cādence®

Encounter® Test: Guide 3: Test Structures

Product Version 12.1.101
February 2013

Contents

List of Tables

List of Figures

Preface

About Encounter Test and Diagnostics

Encounter® Test uses breakthrough timing-aware and power -aware technologies to enable customers to manufacture higher-quality power-efficient silicon, faster and at lower cost. Encounter Diagnostics identifies critical yield-limiting issues and locates their root causes to speed yield ramp.

Encounter Test is integrated with Encounter RTL Compiler global synthesis and inserts a complete test infrastructure to assure high testability while reducing the cost-of-test with on-chip test data compression.

Encounter Test also supports manufacturing test of low-power devices by using power intent information to automatically create distinct test modes for power domains and shut-off requirements. It also inserts design-for-test (DFT) structures to enable control of power shut-off during test. The power-aware ATPG engine targets low-power structures, such as level shifters and isolation cells, and generates low-power scan vectors that significantly reduce power consumption during test. Cumulatively, these capabilities minimize power consumption during test while still delivering the high quality of test for low-power devices.

Encounter Test uses XOR-based compression architecture to allow a mixed-vendor flow, giving flexibility and options to control test costs. It works with all popular design libraries and automatic test equipment (ATE).

Typographic and Syntax Conventions

The Encounter Test library set uses the following typographic and syntax conventions.

- Text that you type, such as commands, filenames, and dialog values, appears in Courier type.

  Example: Type `build_model -h` to display help for the command.

- Variables appear in Courier italic type.

  Example: Use `TB_SPACE_SCRIPT=`*`input_filename`* to specify the name of the script that determines where Encounter Test binary files are stored.

- Optional arguments are enclosed in brackets.

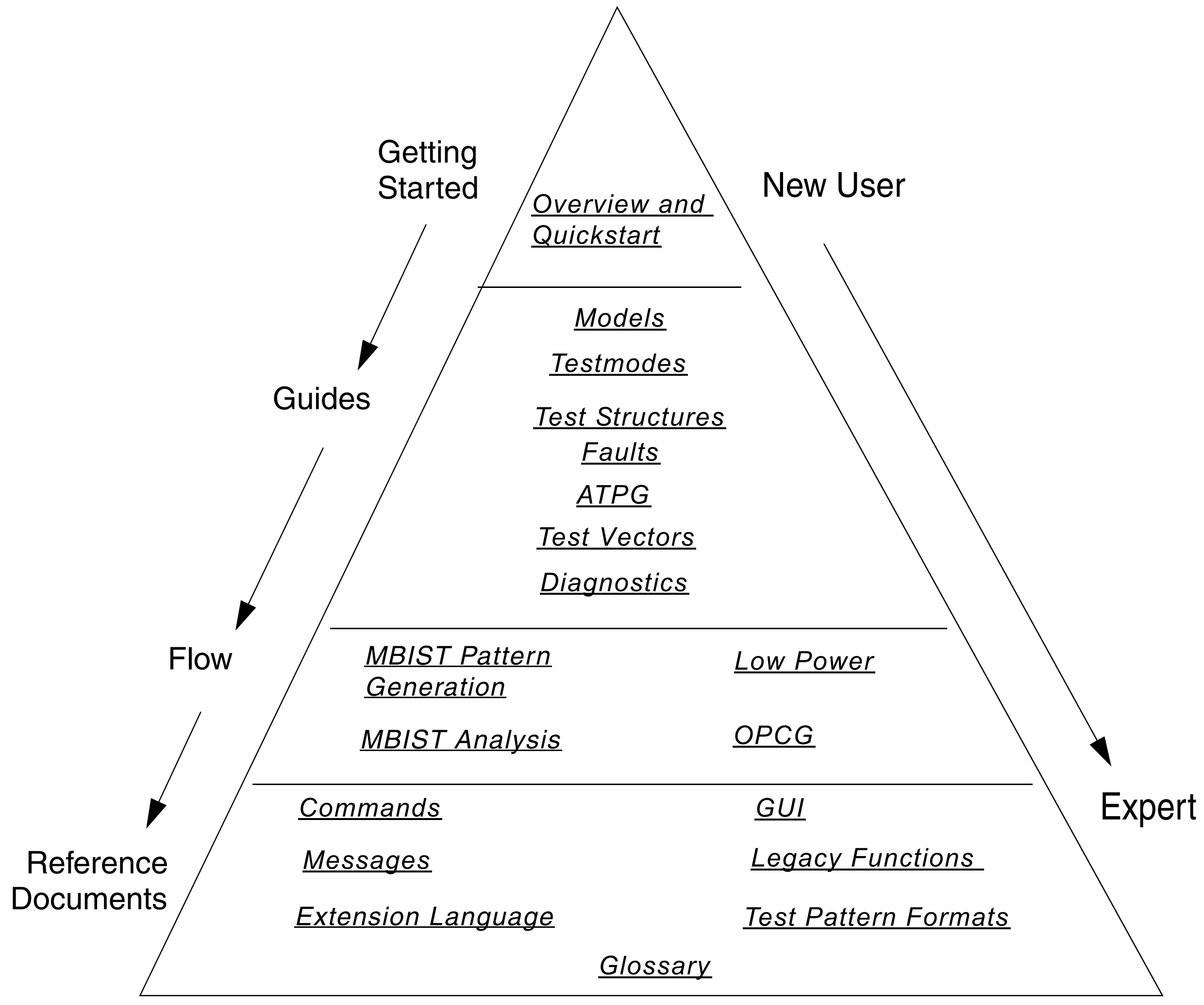
Example: `[simulation=gp|hsscan]`

- User interface elements, such as field names, button names, menus, menu commands, and items in clickable list boxes, appear in Helvetica italic type.

  Example: Select *File - Delete - Model* and fill in the information about the model.

Encounter Test Documentation Roadmap

The following figure depicts a recommended flow for traversing the documentation structure.



Getting Help for Encounter Test and Diagnostics

Use the following methods to obtain help information:

1. From the *<installation_dir>*`/tools/bin` directory, type `cdnshelp` at the command prompt.
2. To view a book, double-click the desired product book collection and double-click the desired book title in the lower pane to open the book.

Click the *Help* or *?* buttons on Encounter Test forms to navigate to help for the form and its related topics.

Refer to the following in the *Encounter Test: Reference: GUI* for additional details:

- "Help Pull-down" describes the *Help* selections for the Encounter Test main window.
- "View Schematic Help Pull-down" describes the Help selections for the Encounter Test View Schematic window.

Contacting Customer Service

Use the following methods to get help for your Cadence product.

- Cadence Online Customer Support

  Cadence online customer support offers answers to your most common technical questions. It lets you search more than 40,000 FAQs, notifications, software updates, and technical solutions documents that give step-by-step instructions on how to solve known problems. It also gives you product-specific e-mail notifications, software updates, service request tracking, up-to-date release information, full site search capabilities, software update ordering, and much more.

  Go to http://www.cadence.com/support/pages/default.aspx for more information on Cadence Online Customer Support.

- Cadence Customer Response Center (CRC)

  A qualified Applications Engineer is ready to answer all of your technical questions on the use of this product through the Cadence Customer Response Center (CRC). Contact the CRC through Cadence Online Support. Go to http://support.cadence.com and click the *Contact Customer Support* link to view contact information for your region.

- IBM Field Design Center Customers

Contact IBM EDA Customer Services at 1-802-769-6753, FAX 1-802-769-7226. From outside the United States call 001-1-802-769-6753, FAX 001-1-802-769-7226. The e-mail address is edahelp@us.ibm.com.

Encounter Test And Diagnostics Licenses

Refer to “Encounter Test and Diagnostics Product License Configuration” in *Encounter Test: Release: What’s New* for details on product license structure and requirements.

Using Encounter Test Contrib Scripts

The files and Perl scripts shipped in the `<ET installation path>/ etc/tb/contrib` directory of the Encounter Test product installation are not considered as "licensed materials". These files are provided AS IS and there is no express, implied, or statutory obligation of support or maintenance of such files by Cadence. These scripts should be considered as samples that you can customize to create functions to meet your specific requirements.

What We Changed for This Edition

There are no significant modifications specific to this version of the manual.

1 Verify Test Structures

Verify Test Structures: An Overview

Verify Test Structures checks the design for compliance with the appropriate test guidelines. The appropriate guidelines are determined by test mode characteristics such as the type of scan, either Level Sensitive Scan Design (LSSD) or General Scan Design (GSD), and the type of tests to be created. These characteristics are specified in the mode definition file used as input to the `build_testmode` command.

Verify Test Structures does the following:

- Performs a series of individually selectable tests that are defaulted according to the testmode characteristics. The various sections in the chapter describe each of these tests.

  The Verify Test Structure tests check for:

  - Clocks' control of memory elements
  - Scan chains' controllability and observability
  - Conditions that might cause manufacturing test problems (such as 3-state contention)
  - Conditions that might cause reduced test coverage (such as feedback)

  Verify Test Structures uses a combination of design-state setting, tracing, simulation, and justification techniques to perform the tests. You can use the optional keywords or menu options to adjust the effort used for these techniques.

- Issues messages in case of any deviation from the Encounter Test guidelines. The messages are prefixed with TSV and the level of the message (INFO, WARNING, WARNING [Severe], or ERROR) indicates how well the design complies with the guidelines.
- Provides an interface that allows interactive analysis of the TSV messages. Encounter Test displays the area of logic involved in the guideline violation on the schematic. The pins or blocks with the detected violations are marked in red (clock pins are marked in

pink). You can use this information in conjunction with other GUI analysis capabilities to determine the source of the problem. The GUI provides the interface to emulate the design-state setting, tracing, simulation, and justification techniques on the schematic browser.

Use the verify_test_structures command to perform Verify Test Structures from the command line, as shown below:

```
verify_test_structures workdir=< directory> testmode=< name> [other options]
```

where:

- workdir is the name of the working directory. This option is required unless you are invoking the command from the working directory.
- testmode is the name of the testmode and is a mandatory option

Refer to verify_test_structures in the *Encounter Test: Reference: Commands* for more information.

To perform Verify Test Structures from GUI:

1. Open the *Verify Test Structures* form by either selecting the task from your methodology or from the pulldown menu (V*erification - Verify - Test Structures*).
2. Fill in the name of your testmode and click *Run*. All other selections are optional and are discussed later in the chapter.
3. View the status and log for your running task from the *Methodology* window (if you invoked it from there) or from *Task View*.

Refer to Verify Test Structures in the *Encounter Test: Reference: GUI* for more information.

Selecting Tests for Performing Verify Test Structures

The testmode characteristics determine the default tests that Verify Test Structures performs. Each testmode has a basic set of tests associated with it and additional tests are performed, by default, for special testing techniques. You can use the available options to select additional tests to be performed or to deselect any of the default tests.

All the test selection keywords on the command line start with `test`, for example, `testclockusage`. Table 1-1 shows the performed tests based on testmode characteristics and the `verify_test_structures` keyword used to enable or disable that test.

To select tests from Verify Test Structures GUI, select the testmode and click the *Tests* button. This opens a tabbed window containing a selectable list of all the possible tests with the default tests, based on the testmode characteristics, already selected. Click *Select Tests* or *Deselect Tests* and click individual buttons. When your selections are completed, click *OK*.

Table 1-1 Verify Test Structures Default Tests based on Testmode Characteristics

TEST_TYPES VALUES	Default TSV Checks	Keyword to Enable/ Disable Test at Runtime
All testmodes	Analyze three-state drivers for contention Analyze feedback loops and keeper devices Analyze test clocks' control of memory elements	testtsdcontention testfeedback testclockusage
All testmodes except those specifically for Core Test Data Migration	Analyze flip-flop and latch scan characteristics Analyze clock choppers Analyze potential clock signal races Analyze explicit fixed value latches	testregsusage testclockchopper testclocksignalraces testexplicitfvregs
Testmodes with OPCG definition (CUTPOINTS and ASSIGN PPI statements are explicitly defined in the mode definition and/or are implied through definition of the OPCG statement).	Analyze On-Product Control Logic	testopclogic

TEST_TYPES VALUES	Default TSV Checks	Keyword to Enable/ Disable Test at Runtime
For OPMISR compression, LBIST, or WRPT testmodes; where getting an X into the signature register is a concern. (TEST_TYPES in the mode definition includes `signatures=yes`, `signatures=check`, or `signatures=only` on one or more of the selected types of tests to be generated).	Analyze X-sources to ensure they cannot be observed	testxsource
LBIST or OPMISR compression testmodes where PRPGs or MISRs are included in the design. (mode definition SCAN statement includes `IN=onboard` and/or `OUT=onboard)`	Analyze on-board stimulus generation (PRPGs) and signature collection (MISRs)	testprpgmisr
LBIST, OPMISR compression, or Compression testmodes	Analyze internal scan chains	testinernalscanchains
LBIST or WRPT testmodes. (ONBOARD_PRPG statement in the mode definition (for LBIST) or PRPG_DEF statement in the TDR (for WRPT) includes `fast_forward=yes`)	Analyze LBIST fast forward support	testfastforward
Reduced Pin Count Test (RPCT) testmodes (`SCAN TYPE BOUNDARY=INTERNAL` in the mode definition)	Analyze non-test pins to ensure they cannot be observed.	testbdyscan

Limiting Runtime and Messages

Encounter Test supports many runtime limiting and message limiting keywords for `verify_test_structures`. These keywords enhance the ability to find and fix problems quickly.

How are the messages related to the test?

When you run perform verify test structures either from command line or GUI, the log displays the label of the test it is going to be performed. All the messages reported after this label up to the next label are a result of the checks done during that test.

Do I have to wait until verify_test_structures completes before starting analysis?

When a specific test is performed, the WARNING messages are written into the database for analysis. Therefore, when see the label for the next test, you can start analyzing the messages that have already been produced. If you see additional tests being completed since you began analysis, you can refresh the message summary to include the additional messages (click the *Messages* icon on the main GUI window to refresh the summary).

Do I lose everything if verify_test_structures is terminated before it ends?

After performing a specific test, Encounter Test writes its status to the database. If the run terminates, all the completed tests are tracked and are not performed again when you rerun verify_test_structures. The logic behind this is that if you do not change the testmode, then the result of running verify_test_structures will be the same so there is no need to run the tests multiple times.

If I want to run an additional test after completing verify_test_structures, do I have to run everything again?

If you rerun verify_test_structures, the tests completed in the previous run will not be performed again. This allows you to run a set of tests, and then run a different set of tests without having to wait for the previous tests to be rerun.

If I need to run a different version of the verify_test_structures code on my testmode, how do I rerun all the tests (since the result may be different):

Use the `reruntests` keyword for this. If you use the command `verify_test_structures reruntests` (or `reruntests=yes`), Encounter Test removes all the previous status and messages and restarts the tests as if they were never run.

Note: If you know the specific test for which the code is changed, you can run just that test using the `reruntests` keyword (see Command Line Usage Notes and Examples on page 21).

If I make changes to the model and/or testmode and then rerun verify_test_structures, will it remember my previous results or do I have to use the reruntests keyword in such case?

No, verify_test_structures does not remember the previous results; and you do not need to use `reruntests`. After rebuilding the model or testmode, Encounter Test removes all data associated with that model or testmode. Therefore, when you run verify_test_structures on the new testmode, it will start from the beginning and run all the selected tests.

I get many messages about broken scan chains though the scan chain connections are correct and the problem is because of the clocking. Is there a way to avoid these messages?

