Graphical Display Window

Filling the required information in the "**Circuit Display**" window and clicking the "**Replace Circuit Display**" tab, you reach the Graphical Display window.

1. Ability to change tracing and display behaviors.
2. Ability to move forward and backward on images.
3. Display information of the selected object (Information can be frozen by pressing Ctrl-s).
4. Allow circuit tracing at primitive or library cell level.



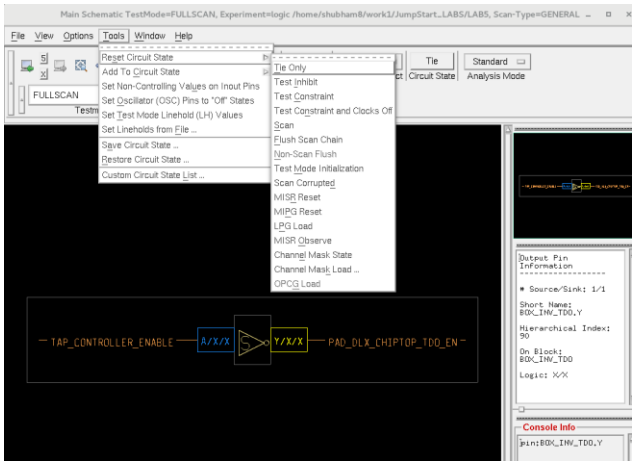
cadence

This page does not contain notes.

Predefining Circuit States

In the main schematic window, choose **Tools -> Reset Circuit State -> Select the Circuit State**.

- When processing designs, Modus saves particular circuit states that it uses during the processing.
- Circuit states are available from the GUI toolbar.
- Applying a circuit state places the saved state into the circuit display window.
- Use these circuit values to debug:
  - Mode initialization states
  - Scan chain debug
  - Untestable faults
  - Miscompares



If Circuit State does not appear in the toolbar, do the following:

1. Options –> Customize Toolbar
2. Click Default > Edit > Apply > OK > Close

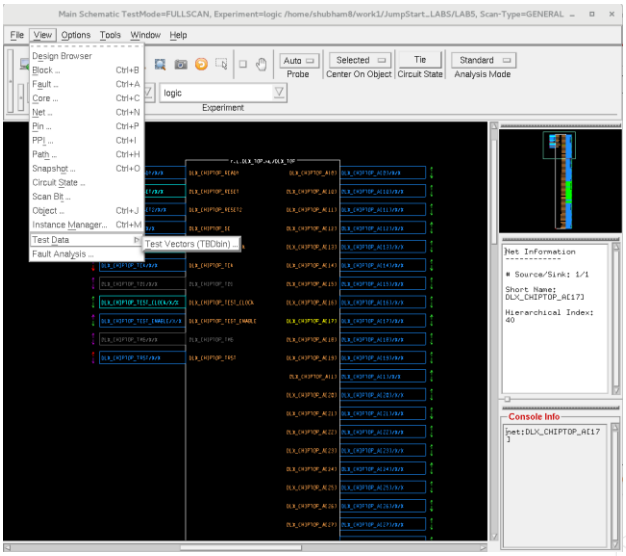
cadence

This page does not contain notes.

Viewing Test Patterns in Circuit Display

In the main schematic window, choose **View -> Test Data -> Test Vectors (TBDbin)…**

- Load an experiment through the Analysis Context window to set the testmode and pattern experiment.
- To view the patterns, you can use the pull-downs in the Circuit Display.



cadence

This page does not contain notes.

Viewing Test Patterns in Circuit Display (continued)

View Test Data displays the pattern contents by odometer value:

- The mode initialization sequence is found under Test Procedure 1.1.1.1
- Click on blue boxes to expand the hierarchy
  - Modus: Test data hierarchy format–

    Experiment: Name specified when tests were created

    Test Section: Unique tests (for example, scan chain integrity, logic, IDDq)

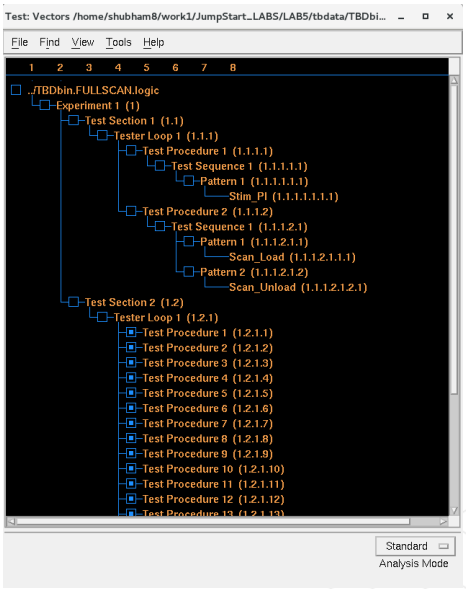
    Tester Loop: Collection of test procedures independently applicable

    Test Procedure: Collection of test sequences, usually with same clocking

    Test Sequence: Ordered set of patterns often targeting a specific defect

    Patten: Ordered set of events applied within a single tester cycle

    Event: Stimulus and response directives for the tester



cadence

This page does not contain notes.

Selecting and Applying Test Sequence

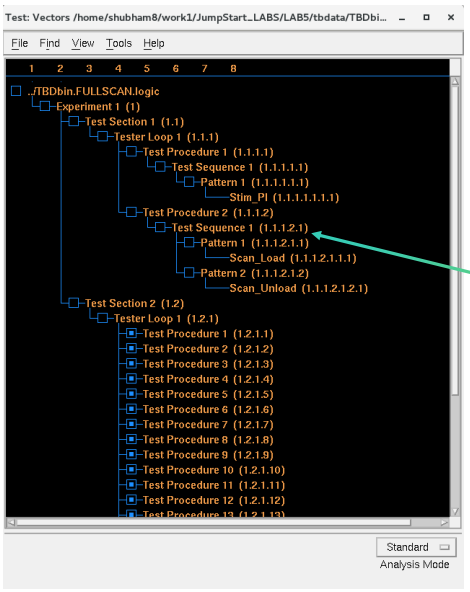
Select a test sequence, and **right**-click to open the action menu. To apply the test sequence, choose View Circuit Values; it will simulate and apply the circuit values on the Schematic Display.

1. Selecting a Test Sequence
2. Applying the Test Sequence

cadence

This page does not contain notes.

Step 01: Selecting a Test Sequence



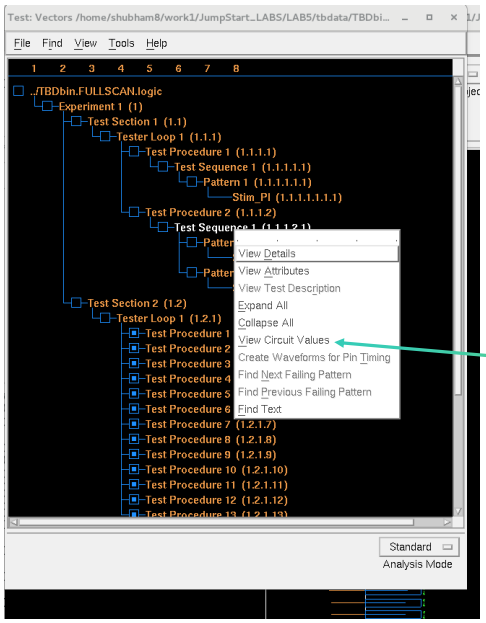
1. Selecting a Test Sequence
2. Applying a Test Sequence

Step 1. Select a test sequence and **right**-click to open the action menu.

cadence

This page does not contain notes.

Step 02: Applying a Test Sequence



1. Selecting a Test Sequence
2. Applying a Test Sequence

Step 2. Choose **View Circuit Values** will simulate and apply the circuit values on the Schematic Display.

cadence

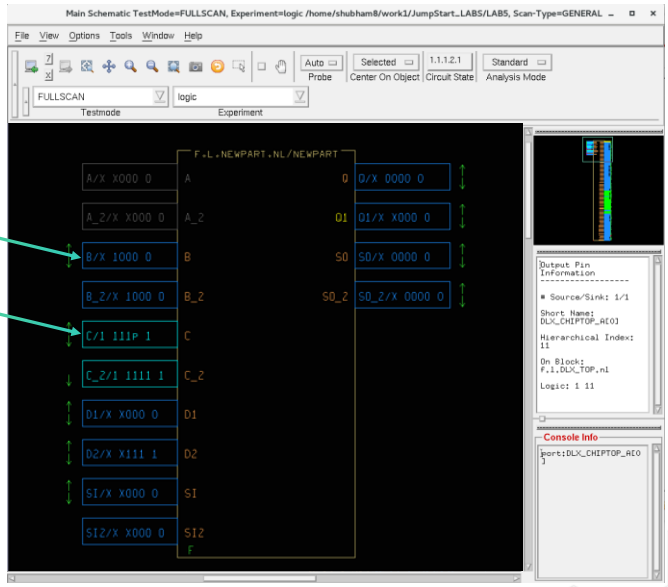
This page does not contain notes.

Simulated Pattern Values

Simulated values appear on pins

Pulses in the patterns appears as “P”

NOTE: These are Good Machine expected values, NOT Verilog simulation results.

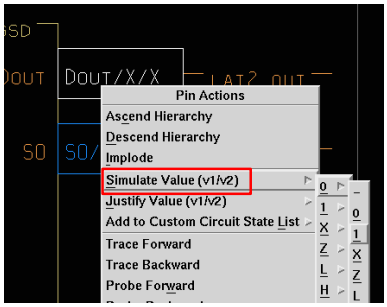


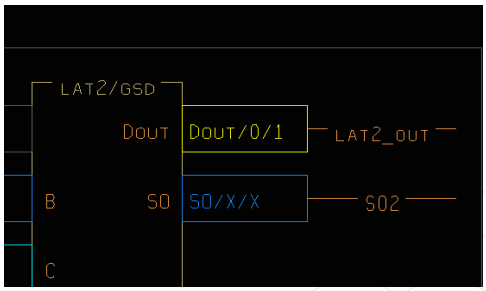
cadence

This page does not contain notes.