If you run `verify_test_structures stoponerrorclockusage=yes`, Encounter test stops the processing when it detects clocking errors. You can fix these errors in the model or testmode and then run verify_test_structures without using the `stoponerrorclockusage` keyword. The clocking checks will be rerun because the testmode has been rebuilt, but if you have successfully corrected the real clocking problems, you should not get any other problems such as broken scan chains unless they are caused by something other than clocking.

My log gets very large because of the number of messages reported in the early stages of the design. Can I limit the number of messages in the log?

The following are some of the methods to limit messages in verify_test_structures:

- `reportspecific=no` - This option eliminates all the specific messages in the log. The messages are still written to the database at the end of every test. The analysis summary at the end of the log shows the cumulative number of each type of detected violation for this testmode. Even when you use `reportspecific=no`, the summary will include the messages that would have been generated for this run. When you perform the interactive analysis, all the specific messages generated by the last set of tests selected are displayed.
- `messagecount` - Several options start with the word messagecount. In addition to limiting the printing of the messages, for some of the long running tests, these options also stop the processing as soon as possible. In such case, if the processing stops before the end of the test, the status for that test is not set and the test will be rerun if verify_test_structures is invoked again. Using `messagecount`, you can limit messages by:

- ❑ severity (`messagecountinfo`, `messagecountwarning`, `messagecounterror`)

  For example, to print no INFO messages and only the first 50 WARNING messages, use `messagecountinfo=0 messagecountwarning=50`

- ❑ message number (`messagecount=messagenumber=max_number_to_process`).

  For example, to print TSV-093 for only the first 25 occurrences, use `messagecount=093=25`

- ❑ total messages (`messagecounttotal, messagecounteach`).

  For example, to print a maximum of 15 of each different TSV messages and limit the total number of messages in the run to 150, use `messagecounttotal=150 messagecounteach=15`.

- `suppressmsg` - If you have specified messages to be suppressed with attributes in the Verilog, with attributes added with Model Edit, or by editing the TEImsgOverrides file or the TEImsgOverrides2 file, then specify `suppressmsg=yes` to stop these messages from being printed in the log.

I get WARNING messages while processing some modules. However, I know from the functional simulation/timing analysis that the Encounter Test simulation will interpret the behavior correctly. Is there a way to eliminate these messages so that they do not appear when the module is instantiated at a higher level?

- You can put an attribute on the module or on an instance inside the module (SUPPRESS_MSG). When you process the higher level design using `verify_test_structures suppressmessage=yes`, a single message will print to indicate that the messages have been suppressed but the individual instances will not be printed.
- When designing technology library elements (modules with `TYPE=CELL`), you can write the `SUPPRESS_MSG` attributes so that `suppressmessage=yes` is not required and the suppression is completely silent.

  Refer to TSV Message Controls on page 73 for more information on attributes.

Command Line Usage Notes and Examples

- Only testmode is required for running verify test structures using all the defaults (workdir will be set to default if you are in the project directory, otherwise it is also required).

```
verify_test_structures workdir=myworkdir testmode=mytestmode
```

- To verify test structures for all the default tests and some additional test(s), specify the keywords for the additional tests (testxsources is additional in the example given below).

  ```
  verify_test_structures workdir=myworkdir testmode=mytestmode
  testxsources=yes
  ```

- To verify test structures for some selected test(s), turn off the default tests and then specify the tests you want to run (testclockusage is the only test selected in the example given below).

  ```
  verify_test_structures workdir=myworkdir testmode=mytestmode testdefaults=no
  testclockusage=yes
  ```

- To verify test structure for all tests except some selected test(s), specify the tests you do not want to run (all default tests except testclockusage are selected in the example given below).

  ```
  verify_test_structures workdir=myworkdir testmode=mytestmode
  testclockusage=no
  ```

- To vary the amount of effort verify_test_structures uses for tracing, simulation and/or justification, use the `effort` keyword. You can increase the effort if you get messages specifying that the checking is aborted. You can reduce the effort to make it run faster (but the results may be more difficult to use).

  ```
  verify_test_structures workdir=myworkdir testmode=mytestmode effort=high
  ```

Inputs and Outputs

Required Input

Encounter Test tbdata information output from build_model and build_testmode.

Optional Input

- tbdata/TEImsgOverrides - This file, created by `build_model`, contains message suppression information. It is initialized based on `SUPPRESS_MSG` or `RAISE_MSG_SEVERITY` attributes in the netlist or entered as Model Edit commands. Refer to “build_model editfile keyword in the *Encounter Test: Reference: Commands* for more information on attributes with Model Edit.

  Note: An expert user may be able to edit the TEImsgOverrides file to add or delete message suppression information before running Verify Test Structures. You cannot move or rename the file. This technique is not recommended; it is recommended that you use the attributes in the design source or through model edit.

- tbdata/TEImsgOverrides2 - This file can be manually created to include additional instance-specific message suppression information.

  The syntax of the file is:

  ```
  MSGID = TSVxxx :
  SUPPRESS_MSG INSTANCE = <block name> ;
  SUPPRESS_MSG  NET = <net name> ;
  ```

  - ❑ *`TSVxxx`* is the message ID to be suppressed. Note that it is the message number with the dash removed; so to suppress `TSV-053` messages, the *`TSVxxx`* will be `TSV053`. Each message ID should be specified only once.
  - ❑ Each `INSTANCE` name is a hierarchical block name that is correlated down to a single logic primitive in the flattened model.
  - ❑ Each `NET` is a hierarchical net name that is correlated to a single flattened model node.
  - ❑ Block and net names may be specified in full proper name form or short name form.

  For example:

  ```
  MSGID = TSV310 :
  SUPPRESS_MSG INSTANCE = clks.abc.pll.core.inst1.l1 ;
  SUPPRESS_MSG INSTANCE = alu.abc.inst1.b0.l1 ;
  SUPPRESS_MSG INSTANCE = Block.f.l.chip.nl.alu.abc.inst1.b0.l2 ;
  MSGID = TSV093 :
  SUPPRESS_MSG NET = alu.abc.inst1.c0.net1 ;
  SUPPRESS_MSG NET = Net.f.l.chip.nl.alu.abc.inst1.c0.net1 ;
  ```

Output

- Additional information in tbdata about the test structures. This information is used by test generation applications.
- Messages describing the location of the detected violation, the type of violation, and possible sources of the violation. The messages are written to:
  - ❑ tbdata in binary form for use by interactive message analysis
  - ❑ stdout and/or the log (unless you specify `reportspecific=no`)

Note: To conserve space in your tbdata directory, you can remove the message analysis data using the command:

```
delete_verify_test_structures_data workdir=<directory> testmode=<name>
```

Analyzing Test Structure Problems in the Design

Design test structure violations indicated by TSV warning messages can affect test coverage and runtime. Severe warnings can cause the patterns to fail at the tester or in simulation. Therefore it is recommended that these should be analyzed before creating tests

This section includes the following:

- Selecting Messages to Analyze - Information about the message severities that helps determine the messages that require analysis
- Analyzing Verify Test Structures Messages - Information for analyzing test structure problems
- Analyzing Broken Scan Chains - A specific scenario and tips for analyzing broken scan chain problems
- Analyzing Contention - A specific scenario and tips for analyzing contention problems
- Analyzing X-Sources - A specific scenario and tips for analyzing sources of unknown values (x-sources) that can cause lower test coverage for all methodologies and can cause bad signatures for signature-based testing such as OPMISR Compression or LBIST.

Selecting Messages to Analyze

Selecting the order for analyzing the messages is partially personal preference. One definite criteria is the severity of the message (see Table 1-2 on page 25). However, you might select to skip messages that you expected and start with unexpected messages. In some cases, you might find a problem that causes multiple messages to be issued and decide to skip the other messages that could be caused by the same problem. If you bring the messages up in the interactive message analysis GUI, they will be sorted in a "recommended analysis order". This order is not strictly based on severity; it tries to include the messages that are easiest to analyze and may cause other problems at the top of the list.

The severity of messages from Verify Test Structures have the following specific meanings:

Table 1-2 TSV Message Severity Description

Message Severity	Description
ERROR	Verify test structures failed due to something other than a design construct and the normal output data is not created. The problem needs to be corrected and then verify_test_structures needs to be rerun.
WARNING [SEVERE]	The design has one or more conditions that will cause the vectors produced by test generation to be suspect. These messages need to be analyzed and corrected to get valid test vectors.
WARNING	The design has some issues that may affect the test coverage, but the vectors produced will be valid.
INFO	The design is fully compliant with the guidelines and Encounter Test will produce valid test data with high coverage.

Analyzing Verify Test Structures Messages

To analyze Verify Test Structures messages using GUI:

1. Set the context for the analysis by clicking the task from the *Task* or *Methodology* panes on the main GUI window. Alternatively, select *Window - Analysis Context* and set the testmode. You can dock the analysis context into the main GUI.
2. Click the *View Messages* icon on the *Messages* tab or on the toolbar. This displays the summary of the messages produced in the most current run of verify_test_structures for this test mode.

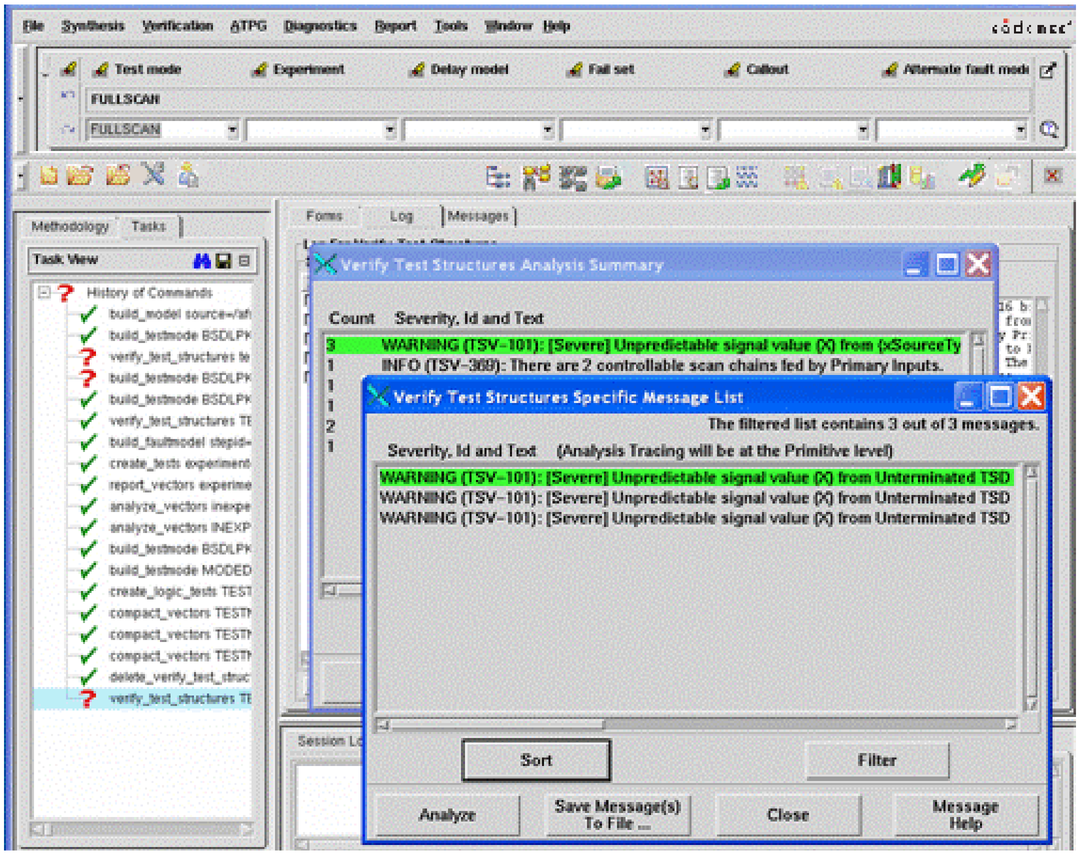
Tip

If the most current invocation of verify_test_structures limited the tests to be run and you want to display all messages for the design, run verify_test_structures with all tests enabled and with the `reruntests` option set to `no` (clicked off on the GUI). This will not rerun the tests, but will bring back all the messages to be displayed.

3. Select a message to analyze from the *Verify Test Structures Message Summary* list and select *View* (or double-click the message). This brings up a list of all the specific messages. If there are more than 50,000 occurrences of the message, only the first 50,000 will be displayed (refer to Customizing Message Analysis Options on page 28 for more information).

4. Select a message to evaluate from the *Verify Test Structures Specific Message* list.

Figure 1-1 Selecting a Message for Analysis



Test mode: FULLSCAN

Verify Test Structures Analysis Summary

Count	Severity, Id and Text
3	WARNING (TSV-101): [Severe] Unpredictable signal value (X) from (xSourceTy
1	INFO (TSV-369): There are 2 controllable scan chains fed by Primary Inputs.

Verify Test Structures Specific Message List

The filtered list contains 3 out of 3 messages.

Severity, Id and Text (Analysis Tracing will be at the Primitive level)

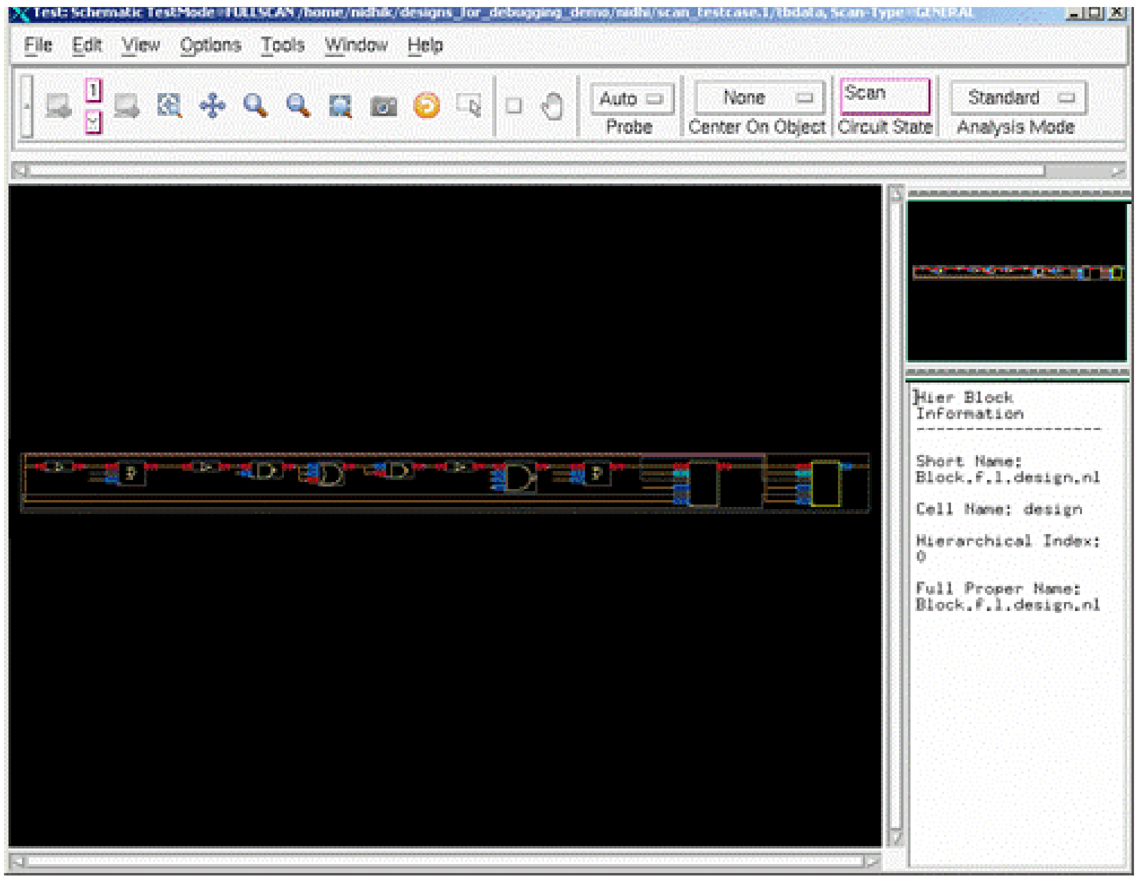
```
WARNING (TSV-101): [Severe] Unpredictable signal value (X) from Unterminated TSD
WARNING (TSV-101): [Severe] Unpredictable signal value (X) from Unterminated TSD
WARNING (TSV-101): [Severe] Unpredictable signal value (X) from Unterminated TSD
```

5. If you do not know what the message means, click the *Message Help* button to access the documentation of the message. Read the User Response section for tips on performing the interactive analysis.