Simulating the Values

- Values can be simulated to see results on circuit design:
  - Values propagate forward
- Can simulate two completely independent machines at once:
  - 0/0,0/1,1/0,1/1 values on a single pin
- Values of 1 and 0 are equivalent to VDD and GND.
- Some pin value that you might also see:
  - +/+ - This states the pins are always a 1 due to a forced constraint
  - -/- - This states the pins are always a 0 due to a forced constraint
  - X/X – Pins are driven. Could be any valid value
  - x/x – Pins are not driven/dangling, therefore tied to an unknown value
- In the circuit browser, the two values specified on each pin represent two independent simulation machines:
  - Simulate Value lets you specify two values down a single path
  - Lets you run multiple scenarios





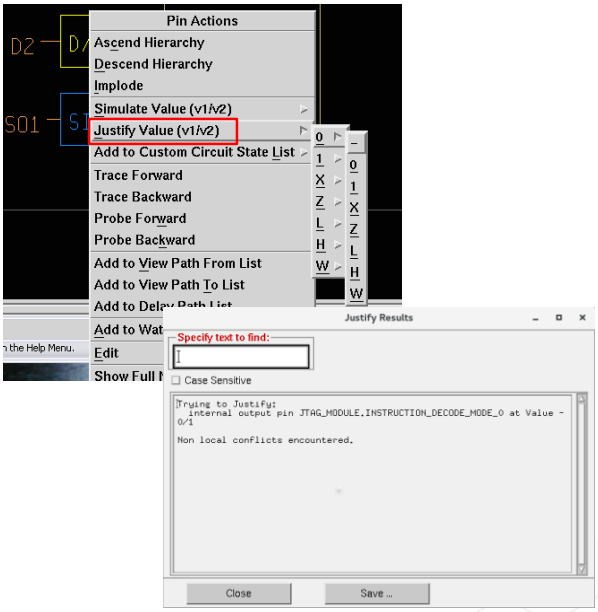
cadence

This page does not contain notes.

Justifying the Values

The test generator will search the input cone of logic to find out what values need to be set to attain the set value on the pin in question.

- Since there are two separate simulators:
  - Justify a 0/0 or a 1/1
  - Also, it can justify two different values back through the logic cone. Ex. 0/1, 1/0

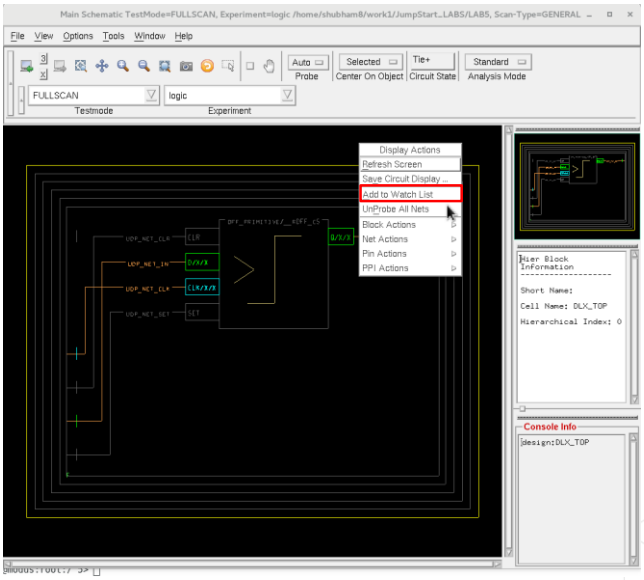


cadence

This page does not contain notes.

Adding Circuit to Watch List Using GUI

In the circuit design browser, Click on the **Empty Screen -> right**-click to see **Display Action** menu **->** Click **Add to Watch List.**



- Modus can create SimVision files to allow pattern tracing and debugging. These will be Good Machine values from what Modus is expecting from the circuit.
- To create a SimVision file, you must simulate patterns telling the software to watch particular nets within the design.
  - A watch list can be created from the GUI or by a script.

cadence

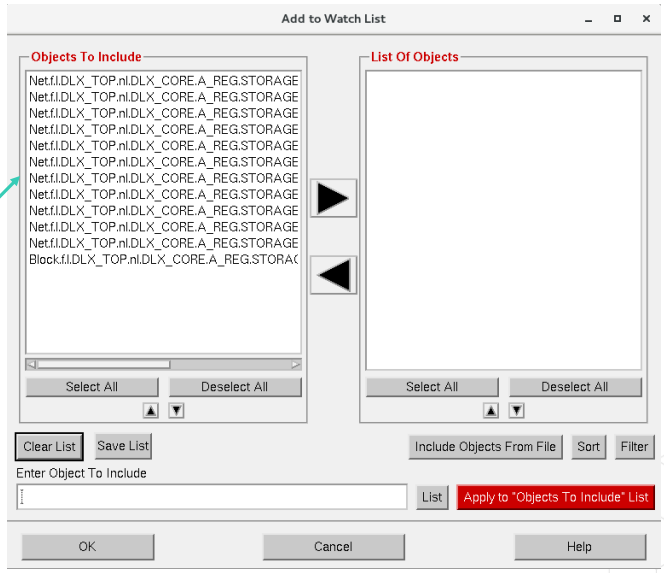
This page does not contain notes.

Watch List Window

After adding the circuit to watch list, a window will appear,

- The Watch List window lets you select and deselect which net and blocks to observe.
- You can merge multiple watch lists.
- Click **OK** to have the tool save list to a file.

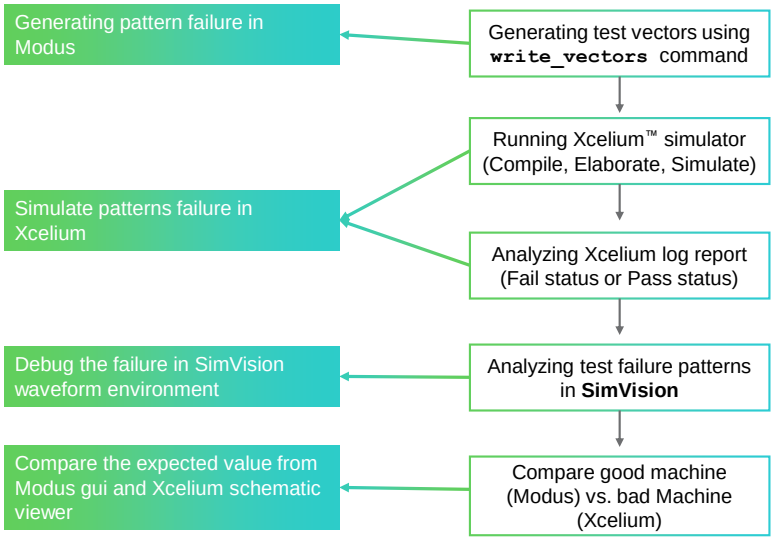
Here is an example of watch list syntax:



cadence

This page does not contain notes.

Miscompare Validation Flow



cadence

This page does not contain notes.

How to Perform Test Pattern Analysis in Waveform

Test Pattern Analysis is the process of:

- Taking data from manual patterns or a test generation process.
- Simulating the patterns to determine if the expected responses match the actual values.
- Analyzing any miscompares to determine the cause of the problem, this simulation check is performed using SimVision.

1. Generate Xcelium run script
2. Simulating the patterns
3. Analyzing the Waveforms in SimVision

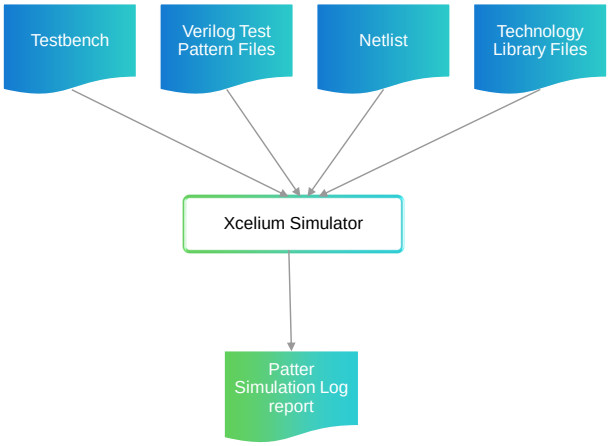
cadence

This page does not contain notes.

Step 01: Generating the Xcelium Script

Xcelium simulator helps to perform the pattern simulation.

- Use **`-writesimulationrunscript yes`** along with write_vectors command can generate Xcelium run set script for Xcelium simulation. It is supported only by Verilog language.
- The script will be dumped in
  - **`testresults/verilog/run.<testmode>.<experiment_name>_sim`**



cadence

This page does not contain notes.

Step 02: Simulating Patterns

- Pattern file invoke in testbench, observe scan and logic in the name
- Display pattern progress in the simulation log file
- Dump CPP failure data for each miscompare
- For 0 delay simulation
- Main testbench file

```
xrun \
   +access+rwc \
   +xmstatus \
   +xm64bit \
   +TESTFILE1=./testresults/verilog/VER.FULLSCAN.logic.data.scan.ex1.ts1.verilog \
   +TESTFILE2=./testresults/verilog/VER.FULLSCAN.logic.data.logic.ex1.ts2.verilog \
   +HEARTBEAT \
   +FAILSET \
   +xmtimescale+1ns/1ps \
   +xmoverride_timescale \
   +xmseq_udp_delay+2ps \
   +libext+.v+.V+.z+.Z+.gz \
   +xmlibdirname+$WORKDIR/Inca_libs_10_13_21 \
   -l $WORKDIR/xmverilog_FULLSCAN.log \
   -v ./techlib/pads.v \
   -v ./techlib/stdcell.v \
   ./netlist/DLX_CORE.v \
   ./netlist/DLX_TOP.v \
   ./testresults/verilog/VER.FULLSCAN.logic.mainsim.v \
```

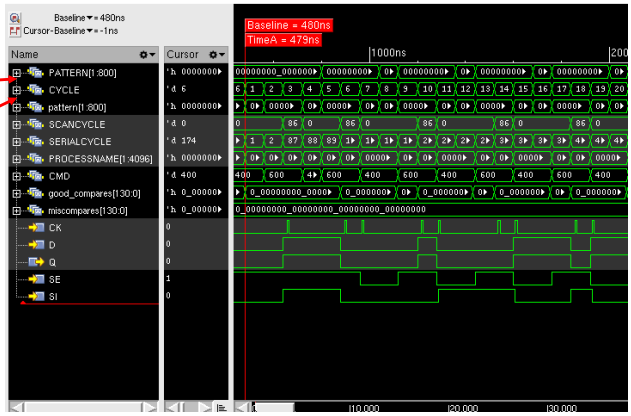
Xcelium run file required options

cadence

This page does not contain notes.

Step 03: Analyzing the Waveforms in SimVision

- **Odometer** where measure is occurring. Odometer where measure is occurring. PATTERN shows the unload values (this is used for debugging).
- **Current Tester Cycle**: Cycle in SimVision is a relative cycle in Xcelium log – at the same time, it is failing.
- CYCLE & TIME STAMP (e.g., 80,000ps) in SimVision simulation matches the relative cycle and time scale in Xcelium ncverilog.log before comparison.
- The CYCLE number reported on SO failures is the Scan Chain bit position relative to SCAN OUT.

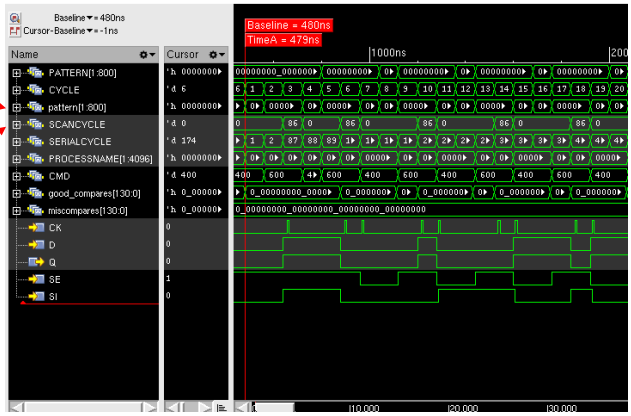


cadence

The Graphical Display allows users to trace through the circuit design. Users can move around objects by using the View options or by tracing forward and backward on objects.

Step 03: Analyzing the Waveforms in SimVision (continued)

- Current Odometer being simulated. The pattern shows the load values.
- Current Scan Cycle since start/end of scan.
- In the parallel simulation, the values we see are 0(start of scan), and 11(end of scan) since it is strobing of 1 flop. In the serial simulation, you will see it from 0,1,2,3,…..11). Current Scan Cycle since start/end of scan.
- In the parallel simulation, the values we see are 0(start of scan), and 11(end of scan) since it is strobing of 1 flop. In serial simulation, you will see it from 0,1,2,3,…..11.



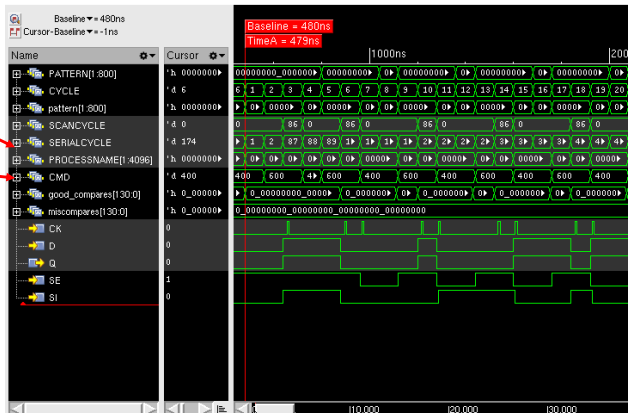
228 © Cadence Design Systems, Inc. All rights reserved.

cadence

This page does not contain notes.

Step 03: Analyzing the Waveforms in SimVision (continued)

- Current Serial Tester Cycle since the start. It is useful when we run serial simulation, not for parallel.
- Shows scan sequence or test sequence or mode_init seq.



cadence

This page does not contain notes.

Analyze Vectors

```
analyze_vectors  -TESTMODE <testmode name> -INEXPERIMENT <input experiment
Name> -EXPERIMENT <New Experiment Name> -watchnetsfile <Filename with
watchlist> -watchvectors <Range of vectors to simulate>
```

- Analyze vectors can generate the Simvision waveforms for the vectors simulated by Modus.
- Specify the nets/pins/blocks to be visualized in a watchlist and pass it to analyze vectors.

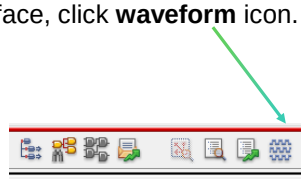
```
analyze_vectors      \
-TESTMODE FULLSCAN    \
-INEXPERIMENT logic    \
-EXPERIMENT analyzed    \
-watchnetsfile ./watchlist.txt \
-watchvectors 1.2.1.3.1:1.2.1.4.1
```

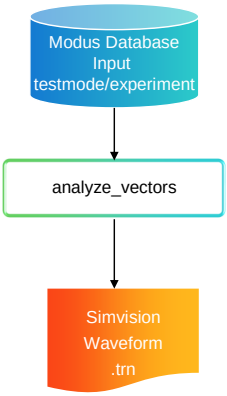

cadence

This page does not contain notes.

Result of Simulation

- The SimVision logs are stored in the tbdata directory.
  - They are named:
    - TBscope.<testmode>.<experiment>.trn
    - TBscope.<testmode>.<experiment>.dsn
- To launch SimVision, enter the following into a command-line window:
  - **simvision &**
- Or, in the graphical interface, click **waveform** icon.





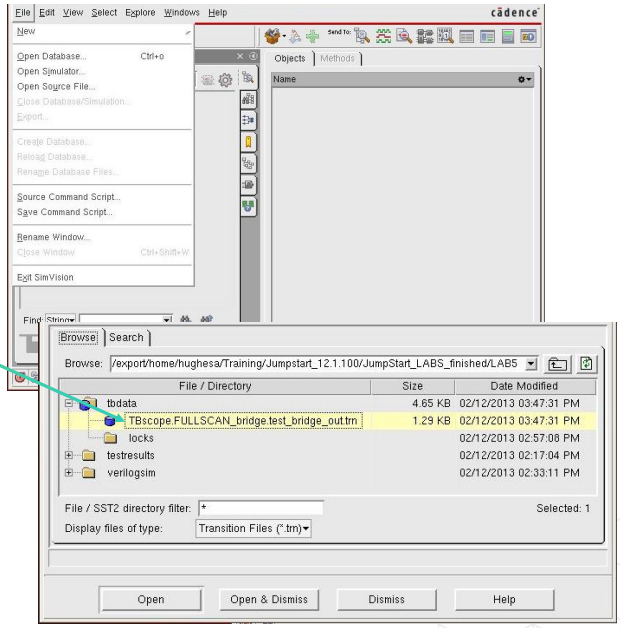
cadence

This page does not contain notes.

SimVision Graphical Interface

Once you invoke the SimVision, to open a database, choose **File > Open Database**.

To open the database, you need to browse the **tbdata** directory and select the **.trn** file.



cadence

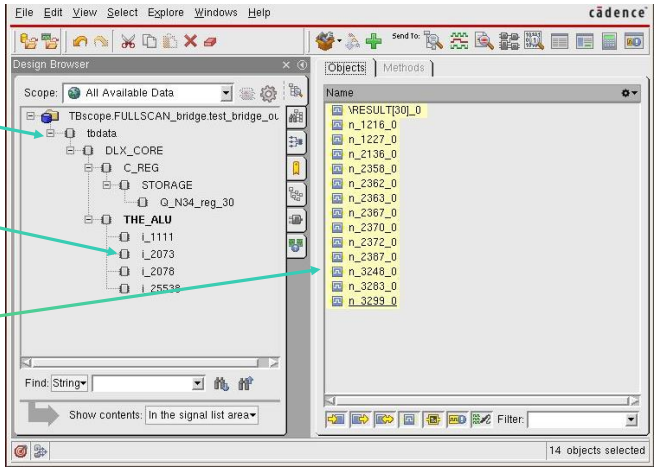
This page does not contain notes.

SimVision Interface with Loaded Database

Design hierarchy is shown and can be traced.

Nets for a particular hierarchy are shown here.

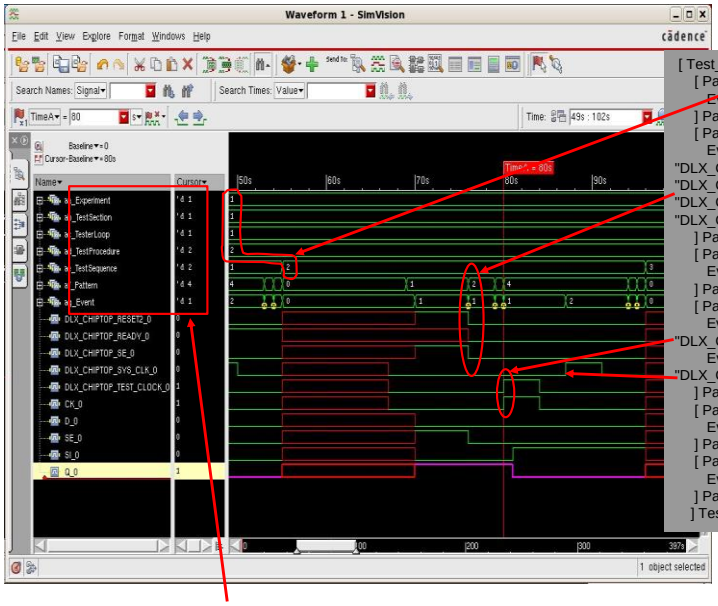
Highlighting these nets and clicking the waveform icon will show waveforms.



cadence

This page does not contain notes.