6. If you do know what the message means but need to see the specific occurrence to determine the cause of the problem, select *Analyze*.

■ The schematic displays the portion of the design causing the message, as shown in the following figure.

Figure 1-2 Faulty Part of the Design in the Schematic


Test: Schematic TestMode=FULLSCAN /home/nidhik/designs_for_debugging_demo/nidhi/scan_testcase.1/tbdata, Scan-Type=GENERAL
File
Edit
View
Options
Tools
Window
Help
Auto
Probe
None
Center On Object
Scan
Circuit State
Standard
Analysis Mode

```
Hier Block
Information
--------------------
Short Name:
Block.f.l.design.nl

Cell Name: design

Hierarchical Index:
0

Full Proper Name:
Block.f.l.design.nl
```

Note: The toolbar(s) on the *Schematic* window are customizable (*Options - Customize Toolbar*) so your window may look a little different, but the function is same.

- The *Information* window (the white pane on the lower right side) displays the additional information.

Note: The Information window has many options that you can select or deselect. Refer to Information Window in *Encounter Test: Reference: GUI* for more information.

- The values on the pins are set to the state for analysis. This is often a combination of the design state and some additional stimulus provided by verify_test_structures. For example, if Encounter Test detects a problem by setting the scan state and then turning on all the scan clocks, the design state indication is set as Scan + and all the scan clock PIs are set to the values opposite of their stability value.

- Use the following GUI options in conjunction with the message analysis information to determine the source of the design problem:
  - Trace - Select an object and use the right mouse button to select a trace. Use `b` (back) and `f` (forward) on the keyboard to trace through the logic. Select *Options - Tracing* to customize the tracing options.
  - View elements of the design (block, pin, net, cell) by name - For example, if there is a clocking problem, select *View - Pin*, enter the name, and click *Add to Display* to add the clock PIs to the display.
  - Display Test Function Pin Assignment Data in the Information Window (*Options - Information Window*) - This is useful for clock data that is propagated through the design so you can see the clock PIs that are controlling clocks at a register.
  - Simulate - Use this to see if setting a specific value would resolve the problem. Select a pin and select Simulate and the value(s) from the shortcut menu. Encounter Test simulates these values on top of the existing values in the state. The options for simulation (such as whether to simulate through regs) are available through *Options - Simulate*.
  - Justify - Use this option to see if a specific pin can achieve the required value while the design is in it's current state. Select the pin and then select Justify and the values from the shortcut menu. Select *Options - Justify* to use options for justification.
  - Custom Design State List - Use this interface to specify multiple pins (and values) to be simulated or justified simultaneously. For example, if your analysis shows that two pins that must be at opposite values are causing the problem and you want to know if there is something to prevent that. Select one pin and select *Add to Custom Circuit State List with values 1 and -* from the shortcut menu. Then select the other pin and select the same option with values 0 and -. Then navigate to the *Custom Circuit State List* and click the *Justify* button.

    Note: Values will be simulated or justified within the constraints of the current design state, therefore reset to a known state (*Tools - Reset Circuit State*) before simulating or justifying.

  Note: To ensure that Encounter Test produces valid test data, analyze and resolve all severe errors before proceeding to other applications. Severe errors may cause downstream test generation applications to produce test vectors that fail in simulation or at the tester.

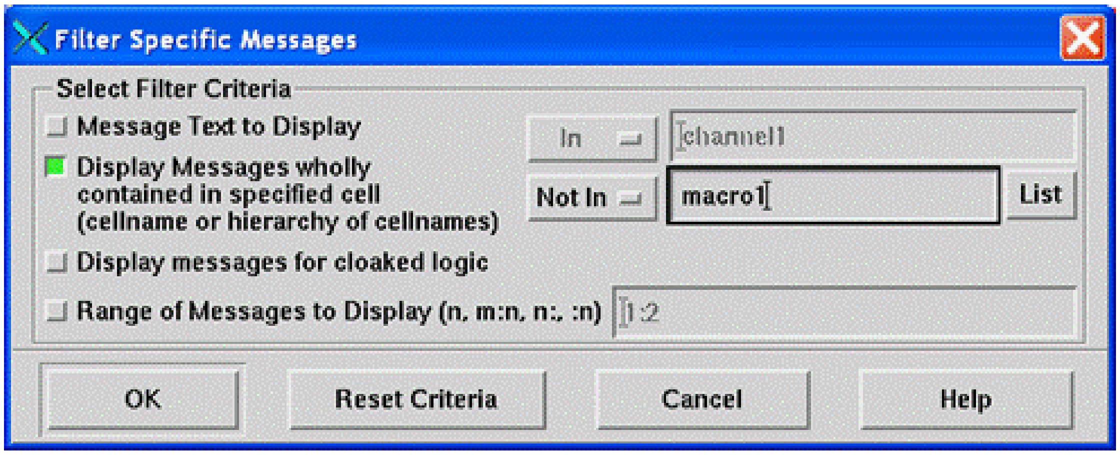
Customizing Message Analysis Options

Large number of occurrences of a specific message can result in longer time for the Specific Message List to display. Moreover, you might find it difficult to navigate to the required messages in the long list. The Filter and Sort capabilities allow you to limit the amount of data to process and organize it to meet your requirements.

The following capabilities help you customize the message analysis options (refer to Message Analysis Windows in the *Encounter Test: Reference: GUI* for more information):

- Use the *Set Filter for Specific Messages* button on the Message Summary window to set the filter before displaying the specific messages. Using this feature you can:
  - ❑ Limit the range of messages to be displayed (for example, 1:20 displays the first 20 messages and 21:40 displays the next 20 messages).
  - ❑ Limit the messages to only those that are not included in a specific cell. For example, if you know the module macro1 in the design has a problem but you want to check if any other part in the design also has a problem, select *Display Messages wholly contained in specified cell,* click *Not In*, enter `macro1` in the field, as shown in the following figure, and click OK. Now when you select *View* for the specific messages, it will display only the messages for parts of the design that are not in marco1.
  - ❑ Limit the messages to only those included in a specific cell. To do so, perform the steps described in the previous point but select *In* instead of *Not In*. Now when you view specific messages, it will only include those that are in macro1.
  - ❑ Limit the messages to only those with specific text. Generally this is used to view/not view all messages that reference a specific pin, net, or block; but, you can also use it to limit the messages to a specific message text. For example, to limit TSV-101 to show only the x-sources from uninitialized RAMs, click the Message Text to *Display*, leave the button as *In*, and fill the entry field with *uninitialized RAM*.
- The features described above are available by clicking the *Filter* button on the *Specific Messages List* window. This will display the same window with your current selections already filled in. For example, if you set the filter initially to display the first 20 messages, click the *Filter* button on the *Specific Messages List* window, change it to display the next 20 messages (21:40), and click *OK*.

Figure 1-3 Filtering Messages



☐ Message Text to Display — In — channel1
☑ Display Messages wholly contained in specified cell (cellname or hierarchy of cellnames) — Not In — macro1
☐ Display messages for cloaked logic
☐ Range of Messages to Display (n, m:n, n:, :n) — 1:2

- Use the *Sort* button on the *Message Summary* window to change the order of the messages on the window. Encounter Test recommends the analysis order that is best suited for the analysis process. For example, a WARNING message about a clocking problem may be easier to analyze and may actually be the cause of a broken scan chain, therefore, Encounter Test keeps the clock message higher in the list.
- Use the *Sort* button on the *Specific Messages List* window to sort by the actual text in the message or by the area of the design where the problem was detected.

Analyzing Broken Scan Chains

Broken scan chains can result from any of the following:

- A design error in the netlist
- An error in pin definitions in the assignfile
- An error in the cell library models
- Error in the initialization sequence in the `seqdef` file

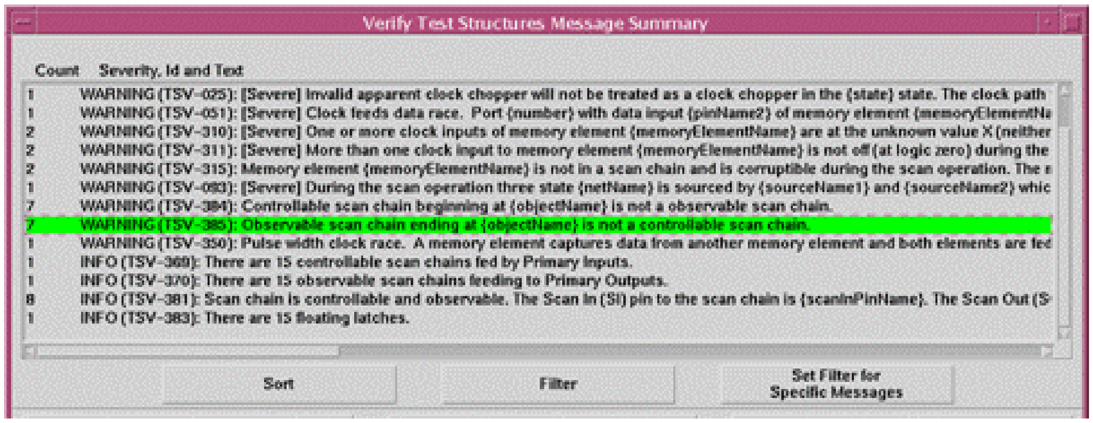
Encounter Test traces scan chains in a design both forwards from scan-in and backwards from scan-out and produces the following messages for broken scan chains:

WARNING (TSV-384)

WARNING (TSV-385)

Figure 1-4 on page 31 displays some sample TSV-385 messages:

Figure 1-4 TSV Messages for Broken Scan Chains



```
Count   Severity, Id and Text
1       WARNING (TSV-025): [Severe] Invalid apparent clock chopper will not be treated as a clock chopper in the {state} state. The clock path
1       WARNING (TSV-051): [Severe] Clock feeds data race.  Port {number} with data input {pinName2} of memory element {memoryElementNa
2       WARNING (TSV-310): [Severe] One or more clock inputs of memory element {memoryElementName} are at the unknown value X (neither
2       WARNING (TSV-311): [Severe] More than one clock input to memory element {memoryElementName} is not off (at logic zero) during the
2       WARNING (TSV-315): Memory element {memoryElementName} is not in a scan chain and is corruptible during the scan operation. The r
1       WARNING (TSV-093): [Severe] During the scan operation three state {netName} is sourced by {sourceName1} and {sourceName2} whic
7       WARNING (TSV-384): Controllable scan chain beginning at {objectName} is not a observable scan chain.
7       WARNING (TSV-385): Observable scan chain ending at {objectName} is not a controllable scan chain.
1       WARNING (TSV-350): Pulse width clock race.  A memory element captures data from another memory element and both elements are fed
1       INFO (TSV-369): There are 15 controllable scan chains fed by Primary Inputs.
1       INFO (TSV-370): There are 15 observable scan chains feeding to Primary Outputs.
8       INFO (TSV-381): Scan chain is controllable and observable. The Scan In (SI) pin to the scan chain is {scanInPinName}. The Scan Out (S
1       INFO (TSV-383): There are 15 floating latches.
```

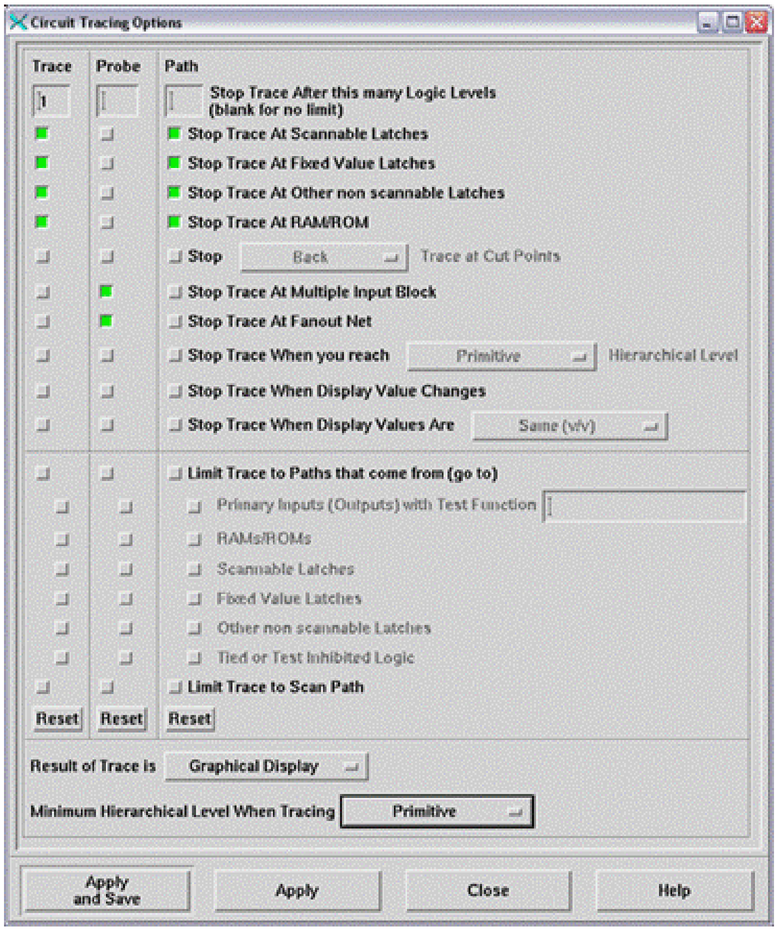
Analyze the specific TSV message to start debugging the scan chains. Refer to Analyzing Verify Test Structures Messages on page 25 for details.

The schematic viewer displays the last correct instance of the design logic before the scan chain break. The instance cell type and name are displayed in *Information Window*.