Display of Modus analyze_vectors Result

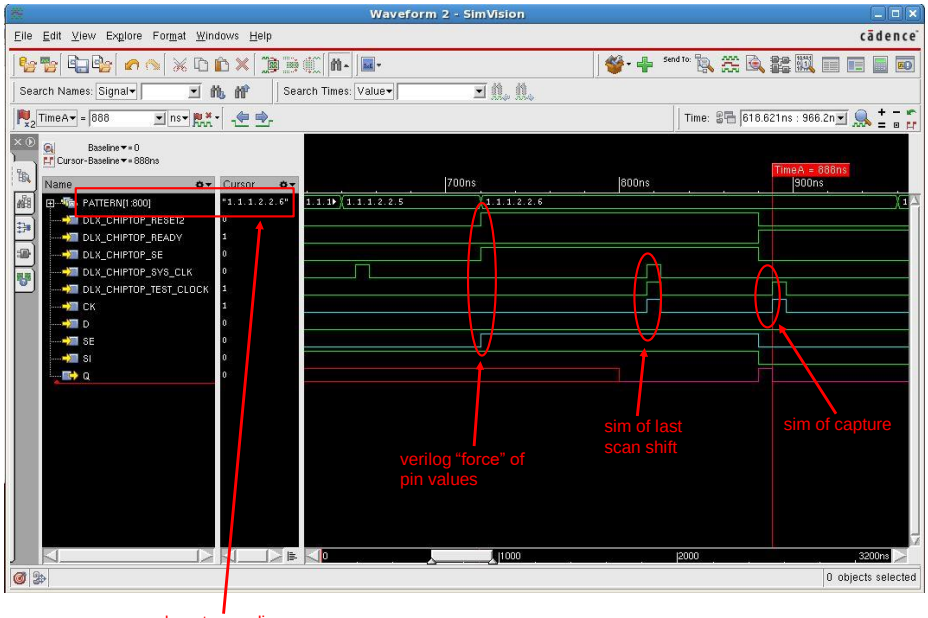



```
[ Test_Sequence 1.1.1.2.2 (); # Test 2
   [ Pattern 1.1.1.2.2.1 (pattern_type = static);
      Event 1.1.1.2.2.1.1  Scan_Load ():
   ] Pattern 1.1.1.2.2.1;
   [ Pattern 1.1.1.2.2.2 (pattern_type = static);
      Event 1.1.1.2.2.2.1  Stim_PI ():
"DLX_CHIPTOP_READY"=0
"DLX_CHIPTOP_RESET2"=0          (Set PI's for pattern)
"DLX_CHIPTOP_SE"=0
"DLX_CHIPTOP_TDO"=0 ;
   ] Pattern 1.1.1.2.2.2;
   [ Pattern 1.1.1.2.2.3 (pattern_type = static);
      Event 1.1.1.2.2.3.1  Stim_PI ():;
   ] Pattern 1.1.1.2.2.3;
   [ Pattern 1.1.1.2.2.4 (pattern_type = static);
      Event 1.1.1.2.2.4.1  Pulse ():          (capture pulse)
"DLX_CHIPTOP_TEST_CLOCK"=+ ;
      Event 1.1.1.2.2.4.2  Pulse ():
"DLX_CHIPTOP_SYS_CLK"=+ ;
   ] Pattern 1.1.1.2.2.4;
   [ Pattern 1.1.1.2.2.5 (pattern_type = static);
      Event 1.1.1.2.2.5.1  Measure_PO ():
   ] Pattern 1.1.1.2.2.5;
   [ Pattern 1.1.1.2.2.6 (pattern_type = static);
      Event 1.1.1.2.2.6.1  Scan_Unload ():
   ] Pattern 1.1.1.2.2.6;
 ] Test_Sequence 1.1.1.2.2;
```


cadence

This page does not contain notes.

Display of Verilog Parallel Load Simulation



This page does not contain notes.

Module Summary

In this module, you learned how to

- Debug the test patterns with Modus diagnostics tool
- Apply and simulate the pattern values
- Debug the waveforms with SimVision

cadence

This page does not contain notes.

Lab

Lab 6-1 Simulating and Debugging Vectors

- Invoking Xcelium and Modus
- Invoking Modus and Generating Vectors
- Running the Good Machine Simulation (Modus)
- Running the Bad Machine Simulation (Xrun)
- Debugging Simulation Miscompares
- Compare Good Machine Simulation vs. Bad Machine Simulation
- Alternate Method to Debug the Simulation

cadence

This page does not contain notes.

Module 7

Additional Definition and Concepts

cadence

This page does not contain notes.

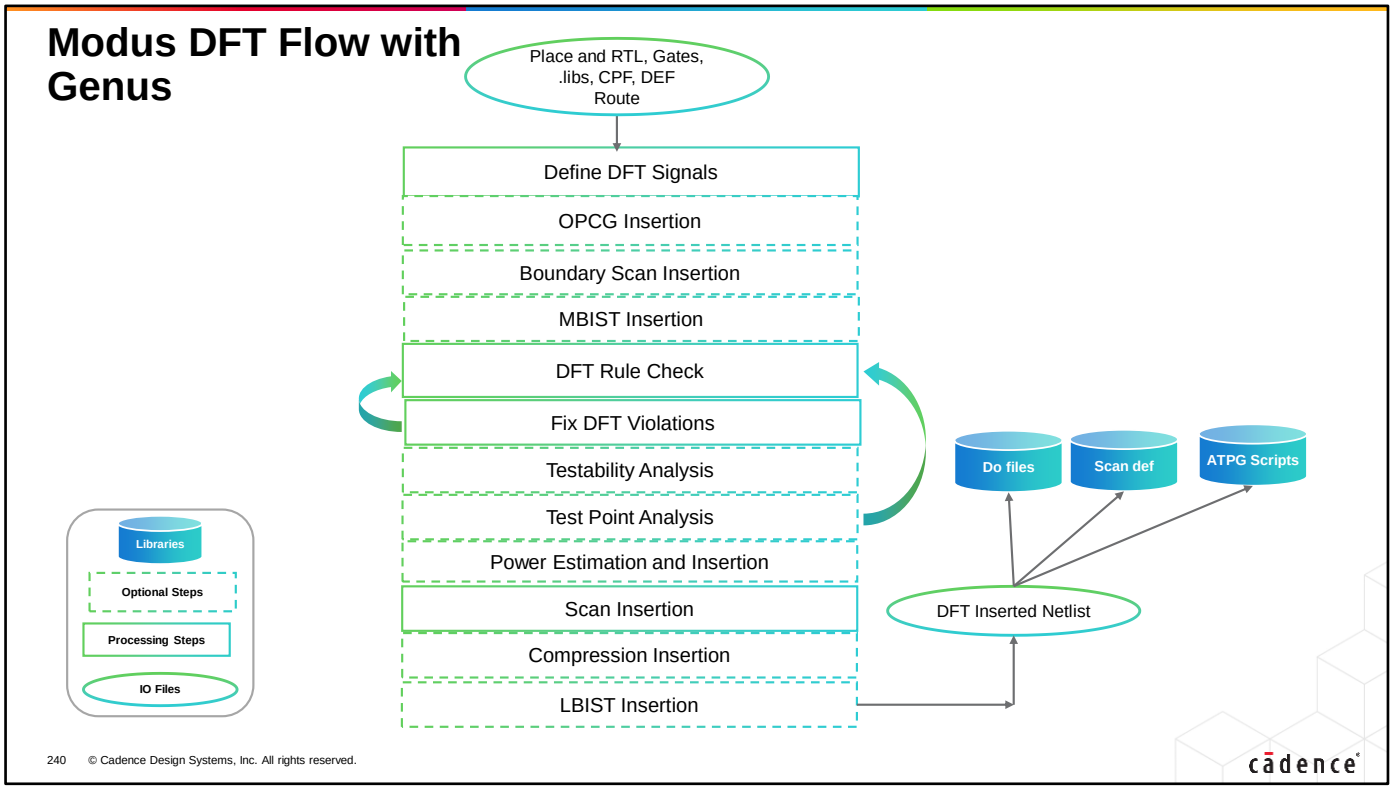
Module Objectives

In this module, you

- Identify the Genus™ – DFT use model flow
- Define Library Modelling, including
  - Flip-flops
  - Latches
  - Muxes
  - RAM
  - ROM, and resistors while building Modus models
- Define the different files and attributes such as constraints file, edit file, include file, and contention on unconnected nets
- Create structural libraries using Conformal LEC
- Explain Modus pattern faults during the fault model building process
- Differentiate between static and dynamic pattern faults
- Describe the parent-child test modes
- Describe the test compaction process

cadence

This page does not contain notes.



This page does not contain notes.

What Is Library Modeling?

Library modeling is the mapping of a Verilog netlist or library into the physical circuit or design model for the purpose of verifying functionality or conducting additional analysis of the design using different simulations. Modus has specific internal models that Verilog is mapped down to.

- Modus has special primitives that can be used to model library cells. These include:
  - Flip-Flops
  - LATCH
  - MUX
  - RAM
  - ROM
  - TSD and RESISTOR
- Using these elements, the users can create higher-level logic functions.

Example: Modeling of multiplexer in Modus

Interface definition

```
// Definition
          module __MUX2 (DOUT, DATA0, DATA1, SEL);
          input DATA0;
          input DATA1;
          input SEL;
          output DOUT;
          endmodule
// Usage
          module MUX_21( y, d0, d1, s);
                output y;
                input d1;
                input d0;
                input s;
                __MUX2 i1(y,d0,d1,s);
          endmodule
```


cadence

This page does not contain notes.

Example: Modeling of Flip-Flop and Latch

Flip-flop modeling:

- Modeling flip-flops in Modus does not require interface definition.

```
module mydff (out1, out2, out3, out4, clr, d, clk, set);
      output out1, out2, out3, out4;
      input d, clk, clr, set;
      __rDFF_C  i1(out1,clr,d,clk);
      __rDFF_S  i2(out2,d,clk,set);
      __fDFF    i3(out3,d,clk);
      __fDFF_cS i4(out4,clr,d,clk,set);
endmodule
```

Latch modeling:

```
// Definition
            module latch_1 (DOUT, P01DATA, P01DCLK);
                        output DOUT;
                        input P01DATA, P01DCLK;
            endmodule
// Usage
            module dff_fall_edge (Q, QN, D, C, R);
                        input C,D;
                        output Q;
                        not cb (C_b, C);
                        latch_1 master (mst2slv, D, C);
                        latch_1 slave (Q, mst2slv, C_b);
            endmodule
```

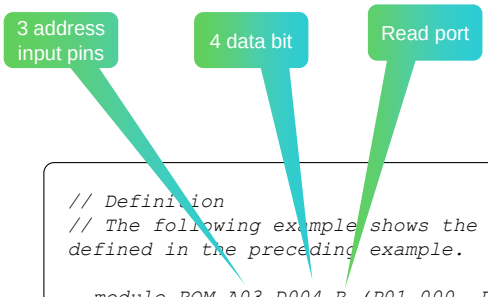
Latch modeling also supports multiple ports (2,3,4,…), for example:

```
module latch_2 (DOUT, P01DATA,
P01DCLK, P02DATA, P02DCLK);
output DOUT;
input P01DATA, P01DCLK, P02DATA,
P02DCLK;
endmodule
```


cadence

This page does not contain notes.

Example: Modeling of ROM




```
// Definition
// The following example shows the usage of the three by four ROM
defined in the preceding example.

  module ROM_A03_D004_R (P01_000, P01_001, P01_002, P01_003, P01A00,
    P01A01, P01A02, P01READ);
      output P01_000, P01_001, P01_002, P01_003;
      input P01A00, P01A01, P01A02;
      input P01READ;
  endmodule
```


cadence

This page does not contain notes.

Example: Modeling of ROM (continued)

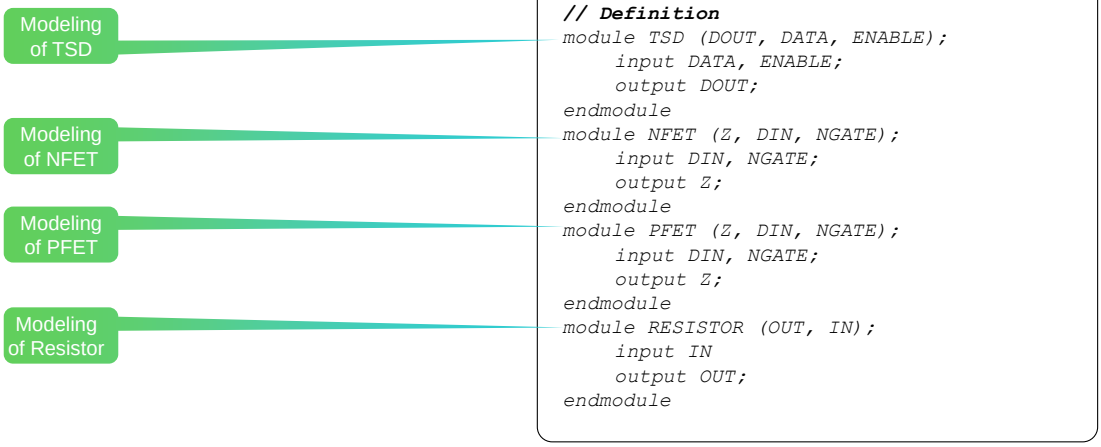
Modus recognizes cell names in the format RAM_Axx_Dyyy_tttt as RAM primitives or ROM_Axx_Dyyy_ttt as ROM primitives. The following table depicts the formats:

RAM_Axx_Dyyy_tttt	Pin Description
xx	Represents the number of address input pins.
yyy	Represents the number of data bits in a word of the RAM, and therefore, the number of data input and data output pins on each port.
tttt	It is a string with one character for each port of the RAM. The port characters are defined as: R as a READ port W as a WRITE port B as a READ/WRITE port S as a SET port

cadence

This page does not contain notes.

Example: Modeling of TSD and Transistor



cadence

This page does not contain notes.

Constraint Files

A constraint file can add logic constraints to prevent certain patterns from being generated.

- Format of the **constraint file**:
  - `[cellName:] constraintName expression;`
    - **`cellName`** is the name of the module/cell where the constraint is to be applied.
    - **`constraintName`** is the name given to the constraint.
    - expression is defined as follows:
      - An arbitrary boolean expression. The expression may combine **`netname(s)`** and the logical operators ^, &, and |, in addition to the provided functions.

Function	Description
oneHot	Exactly one net at logic 1
oneCold	Exactly one net at logic 0
zeroOneHot	Zero or one nets at logic 1
zeroOneCold	Zero or one nets at logic 0
Equiv	All nets have same value
bitwiseOneHot	Bitwise version of oneHot
bitwiseOneCold	Bitwise version of oneCold
bitwiseOneZeroHot	Bitwise version of zeroOneHot
bitwiseOneZerocold	Bitwise version of zeroOneCold
bitwiseEquiv	Bitwise version of equiv

One of the functions using a list of netnames as arguments can be used in the expression.

cadence

This page does not contain notes.

Edit Files

Edit files are used to edit the model from the command line (and GUI). Edit files can modify or delete the Pin, net, and block objects in the model.

- To delete pin attributes from the model, use the attribute:
  - `DELETE PIN ATTRIBUTE attrName1 [=attrValue1 ] [ON] PIN pinName1 [ON] CELL cellName1;`
- To change attribute names:
  - `CHANGE ATTRIBUTE NAME [FROM] attrName1 [TO] attrName2 [ON] CELL cellName1;`
- To connect/disconnect pin/nets:
  - `CONNECT PIN pinName1 [TO] NETnetName1 [IN] CELL cellName1;`
  - `DISCONNECT PIN pinName1 [ON] INSTANCE instanceName1 [FROM] NET netName1 [IN] CELL cellName1;`
- To change the test functions:
  - `CHANGE TEST FUNCTION [FROM] tf1 [TO] tf2 [ON] PIN hierpinName1;`
- To insert test points:
  - `ADD TEST POINT testPointType [AT] PIN [=] pinName1;`
- To remove faults:
  - `ADD ATTRIBUTE FAULTS=NO CELL <yourcellname>;`

cadence

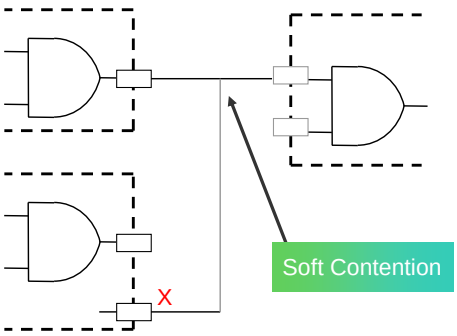
This page does not contain notes.

Fixing Contention on Unconnected Nets

Modus is often pessimistic while building test models. If a pin defined on a module is an **inout** or **output** and remains undriven, it's assumed that the pin is driven by **X**. This can cause issues if the net is multi-driven, leading to **soft contention**.

We can fix this contention issue on an unconnected net while building the fault model.

- During **`build_model`**, the message "**`TLM105 € Multi-source net`**" will be used for each occurrence.
  - If the pin is defined at "**`inout`**," then use **`build_model -i`** to fix this problem.
  - If this is an **output pin,** make pin input only or change net name.



This page does not contain notes.

Create Structural Libraries Using Conformal LEC

Our **build_model** tool is unable to understand certain constructs in the Verilog 2001 standard. In the models, connections between pins are established via the **$setuphold** timing check in the specified block instead of using an assign statement or a functional buffer.

- As a result, the following message is reported by **build_model**:
  - **TEI108 (I) No internal net is connected to input interface port CK of cell SDH_FSDPQ_1.**

The Cadence® Conformal® software can be used to clean up libraries of unused logic and solve this issue.

- The dofile.do file for the Conformal software is simple :
  1. read design <mylibrary>.v
  2. write design <my_newlib>.v –all
  3. exit -f
- To run it, enter this command:
  - **lec -dofile dofile.do**

249

cadence

Verilog 2001 has constructs not supported by Modus. The use of Cadence's Conformal tool can be used to clean up these issues.

Defining a Variable in Verilog `ifdef Statements

There are a few ways to define a variable to use in ‘ifdef statements:

- Create a file that has the following syntax:
  - `define atpgmodel
  - When in build_model, **DESIGNSOURCE=top.v:<your file>.v**
    - Must be colon concatenated with the Verilog files that need the variable.
- Another method is to change the Verilog files themselves:
  - `define atpgmodel
    - This requires you to edit your Verilog files.
- Finally, one can add the **definemacro=atpgmodel** to the build_model command.

When Modus builds a test model, the else statements will be followed by default.

```
// Example
            `ifdef atpgmodel
              buf b1 (SO,A);
            `else
              or  b2 (SO,A,B);
            `endif
```


cadence

This page does not contain notes.

Including File Statements for Verilog

When Modus builds its test model with Verilog, you can specify a list of files by using an include file.

- The list of files to include should be in a file name that ends with .v. This lets the Verilog parser process the data correctly.
- Treat this file as just another input file for your DESIGNSOURCE or TECHLIB.
- Syntax to include files for Verilog files is: **`include "<file name with correct path>"**

```
//Sample Include file myinclude.v

`include "/dir1/dir2/stdout.v"
`include "/dir1/dir2/file1.v"
`include "/dir1/dir2/file2.v"
`include "/dir1/dir2/file3.v"
```


cadence

This is the syntax for include files for verilog files. This is a way of greatly simplifying the commandline length of the build_model command. It can be used for the designsource and techlib switch options.

Including File Statements for MTVs

When the Modus builds a test model with MTVs, you can specify a list of files by using an **include** file.

- The list of files to include should be in a file name that ends with .mtv. This enables the MTV parser to process the data correctly.
- Treat this file as just another input file for your DESIGNSOURCE or TECHLIB.
- Syntax to include files for Verilog files is: **`include "<file name with correct path>"**

```
//Sample Include File myincludeFile.mtv

#include "/dir1/dir2/stdout.lib"
#include "/dir1/dir2/file1.mtv"
#include "/dir1/dir2/file2.fslib"
#include "/dir1/dir2/file3.fs"
```


cadence

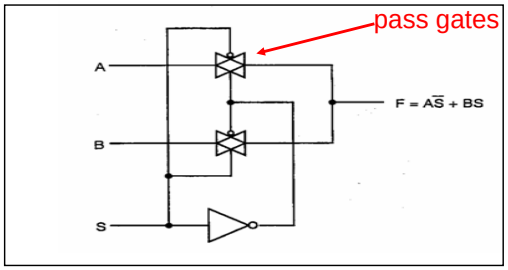
This is the syntax for include files for MTV files.

What Is Pattern Faults?

Pattern fault is a method to describe static or dynamic faults that are difficult, if not impossible, to represent using stuck-at-pin failures.

CMOS MUX with pass gates: Internal selector defects can cause high-Z or contention.

- Not detectable by Cell pin stuck-at or transition test
- Not detectable in AND-OR equivalent circuit internals
- Pattern faults detect inner circuit defects



cadence

This page does not contain notes.

Patterns Faults: Goals and Features

Pattern faults feature:

- Simple cell-level fault model is inaccurate for technology cells with internal fan-out or pass-gates, such as latch.
- Transistor-level modeling would be inefficient in representing true transistor-level faults.
- Pattern faults allow specific test sets to be applied.
  - Provides better “defect” coverage.

Goals:

- Comparisons excluding pattern faults ignore a unique differentiator of the product.
- The true target is defect coverage, minimizing test escapes.
  - Simple fault coverage measurements are only partial measurements.
  - Below 0.18 µm delay, fault coverage becomes critical.

cadence

Pattern faults can help fill in where the standard static fault model doesn’t represent the true transistor level faults.

Pattern Faults Model Anatomy

“Simple” pattern fault consists of:

- Conditions required to activate the defect.
  - Single value for static faults; value pair (initial/final) for dynamic faults
- Pin/net and good/fault machine values for fault propagation.

“**Complex”** pattern fault consists of multiple simple pattern faults.

- “ORed” pattern faults imply the defect is present when any of the simple pattern faults are detected.
- “ANDed” pattern faults imply the defect is present when all of the simple pattern faults are detected.

cadence

This shows what pattern faults consist of.