Scan chains can be traced at primitive, library cell, or macro level. Perform the following steps to customize the tracing level for the design logic:

1. In the *Circuit Display* window, select *Options* - *Circuit Tracing*. The *Circuit Tracing Options* window is displayed:

Figure 1-5 Circuit Tracing Options Window



Trace	Probe	Path
1		Stop Trace After this many Logic Levels (blank for no limit)
✓	☐	✓ Stop Trace At Scannable Latches
✓	☐	✓ Stop Trace At Fixed Value Latches
✓	☐	✓ Stop Trace At Other non scannable Latches
✓	☐	✓ Stop Trace At RAM/ROM
☐	☐	☐ Stop Back Trace at Cut Points
☐	✓	☐ Stop Trace At Multiple Input Block
☐	✓	☐ Stop Trace At Fanout Net
☐	☐	☐ Stop Trace When you reach Primitive Hierarchical Level
☐	☐	☐ Stop Trace When Display Value Changes
☐	☐	☐ Stop Trace When Display Values Are Same (v/v)
☐	☐	☐ Limit Trace to Paths that come from (go to)
☐	☐	☐ Primary Inputs (Outputs) with Test Function
☐	☐	☐ RAMs/ROMs
☐	☐	☐ Scannable Latches
☐	☐	☐ Fixed Value Latches
☐	☐	☐ Other non scannable Latches
☐	☐	☐ Tied or Test Inhibited Logic
☐	☐	☐ Limit Trace to Scan Path

Result of Trace is: Graphical Display

Minimum Hierarchical Level When Tracing: Primitive

2. Set a value for the *Minimum Hierarchical Level When Tracing* option and click either of the following:

   ❑ *Apply* - Apply the changes for use while the window remains open

   ❑ *Apply and Save* - To save the changes for all the future sessions. This is the recommended usage for this scenario.

Check the displayed instance for the following:

- If the instance is a top-level pin, it implies that the flops were not traced correctly.
- If the instance is a flop, it implies that this is the last working flop.

Complete the following steps to move up the hierarchy to debug the instance at the library level:

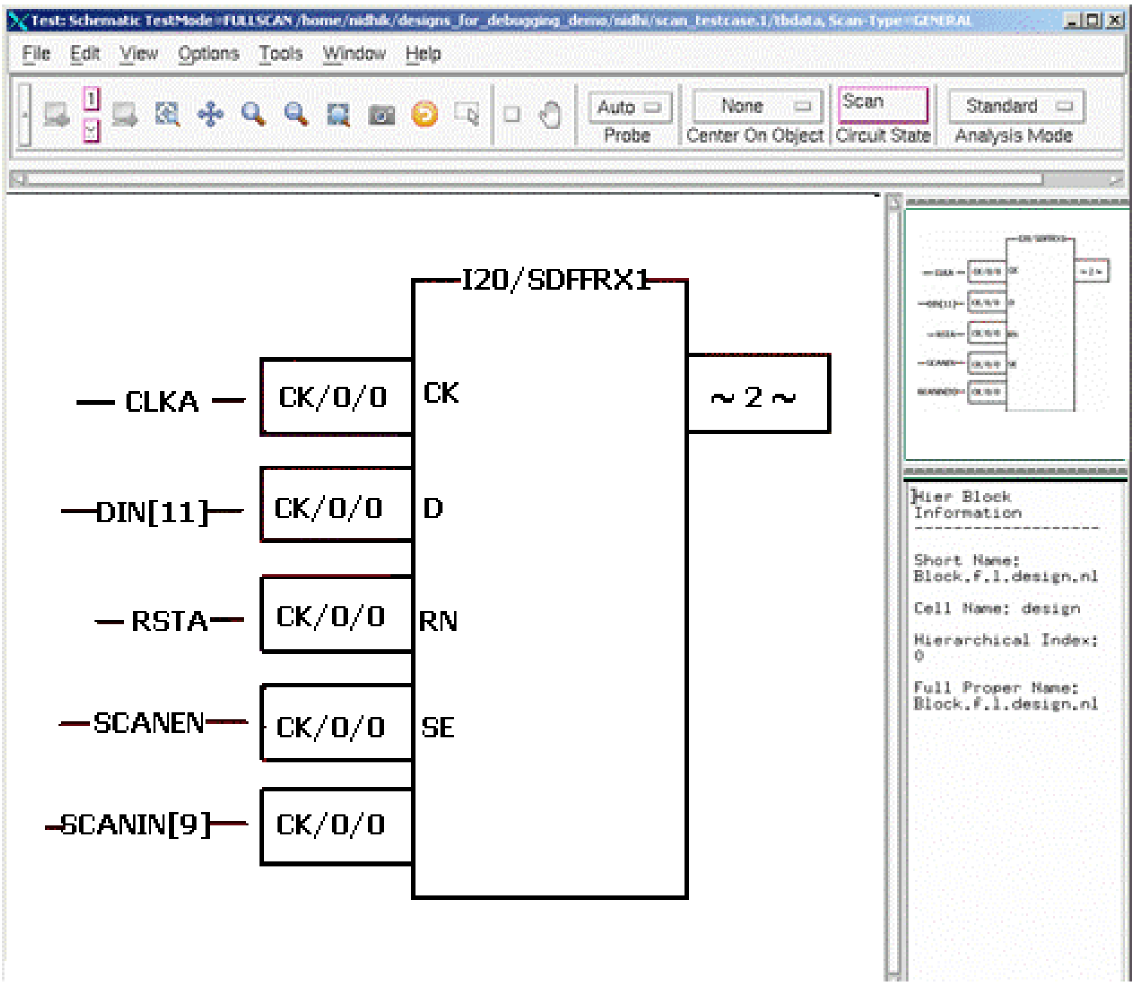
1. Select the block in the design diagram.

2. Right click and select *Implode*.

Note: This may need to be done multiple times until the library level cell is found.

Figure 1-6 on page 33 displays values for a sample scan flop that is functioning correctly:

Figure 1-6 Correctly Functioning Scan Flop



Test: Schematic TestMode=FULLSCAN /home/nidhik/designs_for_debugging_demo/nidhi/scan_testcase.1/tbdata, Scan-Type=GENERAL

Probe: Auto
Center On Object: None
Circuit State: Scan
Analysis Mode: Standard

```
Hier Block
Information
--------------------

Short Name:
Block.f.1.design.nl

Cell Name: design

Hierarchical Index:
0

Full Proper Name:
Block.f.1.design.nl
```

In the preceding figure:

- The clock pin shows a defined off state: 0 or 1 is acceptable, and X is an error
- The asynchronous set or reset pin shows a valid off state: 0 in this case, X would be an error
- The scan-enable pin is active: 0 in this case

- The data-in and scan-in pins are undefined: This is normal during scan chain trace and is not an error

Analyze the initial instance for the following possible errors:

- Check the logic state of the scan-enable pin. If the scan-in pin shows:
  - `x` (lower case) then the input is floating or undriven. The unknown value on such pins is typically a result of a TIEX block in the design.

  Note: `x` (lower case) and `X` (uppercase) represent different logic values only in the Schematic Viewer. While `x` represents a foating or undriven pin, `X` represents unknown or uninitialized pins.

  - `0`, `1`, `+`, or `-` then the scan-in pin has been forced to a fixed logic value

  For example, Figure 1-7 on page 34 represents a scan enable at incorrect state:

Figure 1-7 Scan Enable at Incorrect State

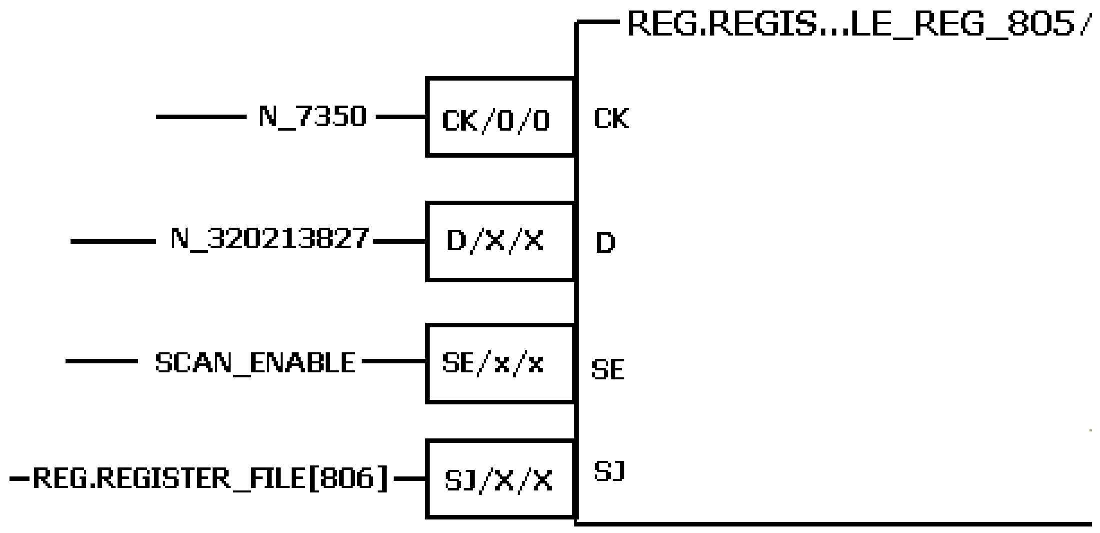
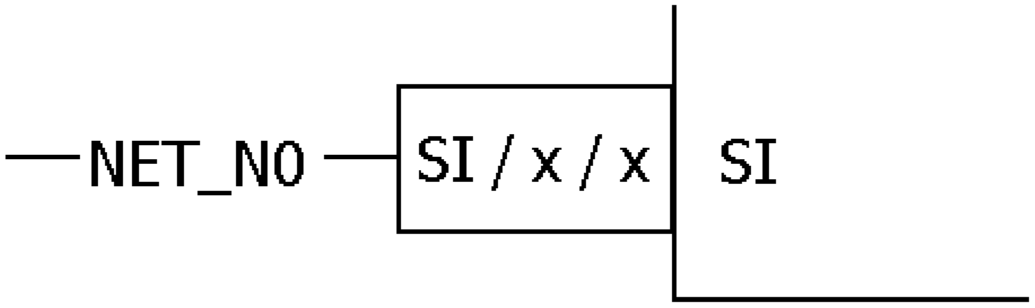


Figure 1-8 on page 35 displays scan-in at `x` state (lower case), and therefore represents an unconnected state:

Figure 1-8 Unconnected Scan-in



Perform the following steps to trace backwards from the scan-in pin of the initial instance to find the faulty instance:

1. Select the scan-in pin.
2. Right click and select *Trace Backward* from the shortcut menu.

Tip

Alternatively use `b` on the keyboard as a shortcut to select the *Trace Backward* option.

Note: If it is not possible to trace backward, that implies that there is no connection.

By default, Encounter Test traces one level of hierarchy at a time and stops at latches and flip flops. However, the GUI can be set up to trace through multiple levels at a time. Complete the following steps to set the number of levels to skip around when tracing forward and backward:

1. In the *Schematic* window, select *Options* - *Circuit Tracing*.
2. In the upper section of the *Circuit Tracing Options* window, set the number of levels to trace through and when to stop tracing.

After tracing to the faulty instance, implode to the library level and check each input pin for the following by placing the cursor over it:

1. Clock pin

   Whether the pin has a defined off state `0` or `1. X` indicates a corrupt clock.

   Whether the clock is defined as `EC` or `SC`

2. Asynchronous set/reset pin

   Whether the pin is tied to the correct inactive level

      `+` or `1` for active low

      `-` or `0` for active high

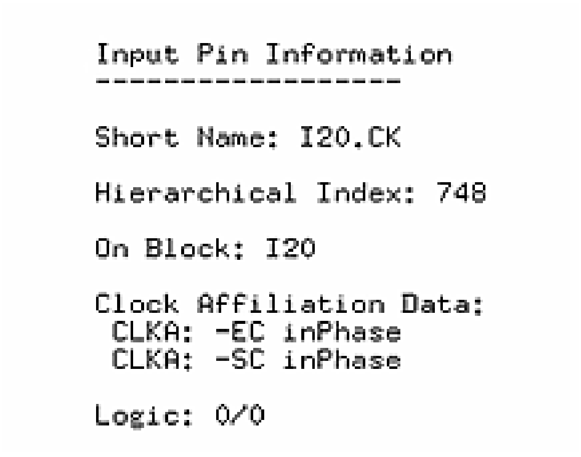
   Whether the pin is defined as an `SC` type clock with the correct off-state

The correct values on pins are:

- Correct scan clock: The clock has a defined off-state `0` (`1` is OK for a negative edge triggered clock).

  The clock is an `EC` type, therefore, will be pulsed during scan, as displayed in Figure 1-9 on page 36:

Figure 1-9 Correct Scan Clock

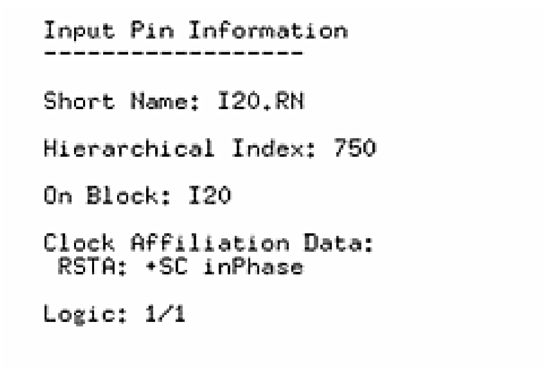


```
Input Pin Information
-----------------

Short Name: I20.CK

Hierarchical Index: 748

On Block: I20

Clock Affiliation Data:
 CLKA: -EC inPhase
 CLKA: -SC inPhase

Logic: 0/0
```

- Correct asynchronous reset: In Figure 1-10 on page 37, the clock has a defined off-state `1`, which is correct for an active low reset.

  The clock is an `SC` type so it will not be pulsed during scan. Alternatively, the clock may be set at a fixed logic value.