Modus Pattern Faults Modelling Technique

Pattern faults are a patented fault modeling technique unique to Modus. Its unique, patented capability:

- Model any defect whose behavior can be described in terms of input values and transitions and expected output values and transitions.
  - Example: Bridging fault between two nets:
    - Modeled by specifying stimulus values at the output of the gates that source the nets and expected values at net sinks

Modus faults model used to model:

- Path Delay
- Bridging and Cross-talk defects
- Optimal test patterns for complex cells, for example, MUX
- Fault Isolation requirements for Diagnostic ATPG, etc.
- Cell-aware defects by Cell-Aware Test (CAT) Technology
  - CAT uses parasitic annotated transistor-level netlists of standard cells to model potential open, short, and transistor-specific defect sites and identify cell-level test patterns that can detect these defects.

Modus pattern fault can be attached to any netlist object and automatically included whenever that object is used.

cadence

This page does not contain notes.

What Is Static Pattern Fault?

A static pattern fault is used to model a defect that can be detected regardless of the speed at which the test patterns are applied. A static pattern fault specifies "**REQUIRED**" and "**PROPAGATION**" values.

This is an example to model a dominant bridging fault where net A dominates net B; the pattern fault might look like…

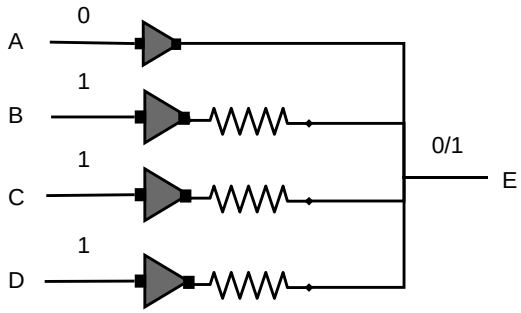
```
OR {
     STATIC {
          REQUIRED { Net A=0; Net B=1; }
          PROPAGATION { Net B=1/0; }
     }
     STATIC {
          REQUIRED { Net A=1; Net B=0; }
          PROPAGATION { Net B=0/1; }
     }
}
```


cadence

Examples of a basic OR'ed pattern fault. Static pattern faults do not have to be ORed.

Static Pattern Fault Example: Overdriving Tri-states

```
OR {
        Static {
                Required {
                        Net A=0
                        Net B=1
                        Net C=1
                        Net D=1
                } Propagation {
                        Net E=0/1
                }

        } Static {
                Required {
                        Net A=1
                        Net B=0
                        Net C=0
                        Net D=0
                } Propagation {
                        Net E=1/0
                }
        }
}
```



Example of Ored Pattern fault testing tri-states.

What Is Dynamic Pattern Fault?

A dynamic pattern fault is used to model a defect that requires a sequence of patterns to be applied to the design within a specific period. A dynamic pattern fault consists of specifying not only the “**REQUIRED**” and “**PROPAGATION**” values but also the “**INITIAL**” values.

This is an example of a potential crosstalk from nets A and B to net C; the pattern fault might look like…

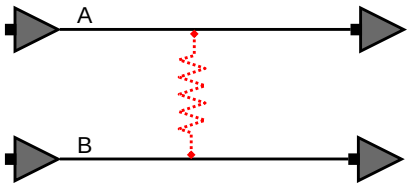
```
DYNAMIC {
     INITIAL { Net A=0; Net B=0; }
     REQUIRED { Net A=1; Net B=1; Net C=0; }
     PROPAGATION { Net C=0/1; }
}
```


cadence

This page does not contain notes.

Dynamic Pattern Fault Example: Shorted Nets

```
OR {
        Static {
                Required {
                                        Net A=0
                                        Net B=1
                } Propagation {
                                        Net B=1/0
                }
        } Static {
                Required {
                                        Net A=1
                                        Net B=0
                } Propagation {
                                        Net A=1/0
                }
        }
}

Or, using Shorthand:
   Short0 { Net A Net B }
```



cadence

This page does not contain notes.

Suppressing the Faults by Creating an Edit File

Edit files can be used to suppress faults on particular levels of hierarchy or particular blocks. The edit file does not require a model change, but this file must be used at the time the model is built. Create an **edit file** to be used at **`build_model`** to identify the blocks to no fault.

1. The following attributes can be set with **no fault** to certain cells or to certain instances in the netlist.
   - To suppress faults in all instances of cell use:
     - `ADD ATTRIBUTE FAULTS=NO CELL <yourcellname>;`
   - To suppress specific instances by using:
     - `ADD ATTRIBUTE FAULTS=NO INSTANCE "instance name" CELL <parentcellname>;`
2. Use **`build_model -editfile <file>`**, and this should be used while the model is building.

cadence

This page does not contain notes.

“No Fault” Example Attributes on Instance and Pins

Fault in netlists in an **instance** can be excluded by using:

```
//! FAULTS="no" attribute on instances
module core2
(SI1,SI2,SI3,SO1,SO2,SO3,D1,D2,D3,D4,Q1,Q2,Q3,Q4
,SG,CLK);
//! FAULTS="NO"
    input  SI1,SI2,SI3,D1,D2,D3,D4,SG,CLK;
    output SO1,SO2,SO3,Q1,Q2,Q3,Q4;
```

Faults on individual **pins** can be excluded by using the following syntax:

```
PFLT = NO, 0, 1, 01 (Pin Fault)
TFLT = NO, 0, 1, 01 (Transition Fault)
DFLT = NO (Driver Fault)
RFLT = NO (Receiver Fault)
and nb1_G
                (nb1_S001,
                 b2, //! PFLT="01"
                 b1,
                 b3 //! PFLT="0"
                );
and na1_G
                (na1_S003,
                  a2,
                  a1,
                  a3 //! PFLT="NO"
                 );
```

Faults SA0 and SA1 will be placed on pin b2 of instance nb1_G, an SA0 fault will be placed on pin b3 of instance nb1_G, and all faults will be removed from pin a3 of instance na1_G.

cadence

This page does not contain notes.

What Is Parent-Child Test Modes?

Test Modes can have a Parent-Child relationship:

- A parent mode that scans values into test control registers.
- In the child mode, the scannable register of the parent can become the fixed-value control register of the child.

Logic BIST Example of parent-child mode test mode:

- Parent mode scans PRPG, MISR, loop counter, and other Logic BIST controller registers to initialize the test.
- Child mode executes BIST, with PRPG, MISR, and the rest of the Logic BIST controller outside the scanned region.

cadence

Parent-Child test modes can be used for multiple reasons. They are commonly used for LBIST and complex clock controlled circuits. Basically it can be used when there are flip-flops that must be initialized for a test mode, but are not scannable in this test mode. Users can create a sequence of input stimuli that initializes those flip-flops and then leaves the PIs in the scan state for the test mode that needs to be defined. The final test mode is the target test mode. The process of initializing the flip-flops typically uses a scan operation in some other test mode. This other test mode is called a parent mode. In this case, you must have already defined the parent test mode.

Example: JTAG LBIST Parent and Child Initialization Sequence

Parent (JTAG) Testmode Initialization Sequence

```
TBDpatt_Format (mode=node, model_entity_form=name);
[ Define_Sequence Mode_Initialization_Sequence 1 (modeinit);
[ Pattern 1.1 (pattern_type = static);
# Initial values
 Event 1.1.1 Stim_PI ():
 "te"=0
 "se"=0
...
] Pattern;
[ Pattern 1.2 (pattern_type = static);
# Move the TAP to the Test-Logic-Reset State
 Event 1.2.1 Stim_PI ():
 "PAD_TMS_1"=1
 "PAD_TRST_1"=0;
 Event 1.2.2 Pulse ():
 "PAD_TCK_1"=+;
] Pattern;
...
] Define_Sequence Mode_Initialization_Sequence 1;
```

Child (JTAG) Testmode Initialization Sequence

```
TBDpatt_Format (mode=node, model_entity_form=name);
[ Define_Sequence Mode_Initialization_Sequence 1
(modeinit);
[Pattern 1;
Event 1.1 Begin_Test_Mode: MODE_JTAG_RUNBIST;
]Pattern 1;
[ Pattern 1.2 (pattern_type = static);
# Initial values
 Event 1.2.1 Stim_PI ():
 "PAD_TCK_1"=0
...
] Pattern;
...
[ Pattern 1.19 (pattern_type = static);
# Move the TAP to Pause-DR State
 Event 1.19.1 Stim_PI ():
 "PAD_TMS_1"=0;
 Event 1.19.2 Pulse ():
 "PAD_TCK_1"=+;
] Pattern;
] Define_Sequence Mode_Initialization_Sequence 1;
```


cadence

This page does not contain notes.

Boundary Scan Verification in Modus

```
verify_11491_boundary -TESTMODE <name> -WORKDIR <directory> -bsdlinput
<file>
```

The Modus Boundary Scan Verification (BSV) tool will verify the boundary scan test logic.

- It verifies that the BSDL (Boundary Scan Description Language) matches the structure and the structure matches the BSDL and that both are compliant with the Standard.
- It verifies full EXTEST behaviors of the Boundary Scan Register and correlation to I/Os.
- It verifies basic connections, ability to scan, and register length for user test data registers, including private instructions if desired.
  - A mix of structural (e.g., register tracing) and functional verification (e.g., simulation)
- Test Patterns can be written out as a Verilog testbench to debug problems.
- Error messages support full graphical display with simulation values for fast debugging.

265

cadence

The Modus Boundary Scan Verification (BSV) tool will verify the boundary scan test logic.

Boundary scan verification will verify that the structures on the design and the BSDL match, and that both are compliant with the IEEE 1149.1 and 1149.6 Standards. In particular, it exercises all of the EXTEST behaviors and verifies the correlation between I/O ports and cells in the boundary scan register.

For other instructions and test data registers—including user-specified instructions and test data registers—basic connections, the ability to scan, and length of each register is verified. Private instructions may be verified or not, as the user desires, as long as the “Register_Access” attribute has been specified.

The tool uses a mix of simulation and structural verification. In all cases, the BSDL is used to customize the simulation vectors and the structural checks. Any resulting errors detected by structural tests may be debugged using a full graphical display with overlay of simulation values. All test patterns may be output as a Verilog test bench to permit further verification and debug of problems detected only by simulation checks.