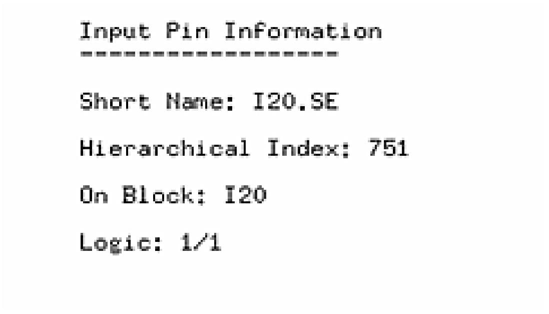
Figure 1-10 Correct Asynchronous Reset



```
Input Pin Information
------------------

Short Name: I20.RN

Hierarchical Index: 750

On Block: I20

Clock Affiliation Data:
 RSTA: +SC inPhase

Logic: 1/1
```

- Correct scan enable: The scan enable has a defined logic `1`, as displayed in the following figure:

Figure 1-11 Correct Scan-enable

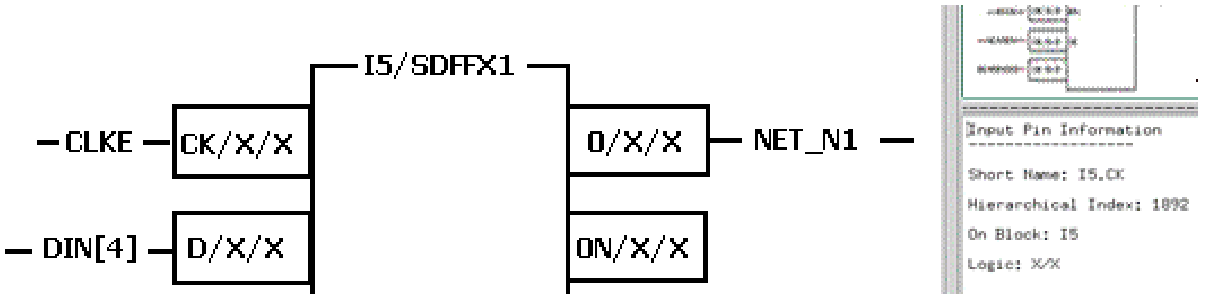


```
Input Pin Information
------------------

Short Name: I20.SE

Hierarchical Index: 751

On Block: I20

Logic: 1/1
```

Some examples of possible errors and how they would appear in the GUI are:

- No scan clock on the flop: In Figure 1-12 on page 38, the logic state of the CK pin is `X`, and no clock is defined:

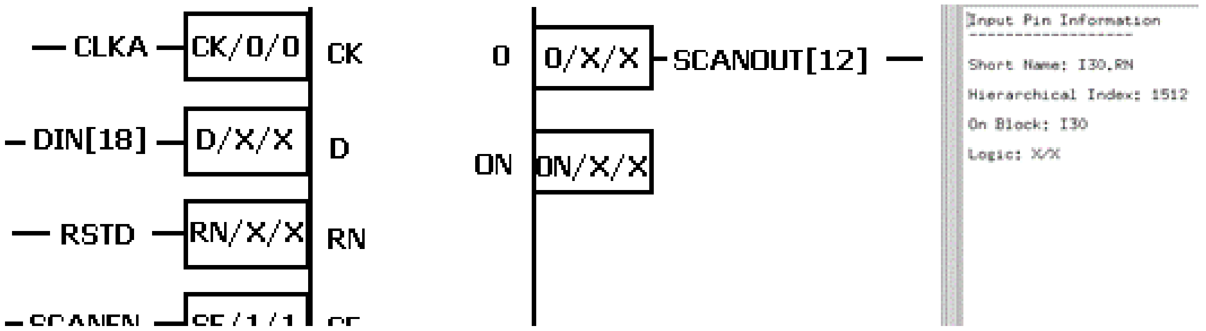
Figure 1-12 No Clock Defined



```
Input Pin Information
---------------------

Short Name: I5.CK

Hierarchical Index: 1892

On Block: I5

Logic: X/X
```

- Uncontrolled reset: In Figure 1-13 on page 38, the reset pin is at a logic `X` state so the flop will be corrupted.

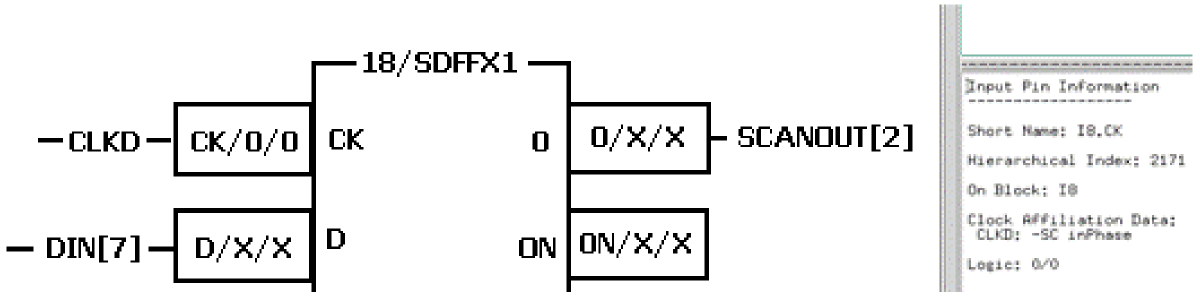
Figure 1-13 Uncontrolled Reset



```
Input Pin Information
---------------------

Short Name: I30.RN

Hierarchical Index: 1512

On Block: I30

Logic: X/X
```

- Incorrect clock type: In Figure 1-14 on page 38 the clock is of type `SC` so will not be clocked during the scan sequence.

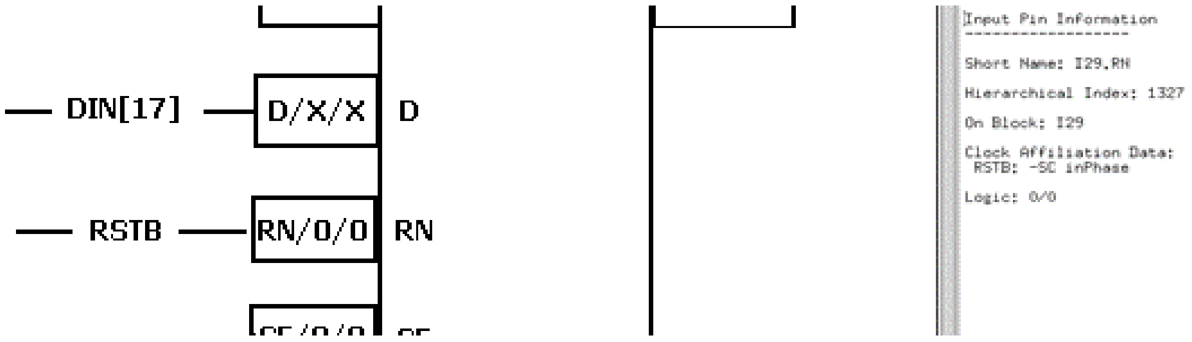
Figure 1-14 Incorrect Clock Type



```
Input Pin Information
---------------------

Short Name: I8.CK

Hierarchical Index: 2171

On Block: I8

Clock Affiliation Data:
 CLKD: -SC inPhase

Logic: 0/0
```

- Incorrect reset clock polarity: In Figure 1-15 on page 39 the asynchronous reset clock has an off-state of `0` so the flop will be reset when the clocks are held off during scan.

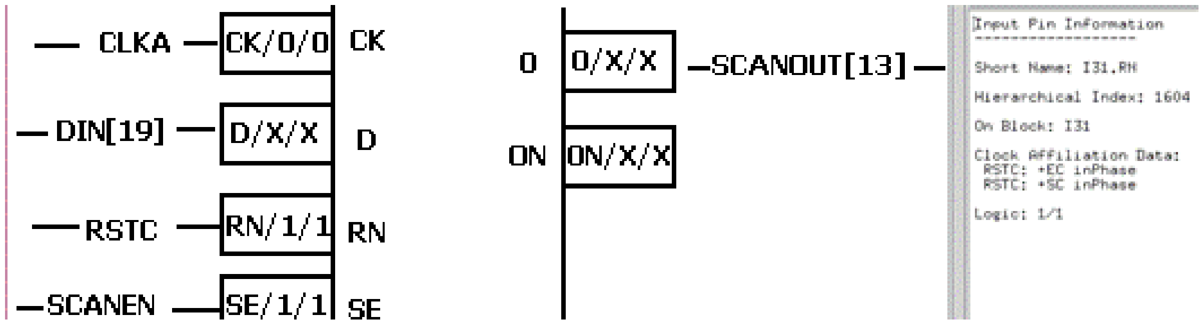
Figure 1-15 Incorrect Reset Clock Polarity



```
Input Pin Information
--------------------

Short Name: I29.RN

Hierarchical Index: 1327

On Block: I29

Clock Affiliation Data:
 RSTB: -SC inPhase

Logic: 0/0
```

- Incorrect clock type on asynchronous reset: In Figure 1-16 on page 39 the reset has an `EC` type clock so will be pulsed during scan. This will corrupt the scan chain operation. Resets and sets should be `SC`.

Figure 1-16 Incorrect Clock Type on Asynchronous Reset



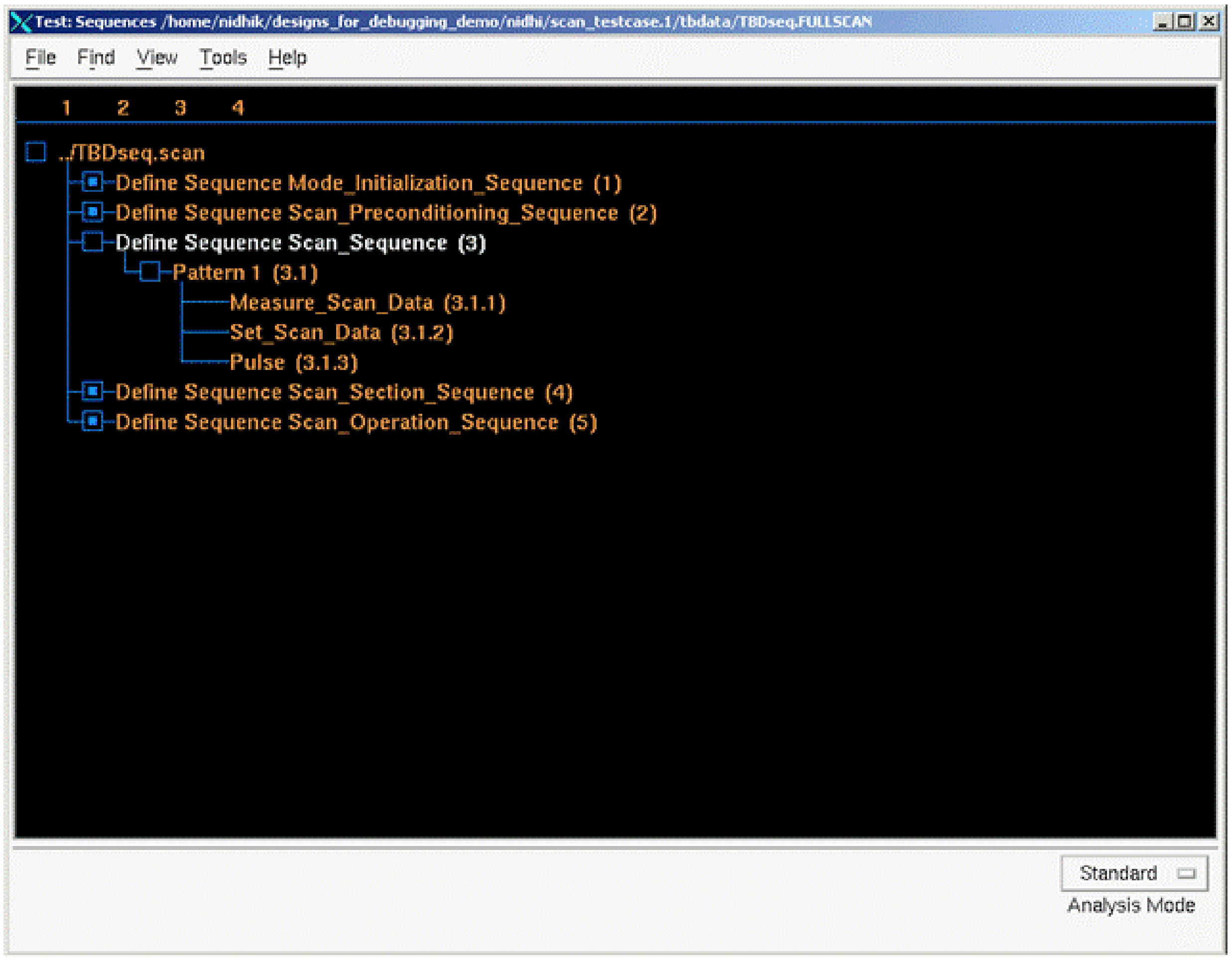
```
Input Pin Information
--------------------

Short Name: I31.RN

Hierarchical Index: 1604

On Block: I31

Clock Affiliation Data:
 RSTC: +EC inPhase
 RSTC: +SC inPhase

Logic: 1/1
```

If all of the previously-mentioned checks pass, analyze the simulation of the scan sequence by doing the following:

1. In the main GUI window, click the *View Sequences* icon*.*
2. Expand *Click blue boxes node* and select *Define Sequence Scan_Sequence.*
3. Use the right mouse button and select *View Circuit Values.*

The *Simulating Sequences* window displays and when the simulation is complete, the pins on the schematic are updated with the scan sequence values.

Figure 1-17 Schematic with Scan Sequence

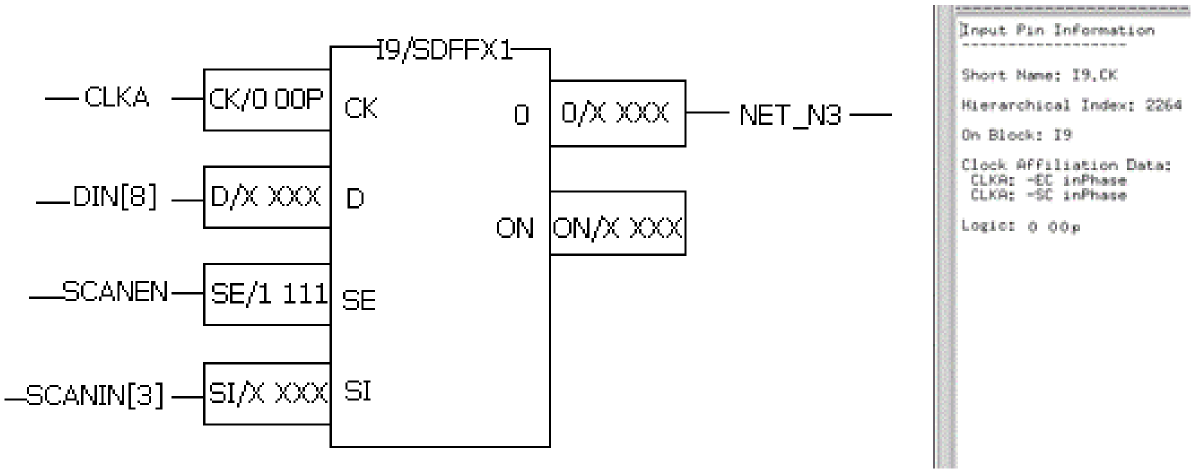

Test: Sequences /home/nidhik/designs_for_debugging_demo/nidhi/scan_testcase.1/tbdata/TBDseq.FULLSCAN
File Find View Tools Help
1 2 3 4
Standard
Analysis Mode

```
../TBDseq.scan
  Define Sequence Mode_Initialization_Sequence (1)
  Define Sequence Scan_Preconditioning_Sequence (2)
  Define Sequence Scan_Sequence (3)
    Pattern 1 (3.1)
      Measure_Scan_Data (3.1.1)
      Set_Scan_Data (3.1.2)
      Pulse (3.1.3)
  Define Sequence Scan_Section_Sequence (4)
  Define Sequence Scan_Operation_Sequence (5)
```

Verify if the scan clock is correct. For example, in Figure 1-18 on page 41, a clean pulse `p` is displayed for the clock pin. The clock pin CK shows that the stability value is 0 (this is the value at the end of test mode initialization and is generally based on the test function pin assignment values except in case of user-defined mode initialization sequence). Then it shows the value for each event in the sequence. For this sequence, the CK pin was at:

- 0 during the first event (Measure_Scan_Data)
- 0 during the second event (Set_Scan_Data)
- pulsed p during the third event (Pulse)

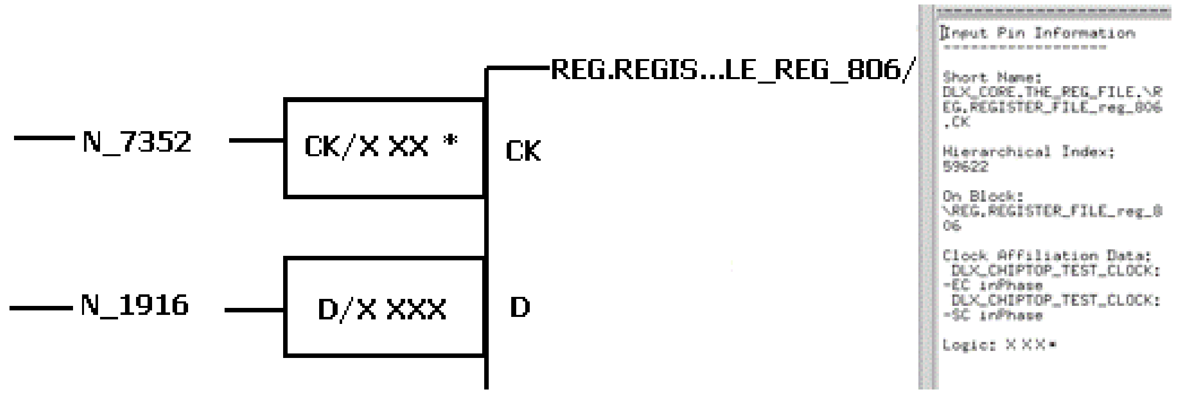
Figure 1-18 Clean Clock Pulse



```
Input Pin Information
---------------------

Short Name: I9.CK

Hierarchical Index: 2264

On Block: I9

Clock Affiliation Data:
 CLKA: -EC inPhase
 CLKA: -SC inPhase

Logic: 0 00p
```

However, in Figure 1-19 on page 41, the clock shows a * state, which indicates an unknown pulse state. The starting state is unknown X when the clock is pulsed and the next pulse is unknown, therefore, it is an unknown pulse state.

Figure 1-19 Unknown Clock Pulse State



```
Input Pin Information
---------------------

Short Name:
DLX_CORE.THE_REG_FILE.\R
EG.REGISTER_FILE_reg_806
.CK

Hierarchical Index:
59622

On Block:
\REG.REGISTER_FILE_reg_8
06

Clock Affiliation Data:
 DLX_CHIPTOP_TEST_CLOCK:
-EC inPhase
 DLX_CHIPTOP_TEST_CLOCK:
-SC inPhase

Logic: X XX*
```

Caution

Known Problem: There are cases where the simulation of the scan sequence does not match the scan state. This is because of a difference in the initial design state and the simulation options used during build_testmode and GUI View Circuit Values. If the final state of the design from View Circuit Values does not match the state when you use Tools - Reset Circuit State - Scan, then do not rely on the View Circuit

Values simulation. This is typically a problem for designs with Fixed Value regs, cutpoint/PPIs, or clock choppers, but may also occur in other cases.

Analyzing Contention Problems

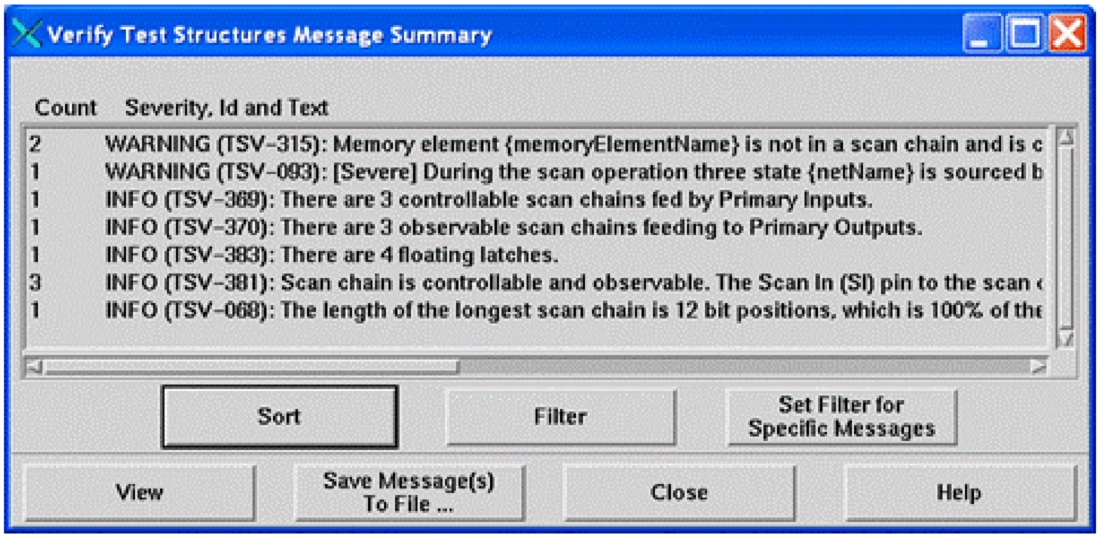
TSV produces the following messages for contention violations:

WARNING TSV-093 [Severe]

WARNING TSV-193

Figure 1-20 on page 42 displays the message summary containing contention messages:

Figure 1-20 TSV Messages for Contention

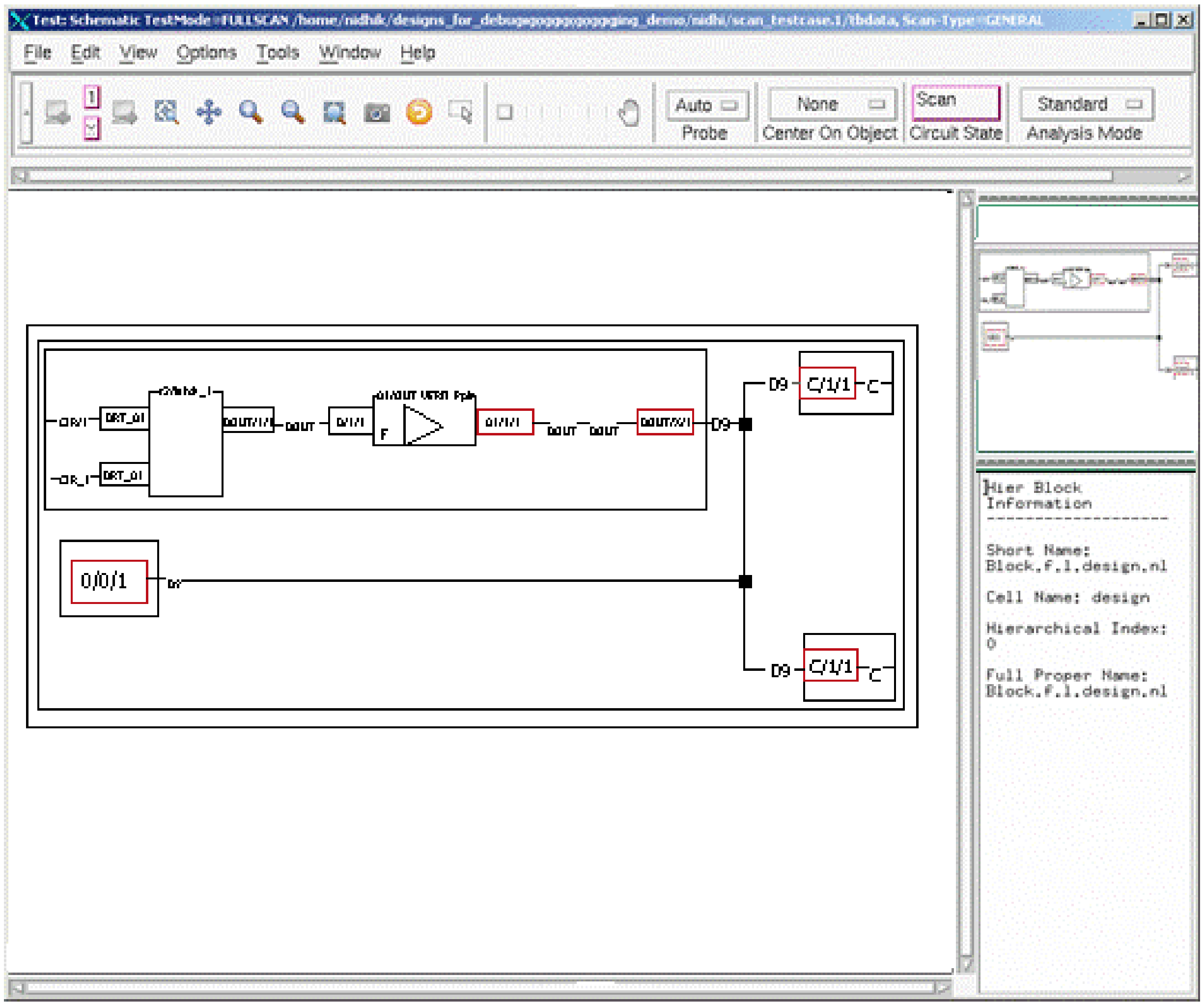

Verify Test Structures Message Summary
Sort
Filter
Set Filter for Specific Messages
View
Save Message(s) To File ...
Close
Help

```
Count   Severity, Id and Text
2       WARNING (TSV-315): Memory element {memoryElementName} is not in a scan chain and is c
1       WARNING (TSV-093): [Severe] During the scan operation three state {netName} is sourced b
1       INFO (TSV-369): There are 3 controllable scan chains fed by Primary Inputs.
1       INFO (TSV-370): There are 3 observable scan chains feeding to Primary Outputs.
1       INFO (TSV-383): There are 4 floating latches.
3       INFO (TSV-381): Scan chain is controllable and observable. The Scan In (SI) pin to the scan c
1       INFO (TSV-068): The length of the longest scan chain is 12 bit positions, which is 100% of the
```

Select the TSV-093 message and click *View* to get the specific instance of the TSV message. Then click *Analyze* to display the contention problem on the schematic.

Encounter Test displays the schematic with problem areas highlighted. For example, Figure 1-21 on page 43 displays a sample design diagram that contains some problem nets.

Figure 1-21 Design Diagram with Contention Problems


Test: Schematic TestMode=FULLSCAN /home/nidhik/designs_for_debugggggggggging_demo/nidhi/scan_testcase.1/tbdata, Scan-Type=GENERAL
File
Edit
View
Options
Tools
Window
Help
Auto
Probe
None
Center On Object
Scan
Circuit State
Standard
Analysis Mode
D9
C/1/1
C
0/0/1
D9
C/1/1
C

```
Hier Block
Information
--------------------

Short Name:
Block.f.1.design.nl

Cell Name: design

Hierarchical Index:
0

Full Proper Name:
Block.f.1.design.nl
```

Verify that the design includes the correct net names and logic.

Analyzing Race Conditions with MEG or Three-State Contention

When analyzing feedback X-State race conditions with MEG, or Three-State Contention, additional information is provided. The additional information is the stimulus points required to establish design values that are displayed. The clock off state is not included in this list. This information is written to stdout.

Analyzing X-Sources

X-sources are sources of unknown values. These may be because of a blackbox, uninitialized memory (RAM/ROM), Primary Inputs that are identified as uncontacted at the tester, unconnected pins, or floating regs. X-sources cause problem in signature-based testing techniques (OPMISR, OPMISR+, LBIST, WRPT) if the X can get into the signature register. For other testing techniques, an X-source may cause lower test coverage.

The potential TSV messages for x-sources are :

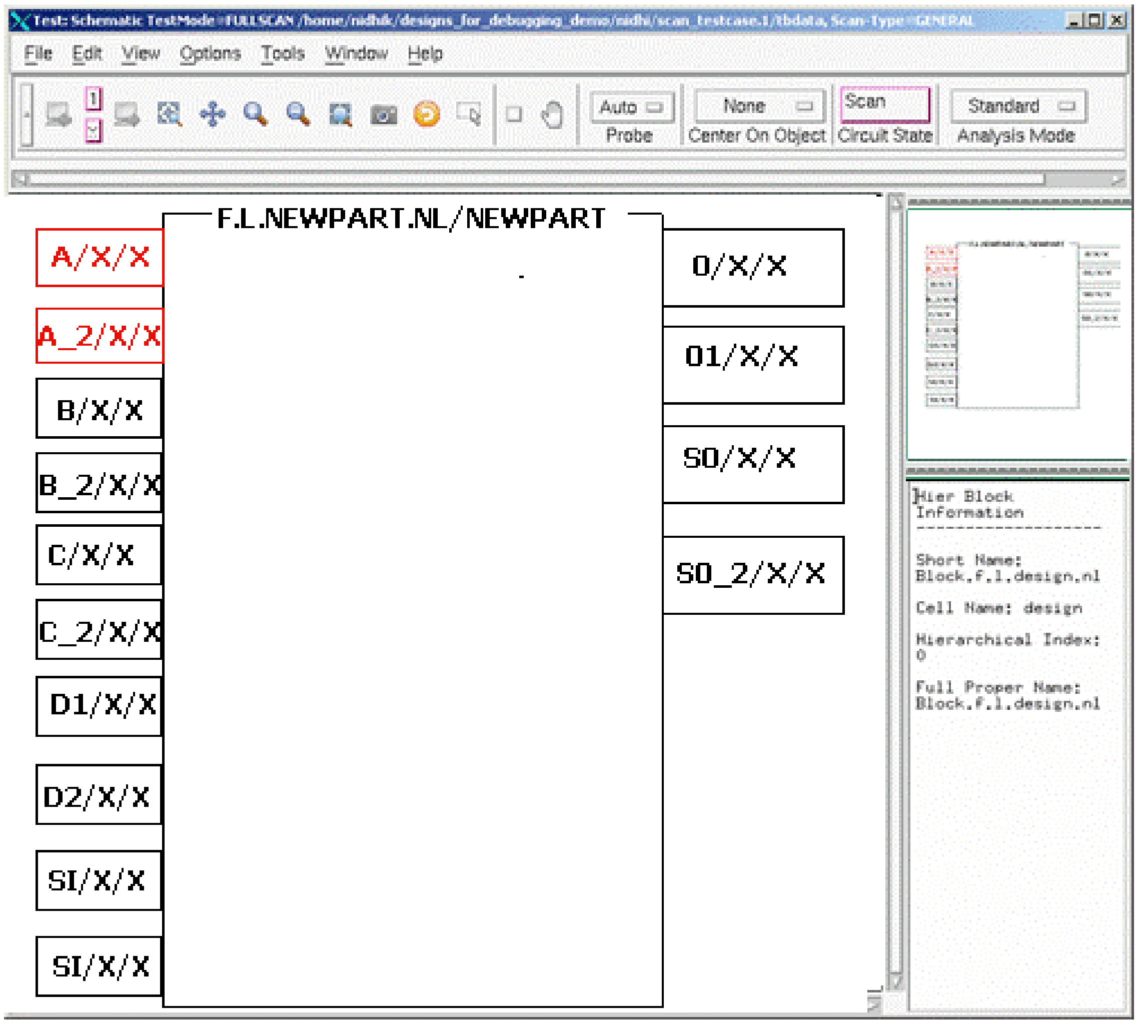
WARNING (TSV-101)

WARNING (TSV-102)

These TSV messages report any X-source that is observable (can propagate to a memory element or Primary Output).