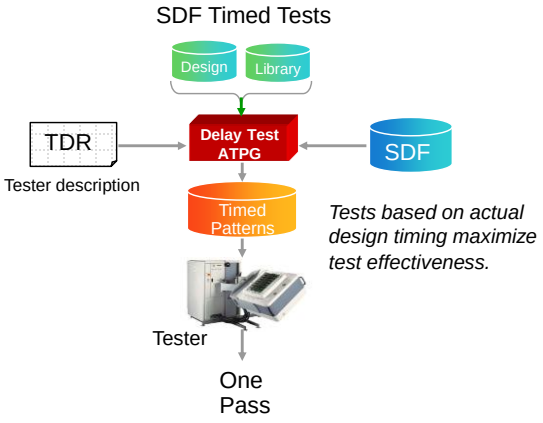
BSDL (Boundary Scan Description Language) is recommended by the IEEE 1149.1 standard to specify boundary scan. It is usually generated by the DFTS tools that insert the boundary scan.

- Used for any 1149 modes.
- Replaces Pin Assignment information as well as defining boundary register, user commands and TDRs, etc.
- Coding per IEEE 1149.1 Standard, is not covered here.

Generating Transition Test Pattern

Modus DFT Solution generates our transition tests differently than the rest of the industry.

- Modus uses a Standard Delay File (SDF):
  - It generates timed tests based on SDF.
  - MCPs tested if they meet single cycle timing.
  - Adjusts clock/control PI times to compensate for actual signal distribution delays and skews.
- Single pass ATPG generation flow.
- Syntax to create the transition test pattern:
  - **`create_logic_delay_tests -testmode<name> -EXPERIMENT <name> -DELAYMODEL <name>`**



This page does not contain notes.

What Is Test Compaction?

Test compaction is the process of reducing the number of test patterns in a test set without decreasing the fault coverage. Reducing test patterns in a test set will reduce the time to test the chip.

At this point, we've covered how the tests for each fault are generated and the role that scan plays in making that happen. The next step is to compact all these individual tests into a smaller number of test patterns.

Single fault tests are merged, or "compacted," into a single pattern if:

- They do not conflict (different values at the same location).
- They have the same clock (launch/capture) sequence.

When no more tests are merging into a pattern, the remaining unspecified pattern bits are filled (usually with pseudo-random data).

The pattern is then fault simulated against all faults (not just those targeted), and the detected faults are marked as tested in the fault list.

cadence

This page does not contain notes.

Filling of Don’t Care Bits

During the process of generating ATPG patterns, there are usually a number of scan bits that are not specified to a particular value. These bits are usually randomly filled to get fault coverage through serendipity. Instead of random fill, Modus has the option to do constant fill or repeat fill.

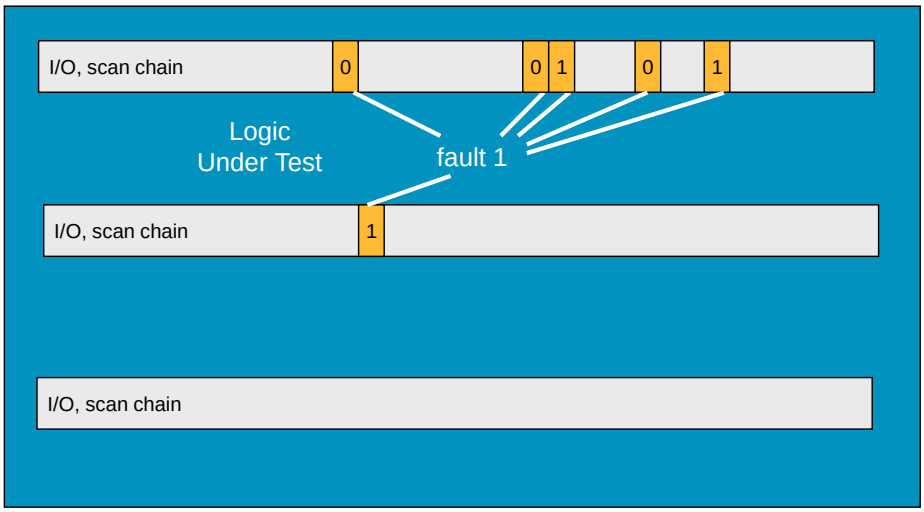
- Pseudo-random-fill:
  - Typical ATPG approach
  - Hard to compress mostly random data
- Constant-fill:
  - 0, 1, or X values
- Repeat-fill (repeat last known care-bit value):
  - Reduces randomness and switching during scan
  - Allows “Run Length Encoded” (repeat-count) vectors

cadence

This page does not contain notes.

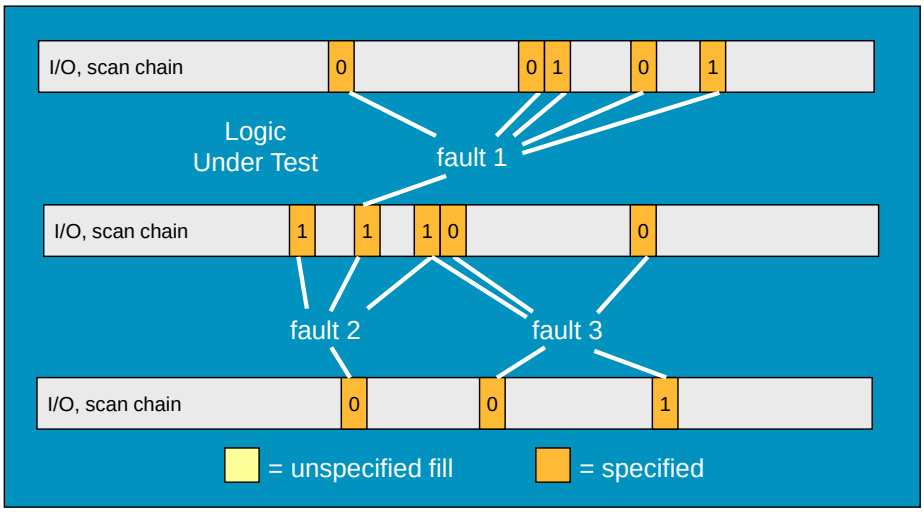
Test Compaction to Create Patterns

On large chips, specified bit densities average about 1% to 2%. A few patterns will have 20% to 90% specified bit densities.



For an average pattern of a large chip, the density of the specified bits will be less than 2%. There will be a few patterns in the set that have specified bit densities in the 20% - 90% range, and many that have densities well below 1%. The first few patterns will have very high density and test many faults. Here we show the values to be loaded for a single pattern into three scan chains in a chip that is being tested. The test generator creates a test for fault 1, with just a few bits specified, and this is passed to the compaction program. Perhaps this is the first test cube, as tests for single faults are called, let's say this test cube doesn't fit in any existing vector, so the compactions program creates a new vector that initially contains just this one test cube.

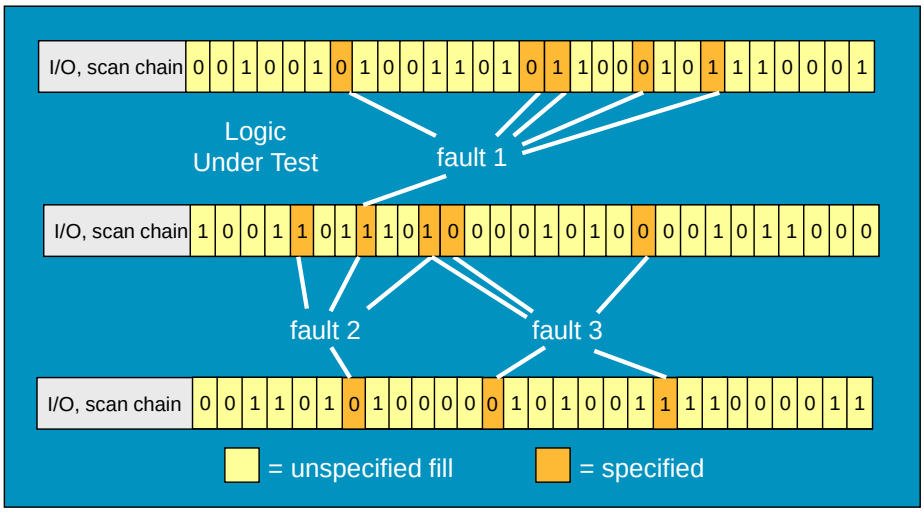
Test Compaction to Create Patterns (continued)



Tests for additional faults specify additional bits as they are merged. Note that the different tests may share some specified bits.

Test Compaction to Create Patterns (continued)

A test pattern is mostly “don’t cares” with random fill.



After some time with no additional tests merging into this pattern, it will be filled and sent to the fault simulator.

Note that most of the data that will be scanned into the chip is fill, usually pseudo-random data. This low density of specified bits has two important consequences:

1. Most of the test data is fill, and many of the faults that will be detected in the fault simulator will be detected by this fill data, especially pseudo-random fill. This happens because the fill “accidentally” sensitizes and propagates a fault effect.
2. The low density of specified bits allows us to compress the test patterns. The patterns will be decompressed and filled by on-chip logic. Compression is actually performed as part of the compaction of the individual tests, and details are covered elsewhere. However, compressed test patterns may have average specified bit densities in the 20% to 80% range, instead of the 1% to 2% range.

Read and Simulate Vectors

```
read_vectors
-language stil|tbdpatt|evcd \
-EXPERIMENT <name>          \
-TESTMODE   <name>
```

Read vectors that are created before and present in the database.

```
Simulate_vectors \
-INEXPERIMENT  <in name> \
-EXPERIMENT    <out name> \
-TESTMODE      <name>
```

Simulate vectors are the recommended approach to simulate and read patterns in SimVision™ database for the required pins.

```
Read_vectors –testmode FULLSCAN –
language wgl –experiment LOGIC
```

```
simulate_vectors -TESTMODE FULLSCAN \
-INEXPERIMENT logic                 \
-EXPERIMENT analyzed
```


cadence

This page does not contain notes.

Fault Grading Functional or Existing Patterns

Test simulation can simulate existing patterns:

- Patterns can be functional or previously generated, any source.
  - Read in STIL, EVCD, and TBDpatt (ASCII)
- Good machine and fault machine simulations available.
- Fault machine simulation used to "fault grade" patterns.
- Can provide a "second opinion" simulation.
- Can verify expected values vs. simulated values (miscompares).
- Can generate viewable test patterns:
  - SimVision waveform file generation via General Purpose Simulator

cadence

This page does not contain notes.