In the schematic, inactive logic is highlighted with a different color. For example, the following figure displays an inactive logic:

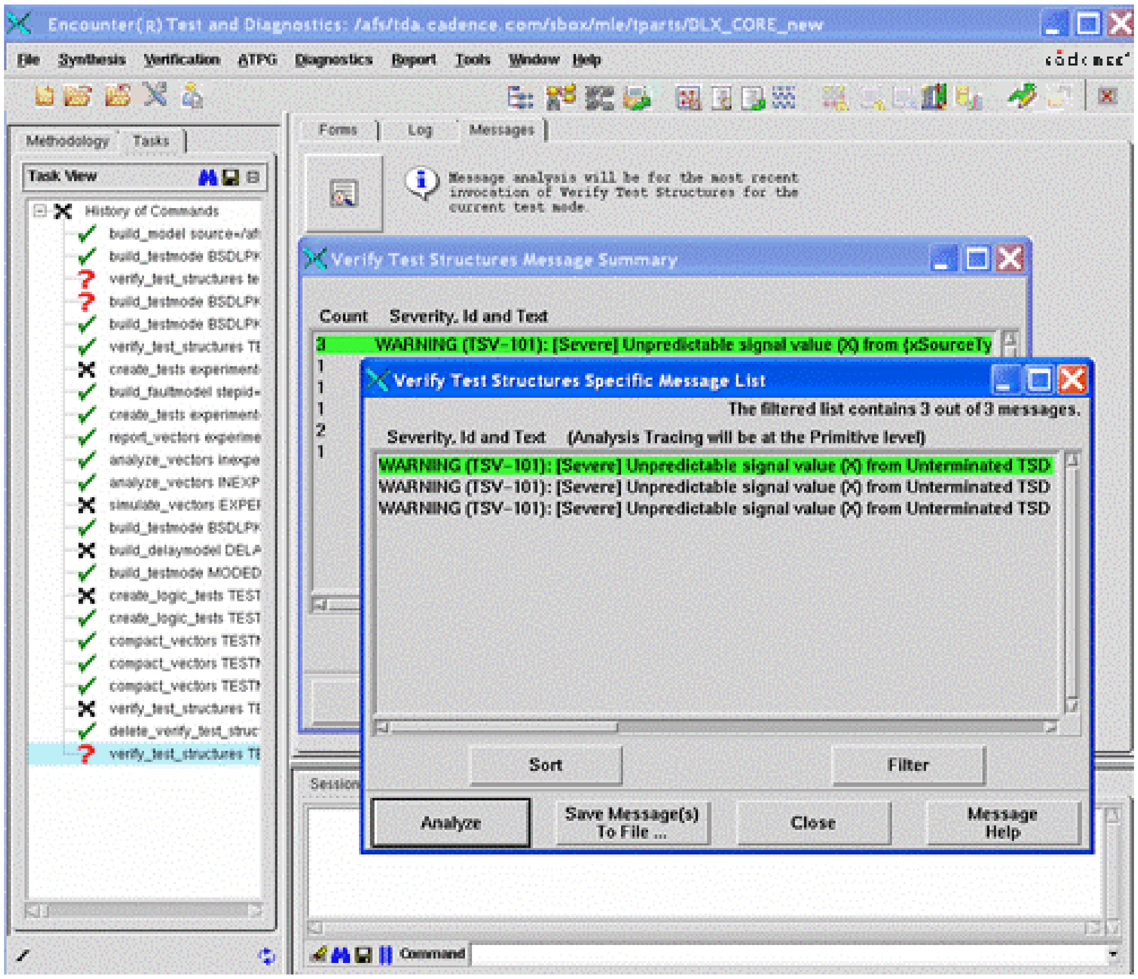
Figure 1-22 Design Diagram with Inactive Logic



```
Hier Block
Information
--------------------

Short Name:
Block.f.l.design.nl

Cell Name: design

Hierarchical Index:
0

Full Proper Name:
Block.f.l.design.nl
```

To start the analysis process, repeat the above-mentioned steps given analyze TSV messages. The *Messages* window displays the list of messages, as shown in the following figure.

Figure 1-23 X-Source Messages in the Message Window


Encounter(R) Test and Diagnostics: /afs/tda.cadence.com/sbox/mle/tparts/DLX_CORE_new
File Synthesis Verification ATPG Diagnostics Report Tools Window Help
Methodology Tasks
Task View
History of Commands
Forms Log Messages
Session
Command
Sort
Filter
Analyze
Save Message(s) To File ...
Close
Message Help

Message analysis will be for the most recent invocation of Verify Test Structures for the current test mode.

Verify Test Structures Message Summary

Count	Severity, Id and Text
3	WARNING (TSV-101): [Severe] Unpredictable signal value (X) from (xSourceTy
1	
1	
1	
2	
1	

Verify Test Structures Specific Message List

The filtered list contains 3 out of 3 messages.

Severity, Id and Text (Analysis Tracing will be at the Primitive level)

```
WARNING (TSV-101): [Severe] Unpredictable signal value (X) from Unterminated TSD
WARNING (TSV-101): [Severe] Unpredictable signal value (X) from Unterminated TSD
WARNING (TSV-101): [Severe] Unpredictable signal value (X) from Unterminated TSD
```

Modify the logic to block all paths from the source of the unknown value to an observable point.

Clock Affiliation

These are pins that may be controlled by a clock signal. This information is used to verify a logic design. For example, it can be used to determine whether a clock signal feeds a data input of a latch/RAM, or whether clock signals interact in a proper manner.

To display Clock Affiliation data:

1. Select *Options* from the View Schematic Window
2. Select *Information Window*.

3. View the *Items to Display* list to determine the presence of the *Clock Affiliation (Carries Clock)* option. If not present, add it by selecting it from the *Items to Choose From* list and clicking the left-pointing arrow to add it to the *Items to Display* list.
4. Select *OK* to activate the option.

The Information Window contains symbols in the *Clock Affiliation* field that denotes the phase and type of chopped clocks. The possible phases are *inPhase* and *outOfPhase*.

The types are:

- *[LE]* - a leading edge chopped clock
- *[TE]* - a trailing edge chopped clock
- *[LE-TE]* - a leading edge chopped clock fed through a trailing edge chopper
- *[TE-TE]* - a clock fed through multiple trailing edge choppers
- *[LE-TE-TE]* - a leading edge chopped clock fed through two (or more) trailing edge choppers
- *[Invalid]* - any illegally chopped clock signal

For example, an A-scan clock driven by a leading edge clock chopper that was in the same phase as the clock_PI (MY_ACLOCK) would have this clock affiliation data displayed in the Information window:

```
MY_ACLOCK - AC [LE] inPhase
```

Selectable Tests

This section describes the TSV checks for the Generalized Scan Design (GSD) Style. GSD is a broad scan design philosophy that supports both edge-sensitive and level-sensitive scan elements such as:

- Mux-scan single-clock multiplexed flip-flop designs (Scan Path)
- Two-phase clocking schemes
- Mixes of these scan styles on the same part, including LSSD

Analyze Three-state Drivers for Contention

The following describes the test guidelines, verification process, intent, and restrictions for this check.

Test Guideline TBT.8 - Three State Driver Contention

The enable inputs of Three-State Drivers (TSDs) and the gate inputs of transistor primitives (NFET or PFET) that drive the same net must be controlled so that no two sources can be actively driving conflicting values during the application of the test data, unless the sources are burnout proof and no product damage can occur.

Verification Process

This check verifies that multiple Three-State Drivers (TSDs) which drive the same net cannot be simultaneously enabled with different data values. If the contention can occur, TSV further verifies that shifting a test pattern into the scan chains does not cause three-state bus contention.

Intent

This guideline ensures that a product will not be damaged during test.

A secondary intent is to ensure that any logic model constraints are not violated in this testmode. Logic model constraints, identified with model attributes or a build_model constraints file, are modeled with TSDs that are simultaneously enabled if the constraint is violated. See "Applying Logic Model Constraints" in the *Encounter Test: Guide 1: Models*.

If the contention can occur while shifting a pattern in the scan chains (in scan state), the test generation processes cannot protect the tests from potential burnout. Therefore, this is always a Severe violation.

If the contention cannot occur in scan state, the stored pattern test generation/fault simulation process can protect the test data from burnout. By default, it removes any patterns with three-state contention. However, the signature-based testing process cannot remove patterns and therefore, cannot protect the test data from potential burnout. Therefore, this is only a severe violation for test modes with signature-based test methodology.

See "Scan States" in the *Encounter Test: Guide 2: Testmodes* for additional information.

Restrictions

1. Encounter Test does not support a method for identifying TSDs as burnout proof. All TSDs are tested for this error condition.
2. There is no method to tell `verify_test_structures` the type of contention required in the report; all types of contention are reported. However, when creating tests, you can select the type of contention to be prevented or removed.

Outcome

The following messages may be produced by this test:

TSV-093	TSV-094	TSV-193
TSV-194		

Analyze Feedback Loops and Keeper Devices

The following describes the test guidelines, verification process, and intent for this check.

Test Guideline TG.2 - No Feedback Loops

No design signal outside of the recognized memory elements (flops, latches, RAMs, and three-state keeper devices) may feed back on itself in such a manner as to allow it to latch a signal value or to produce oscillating design values. In addition, in both the stability state and in each of the states defined by placing a single clock opposite its stability state value, there must exist no global feedback through flop, latch, or RAM ports whose clock input is on.

Verification Process

Traces through the design to identify feedbacks. Analyzes the logic to determine if the feedback is broken with some gating condition. If mutually exclusive gating prevents the feedback from occurring, no error is generated.

Intent

Ensure efficient and accurate automatic test pattern generation on the design.

Asynchronous feedback loops can latch up, oscillate, or produce other complicated sequential behavior. It may be impossible to correctly predict this behavior without understanding the detailed timings, and the test generator may not be able to handle the sequential behavior efficiently. As a result, test coverage will suffer and there is also a possibility that invalid test data could be produced.

Outcome

The following messages may be produced by this test:

TSV-001	TSV-002	TSV-030
TSV-031	TSV-032	TSV-033
TSV-034	TSV-035	

Analyze Test Clocks Control of Memory Elements

The following describes the test guidelines, verification process, and intent for this check.

Test Guideline TG.3 - Stable State

The clock inputs of all memory elements must be controlled from clock primary inputs that have a defined stability state value. This ensures the design state is maintained regardless of the values at non-clock and non-test inhibit PIs.

Test Guideline TG.7 - Clock Requirements

It must be possible to identify a set of clock pins which control the clock inputs to flops, latches and RAMs to a known value (0 or 1). For those identified clock pins, the following guidelines must hold:

1. Each scannable reg must have one and only one clock input pin enabled when the test control scan inputs are set to active and the scan clock that controls this reg is turned on.
2. Opposite phases of the same clock signal must not be logically combined. The allowable exception to this rule is when the signals reconverge in a user-specified clock chopper. Refer to “Guideline TG.9 - Clock Choppers” on page 52.
3. If two different clock primary input signals are logically combined, the stable value of the two signals at the logic gate must be the non-controlling value (zero for OR/NOR, one for

AND/NAND) of the gate. This ensures that a change on only one of the clock PIs will switch the gate.

Test Guideline TG.6 - Only One Port Active

While turning ON at most one clock primary input at a time, it should not be possible to simultaneously activate the clock inputs of two or more separate ports of the same Flip-Flop, Latch or RAM.

Violation of this guideline will cause a race when the clock is turning OFF at the memory element due to the uncertainty about which port was the last to control the value into the Flip-Flop, Latch or RAM. Encounter Test simulators will perform pessimistic simulation such that when two ports attempt to write opposing values into the same memory element (Flip-Flop, Latch, or RAM bits at the same address), an unknown value is predicted. While this may provide adequate pessimism for expect values, it does not account for any technology-specific problems such as a high current condition that could result from simultaneous writes of opposing values.

Verification Process

Sets various testmode design states and interrogates the clock inputs of the memory elements.

Intent

The test generation system must be able to control when memory elements change state. Likewise, it must be able to change data PIs without disturbing the contents of memory elements that are maintaining the design state. Further, data stored in non-scan regs or RAMs must remain unchanged until the reg or RAM is clocked. Violations of this check may cause erroneous test data to be produced.

Outcome

The following messages may be produced by this test:

TSV-003	TSV-007	TSV-008
TSV-009	TSV-010	TSV-011
TSV-012	TSV-013	TSV-015
TSV-016	TSV-017	

Analyze Clock Choppers

The following describes the test guidelines, verification process, and intent for this check.

Guideline TG.9 - Clock Choppers

A clock chopper is a sequential circuit that emits a short width pulse when it receives an input clock transition in a specific direction. This behavior makes it necessary that specific modeling guidelines be followed in order for Encounter Test to reliably and efficiently process the clock choppers. Encounter Test supports both leading-edge and trailing-edge clock choppers. In both cases the clock chopper may be explicitly identified by the presence of a CHOP primitive in its model to denote that the technology provider is assuring that the inherent race of a chopped signal has been correctly timed and will not cause zero-yield problems. Logic structures which behave as clock choppers but do not include the special Encounter Test CHOP primitive blocks are identified implicitly, but without the technology provider's assurance, the generated test data will be suspect. See "Clock Chopper Primitives" in the *Encounter Test: Guide 1: Models* for information on correct modeling of clock choppers.

1. Cascading more than one trailing-edge chopper is not formally supported by Encounter Test. ATPG may generate patterns with poor fault coverage.
2. You can cascade any number of leading-edge choppers and they will all produce an output pulse of the same shape and timing (for zero-delay simulation).

Verification Process

Clock choppers are identified by tracing back from memory element clock inputs. The verification checks each of the identified clock choppers to determine if each clock chopper has valid structure. It also determines if there are cascaded trailing edge choppers.

Intent

These guidelines ensure that clock choppers are properly controlled so that valid test data will be produced.

Outcome

The following messages may be produced by this test:

TSV-021	TSV-022	TSV-023
TSV-025	TSV-026	TSV-027

TSV-028	TSV-029	TSV-038
TSV-039	TSV-041	TSV-044
TSV-045	TSV-046	TSV-048

Analyze Flip-Flop and Latch Scan Characteristics

The following describes the test guidelines, verification process, and intent for this check.

Test Guideline TG.8 - Section Scan State

It must be possible to define a scan operation composed of one or more scan sections, each of which scans in parallel one or more scan chains. This is made possible by having all latches and flip-flops interconnected to form one or more scan chains whose scan clocks are identified and controllable during test. The sequential application of the defined scan sections results in the scanning of all scannable memory elements. For each of these scan sections a section scan state must exist, having the following properties:

1. Each scan chain of the scan section must have its own unique scan-in pin and scan-out pin.
2. During the scan operation each latch of a section scan chain is a function of only the single preceding latch in the scan chain or the scan-in pin, as defined above.
3. Any scan clock to a flop/latch must be turned "on" and "off" by changing just the corresponding clock primary input for each clock.
4. It must be possible to scan data forward by one bit position through all the scan chains of the scan section in parallel by first applying the scan preconditioning sequence and then applying one iteration of the scan sequence.

   Note: Unless you provide a custom scan protocol, this one bit scan must be accomplished by first putting all test pins at their scan state values and then pulsing all scan A clocks in numeric order, then all scan E clocks in numeric order, and finally all scan B clocks in numeric order. Refer to Double Latch Design in the *Encounter Test: Reference: Legacy Functions*.
5. The clock inputs to non-scanned latches and write clock inputs of RAMs must be held "off" (to logic zero) during scan.

See "Scan States" in *Encounter Test: Guide 2: Testmodes* for additional information.

Verification Process

Set the scan state and use a combination of tracing and simulation to ensure that:

- each reg's scan data input is fed either from the preceding reg in the scan chain or else from a SI PI.
- each SO pin is fed from the last reg in the scan chain.
- scan chains can be exercised in parallel.

Intent

Connecting regs into scan chains allows them to be directly controlled and observed by way of the scan path, thus immensely simplifying the test generation task. In addition, by requiring that all scan chains shift in parallel, the elapsed tester time is reduced.

Note:

1. Clocking violations will cause the scan chain to be "broken" even if it is topologically connected.
2. Scan chain path gating that includes logical redundancies may cause the scan chain to go unrecognized.
3. All scan chains must be scannable in parallel.
4. Regs that are not connected in scan chains are identified as floating.