Reading the Test Sequence Definition File

```
read_sequence_definition –TESTMODE <name> \
-importfile     <test_sequence_file name>
```

- Reading the sequence definition file defined the test sequence into a given testmode.
- When running ATPG, the following is the keyword to add to the command line:
  - **-testsequence <sequence name>**
    - For example:
      - **-testsequence DoubleClock**

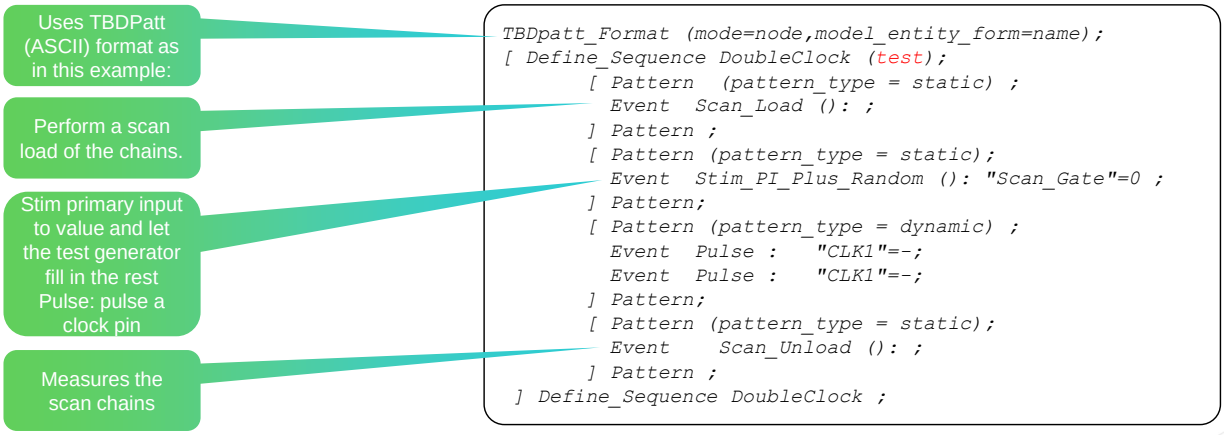
```
read_sequence_definition –TESTMODE
FULLSCAN -importfile
<test_sequence_file name>
```


cadence

This page does not contain notes.

Example: Sequence Definition File

The test sequence file can guide the test generator to use a specific clocking structure for ATPG.



cadence

This page does not contain notes.

Module Summary

In this module, you learned how to

- Identify the Genus – DFT use model flow
- Define Library Modelling, including
  - Flip-flops
  - Latches
  - Muxes
  - RAM
  - ROM, and resistors while building Modus models
- Define the different files and attributes such as constraints file, edit file, include file, and contention on unconnected nets
- Create structural libraries using Conformal LEC
- Explain Modus pattern faults during the fault model building process
- Differentiate between static and dynamic pattern faults
- Describe the parent-child test modes
- Describe the test compaction process

cadence

This page does not contain notes.

Module 8

Course Conclusions

cadence

This page does not contain notes.

Summary

In this course, you learned to

- Introduce the Modus DFT Software Solution
- Define the ATPG (Automatic Test Pattern Generation) flow
- Build a test model (building the Modus DFT Software Solution design database)
- Build the fault model
- Build test modes
- Verify test structures (design rule checking)
- Create static tests (Automatic Test Pattern Generation)
- Write the vectors (Verilog, STIL, WGL)
- Debug the broken scan chains using the GUI and Tcl command-line techniques
- Debug the test patterns

cadence

This page does not contain notes.

Module 9

Next Steps

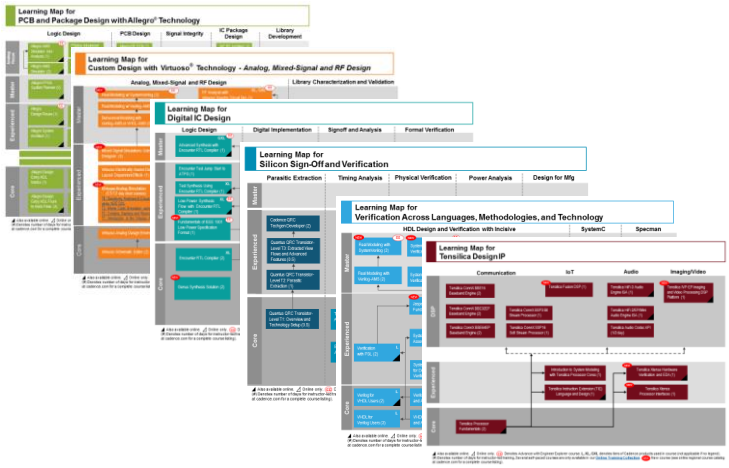
cadence

This page does not contain notes.

Learning Maps

Cadence® Training Services learning maps provide a comprehensive visual overview of the learning opportunities for Cadence customers.

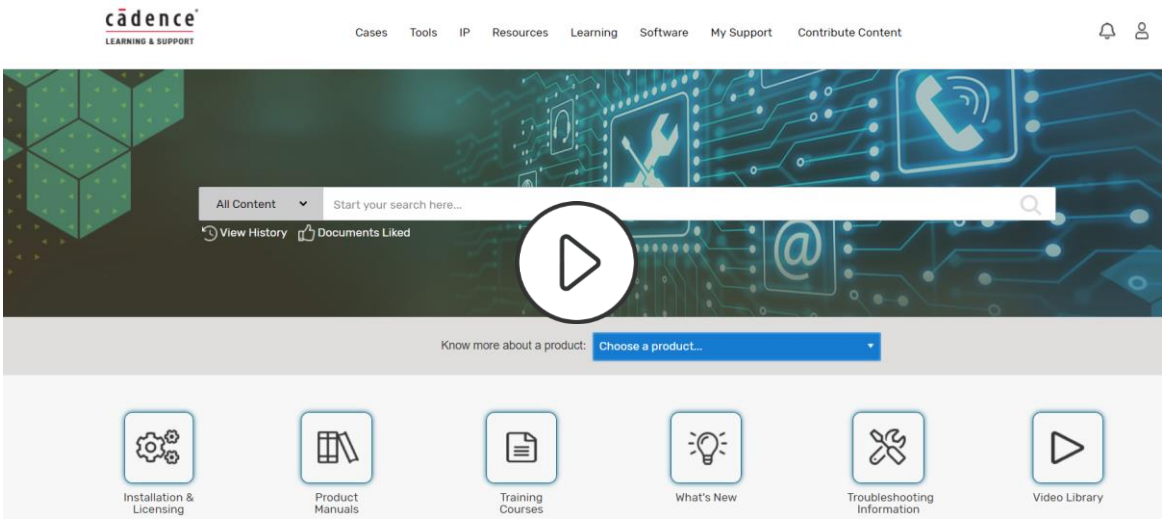
Click here to see all our courses in each technology area and the recommended order in which to take them.



Go here to view the learning maps:

http://www.cadence.com/Training/Pages/learning_maps.aspx

Cadence Learning and Support



Customer Support now includes over 2000 product/language/methodology videos ("Training Bytes")!

Click the play button in the figure on this slide to view the demo of Cadence Learning and Support.

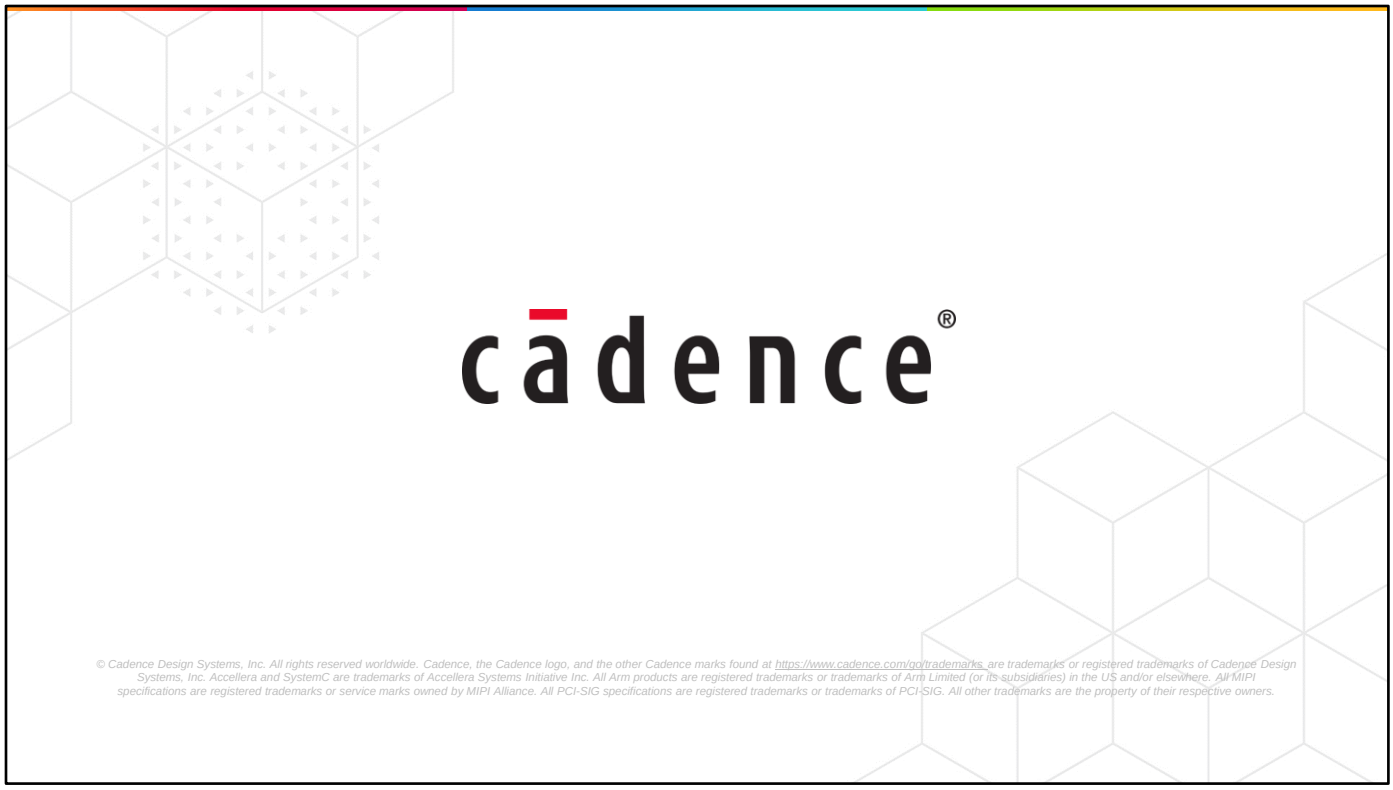
Wrap Up

- Complete Post Assessment, if provided
- Complete the Course Evaluation
- Get a Certificate of Course Completion

Thank you!

cadence

This page does not contain notes.



This page does not contain notes.