Outcome

The following messages may be produced by this test:

TSV-070	TSV-071	TSV-072
TSV-073	TSV-074	TSV-075
TSV-076	TSV-077	TSV-079
TSV-080	TSV-081	TSV-083
TSV-084	TSV-085	TSV-086
TSV-239	TSV-270	TSV-271
TSV-272	TSV-273	TSV-274
TSV-276	TSV-277	TSV-279

TSV-280	TSV-281	TSV-283
TSV-284	TSV-285	TSV-286

Analyzing Explicit Fixed Value Latches/Flops

The following describes the test guidelines, verification process, and intent for this check.

Test Guideline TG.11 - Fixed Value Latches/Flip-Flops

A fixed value reg functions, in a given test mode, as a tie block or a one-cell ROM. It can be used to drive anything that a TI-attributed primary input might feed. Encounter Test supports fixed value regs which conform to the following guidelines:

1. The reg must be initialized via the mode initialization sequence.
2. The reg must be stable (hold its current state) when all primary input TI signals and latch TI signals are applied, regardless of the state of any other flop, latch, clock, or data primary input.
3. A fixed value reg may be clocked as long as its new state is guaranteed to be identical to its current state. In the case of a TI-attributed fixed value reg, this means that the new signal must be the same as that specified by the TI attribute, and does not necessarily have to derive from the state of the latch/flop itself.

Note: Fixed value regs that are assigned as FLH (fixed value linehold) may have the fixed value changed by specifying a linehold on the reg during test generation.

Verification Process

Analyze each reg with a test function of TI or FLH to ensure its initialized value matches the stability value. Ensure the fixed value regs remain at the fixed value.

Intent

Ensures that each identified fixed value reg provides a constant signal in the given test mode.

Outcome

The following messages may be produced by this test:

TSV-110	TSV-111

Analyze Default Fixed Value Flip-flops

The following describes the test guidelines, verification process, and intent for this check.

Encounter Test supports fixed value regs as described in the previous check (see "Test Guideline TG.11 - Fixed Value Latches/Flip-Flops" on page 55).

Verification Process

Determine the test function to apply to any regs that are identified as fixed value but do not assign an explicit test function. The test function assigned is the value (TI or FLH) specified in the FIXED_VALUE_LATCH statement in the mode definition file. If this statement was not included in the mode definition file, then the reg is assigned to be a TI.

Intent

Ensure that all fixed value regs have a test function assigned, and that the user is aware of the test function being assigned.

Outcome

The following message may be produced by this test:

TSV-115

Analyze Potential Clock Signal Races

The following describes the test guidelines, verification process, and intent for this check.

Test Guideline TG.4 - Clock vs. Data Races

If a level-sensitive storage element is controlled by a clock primary input such that it can change state when this clock primary input is set to value V (0 or 1), or if an edge-sensitive storage element can change state when the clock primary input switches to value V, then:

1. This storage element should not feed to the data port of a level-sensitive storage element whose clock input can be set ON by setting this clock PI to value V nor should it feed to an edge-sensitive memory element whose clock input is sensitive to the opposite edge of the same primary input clock.
2. This storage element should not gate this same clock signal except when value V at the PI results in the clock signal dominating the gating logic (zero at the input of an AND/NAND gate, one at the input of an OR/NOR gate).
3. The clock primary input should not feed to a data pin of the same port that it controls.

Notes:

1. A level-sensitive storage element that feeds the data port of another level-sensitive storage element controlled by the same clock, but on opposite phases of that clock, causes a race condition whose outcome depends upon the clock skew to those memory elements; however, when the clock is changing at the PI, it is assumed that the clock arrives at both storage elements simultaneously and this will turn OFF at the receiving storage element before any new data arrives from the sending storage element. This is critical for effective use of edge-triggered flip-flops modeled with level-sensitive master and slave latches.
2. Clock gating from a storage element that is controlled by the same clock that is being gated causes a race condition. However, if the clock input to the gate is switching to the controlling value of the gate when a new value may be launched from the storage element, it is assumed that the clock signal will propagate to the gate before any new value from the storage element, effectively blocking any such transitions so that no glitch occurs.
3. A violation of provision one of this guideline can be tolerated if the clock pulse can be guaranteed to be wide enough to propagate all storage element changes through all other storage elements whose clock is also ON. This assumption is likely to be correct as long as the clock signal does not feed through a clock chopper on its way to any of these latches. It is not possible to tolerate a violation of provision one of this guideline if the receiving memory element is a level-sensitive RAM and the data race feeds to a write port address input.
4. If this guideline is violated in the stable clock state (flushed latches directly communicating), then the race is of most concern if the clock involved is pulsed two or

more times in rapid succession. When this happens, there is the possibility that the latch on the receiving end of this violation will have its clock input turn off (opposite stability) before the new data value from the sending latch has had a chance to propagate to it. This would result in a miscompare at the tester. As long as this clock is pulsed but once per tester cycle, however, then there should be enough propagation time between pulses to prevent this from happening.

5. When a clock feeds to the data input of a memory element that it controls or to an address input of a RAM that it writes into, there is an inherent race condition. As the clock transitions to capture the data input value, the value may change due to the clock's influence on the data input logic. This may violate the hold time for the memory element. Such a race may be tolerated if it is known that there is no hold time violation and that the value captured by the memory element is the value seen at the memory element data input prior to the clock primary input switching. The presumption will be that the clock edge that captures the input states will arrive at the memory element before any data or address input changes occur without violating hold times. For level-sensitive RAM address inputs, it will be assumed that the address changed due to the write clock going ON will be the address written and not the address present prior to the clock going ON at the RAM.

Verification Process

- Check for all cases where one memory element feeds another memory element and both are controlled by the same phase of the same clock. If the optional mutually exclusive gating logic (MEG) check is enabled, additional checking is performed to ensure that the design conditions enable the clocks to the memory elements and that a path is sensitized between the memory elements. If the gating blocks the data and/or clock path, no messages are issued.

All clock signals, whether chopped or not, are assumed to be of infinitely long duration

Intent

To limit clock and data races to just those situations where the GSD clock-wins-race assumption can be relied upon to produce valid simulation results.

Race conditions occur when:

- a data signal to a storage element depends upon a clock that controls the storage element
- a clock signal to a storage element depends upon data controlled by the same clock
- multiple ports of a storage element are clocked simultaneously

Adherence to these guidelines allow an automatic pattern generator to generate a set of patterns which produce predictable results regardless of the actual design delays.

Note: TSV honors three attributes whose effect is to reduce the number of potential race conditions which TSV identifies. These three attributes are the `CLOCKED_FROM=NO`, the `CLOCKED_TO=NO`, and the `LPD=YES` attributes.

- ❑ The `CLOCKED_FROM=NO` attribute reduces potential violations by telling TSV not to consider data controlled by the write clock to the RAM as clocked data. That is, TSV will not consider any flip-flop or RAM with this attribute to be a source of clocked data. Use of this attribute assumes the design designer has guaranteed that the race condition will be correctly simulated by Encounter Test.
- ❑ The `CLOCKED_TO=NO` attribute reduces potential violations by telling TSV not to check for any race conditions at the RAM or latch with the attribute. That is TSV will not consider any flip-flop or RAM with this attribute to be a capture point for clocked data. The design designer has guaranteed that the race condition will be simulated by Encounter Test.
- ❑ The LPD (Lower Port Dominant) attribute reduces potential violations by telling TSV that the technology guarantees that if two Write Clock inputs can become active at the same time, then the data from the (visually, not numerically) lower port will ALWAYS be captured into the RAM.
- ❑ To ensure simulation results are accurate, the LPD attribute should be specified by the technology rules developer when lower port dominance is guaranteed.

Outcome

The following messages may be produced by this test:

TSV-018	TSV-019	TSV-020
TSV-050	TSV-051	TSV-052
TSV-053	TSV-059	TSV-060
TSV-062	TSV-069	

Ensure External Three-state Drivers are Disabled

The following describes the test guidelines, verification process, and intent for this check.

Test Guideline TBT.7 - Three-State Drivers

There must exist some product input pins (those identified as BI test function pins), which force all TSDs that directly feed product output pins to high impedance. Refer to Guideline TBT.7 - Three-State Drivers on page 69 for more information.

Verification Process

Set all primary inputs with test function BI to their stability value. Check all TSDs that feed directly to primary outputs to ensure they are at high impedance.

Intent

By ensuring that TSDs which drive primary output nets can reach high impedance, it is less likely that damaging conditions will occur when testing the product. In addition, better diagnosis of errors associated with TSD nets can be performed.

Outcome

The following messages may be produced by this test:

TSV-090	TSV-091	TSV-092

Ensure X-sources Cannot be Observed

The following describes the test guidelines, verification process, and intent for this check.

Test Guideline TBS.2 - Unpredictable Signal Values

Sources of unknown or oscillating signal values (modeled as a logic X in Encounter Test) must not exist in the logic under test, or must be blocked from reaching an observation point during the application of signature-based test vectors (OPMISR compression, LBIST, WRPT).

Sources of logic X are:

1. RAM primitive output pins when the RAM has not been initialized to known, reproducible signal values or the contents of the RAM are corrupted by the scan sequence.

2. ROM primitive outputs when the read enable input is NOT tied active (logic one) and when the value the output pins achieve when not reading (specified by the READOFF attribute) is X, or a READOFF value was not specified.
3. RAM primitive outputs when reading an “indeterminate” address and the specified output value is X (specified by the MASKOUT attribute) or unspecified.
4. TIEX blocks
5. Sourceless nets whose value is indeterminate. The value is indeterminate if no pin in the net had a non-X TIE value specified or some pins had conflicting TIE values specified (a value of one on one pin, zero on another).
6. Unterminated internal TSDs that can achieve the high impedance state.
7. Three-State Drivers (TSDs) with contention.
8. Product input pins that are not connected to the tester.
9. Global feedback networks that produce oscillating signal values.
10. A global feedback network which latches a value. This is considered an uninitialized memory element.
11. Multi-source nets driven by primitives which can produce conflicting values which can result in an unpredictable state on the net.
12. PRPG Save Register latches.
13. Floating regs that are not initialized to known signal values or are corrupted by the scan sequence.
14. Any keeper device that is not precharged (clock-off), and for which all strong sources can be disabled (i.e., the keeper can latch up).
15. Any flop/latch or RAM that has two or more ports clocked by the same clock.

    Note: An exception to this is that a RAM may have multiple ports clocked by the same clock as long as the LPD attribute is present in the RAM's logic description to specify the outcome of the resultant race.

Verification Process

Find all sources of X and simulate them to ensure they are blocked from being observed.

Intent

For signature-based testing, this check identifies any sources of logic X signals that may be captured in a signature collection register during the application of stimulus to the product under test. Failure to fix these violations means that the generated signatures may be invalid.

For stored pattern testing, this check is optional. If run it identifies sourced of logic X that may be observed and, as a result, lower test coverage.

Outcome

The following messages are produced by this test

TSV-101	TSV-103

Ensure Non-test Pins Cannot be Observed (RPCT Boundary)

The following describes the test guidelines, and intent for this check.

Test Guideline

In the reduced pin count test environment, input pins that do not have test function pin attributes are not contacted by the tester, and therefore must have boundary latches which provide the necessary stimulus to the product while under test.

Intent

This guideline ensures that the uncontacted inputs cannot be observed.

Note: Reduced Pin Count Test assumes the tests generated for the design under test may be migrated for use on a higher level package where the uncontacted pins may be connected to other logic. Therefore, any uncontacted pins are assumed to be at X.

Outcome

The folliwng are messages produced by this test:

TSV-102	TSV-104

Analyze On-Product Control Logic

The following describes the test guidelines and intent for this check.

Test Guideline OPC.1 - OPC Inputs

OPC logic must be controlled solely from primary inputs and fixed value latches.

Each signal produced by OPC logic and feeding into the “standard” logic which Encounter Test applications work on must depend, in any given pattern sequence, upon signals specified in a user-supplied clocking template and applied at the package primary inputs and static control signals. The “static control signals” may be fixed value latches which were pre-loaded by the mode initialization sequence and cannot be changed thereafter, or they may be pre-loaded by a *setup* sequence and then held fixed during the succeeding tests which use a specific user-supplied clocking template.

This expressly prohibits any latch whose state varies from one test vector to the next, from feeding back into the OPC logic in such a way as to affect the output waveforms (or the internal state) of the OPC logic.

Intent: This guideline that makes it feasible for Encounter Test to ignore the OPC logic by constraining the OPC logic output signals to those patterns or waveforms that can be produced by primary input sequences (clocking templates) predefined by the user. If the OPC logic output signals could be dynamically altered by the operation of the downstream logic, then the task for the user to predefine its behavior and the task for Encounter Test to properly manage it would both be greatly complicated.

Test Guideline OPC.2 - OPC Fixed Value Latches

The fixed value latches within OPC logic must be pre-loaded by a user-supplied sequence and must be held fixed throughout any sequence during which their values may affect the OPC outputs.

This is basically a description of how fixed value latches are generally used. The wording of this guideline is intended to support latches of the TC variety, which hold a pre-defined, constant value during a test sequence, but which may be altered as necessary during a scan operation. For interpretation of this guideline, a narrow definition of “test sequence” must be used, to include only the activity between scan operations, treating the scan as a separate, distinct sequence. Most other places in Encounter Test documentation refer to a test sequence as including any imbedded scan operations.

This guideline prohibits a “linehold” variety of fixed value latch from affecting the OPC outputs. The state of a linehold fixed value latch (as opposed to a TC latch) can be specified at test generation run time; therefore the value cannot be pre-defined by the test sequence definition (clocking template), since the same test sequence definition may be used in several different test generation runs. The same effect as a linehold latch can be produced by a latch which is dynamically loaded in the “setup” sequence applied before running a given set of repetitions of a given test sequence.

Intent: The use of a “linehold” (FLH) latch inside OPC logic would generate invalid tests if the value of this latch were ever specified to be different from the value assumed by the test sequence being applied, because Encounter Test does not simulate the OPC logic itself, but depends instead upon the correctness of the OPC output values specified by the sequence definition supplied by the user.

Guideline OPC.3 - OPC Floating Latches

Floating latches are allowed, and expected in OPC logic.

Usually these latches are initialized by either the mode initialization sequence or (for a test sequence) by a setup sequence. These latches may switch during the application of any given sequence, or may remain constant, as dictated by the needs of the OPC logic design.

Intent: This guideline is simply a clarification of the fact that floating latches can be used in OPC logic. It is not necessary that they be used if the OPC logic function does not require such latches. Aside from bugs in the OPC design, there is no way to use them in a way that could be considered invalid.

Test Guideline OPC.4 - OPC Scannable Latches

It is not recommended that any scannable latch be used within OPC logic, but this is permissible as long as “Test Guideline OPC.1 - OPC Inputs” on page 63 is adhered to.

Scannable latches are, in general, dynamic, and the use of one inside the OPC logic that is feeding the logic containing the scannable latch might seem to violate Guideline OPC.1. However, the sequence definition may specify the state of the latch so that the loaded value controls some subsequent OPC output values during the sequence. The spirit of Guideline OPC.1 is not violated if the OPC outputs are independent of the scannable latch until after the scan operation (which is a primary input sequence) has been applied, and then this scannable latch is either held fixed or controlled in some deterministic way throughout the remainder of the sequence. Note that if used, the scannable latch must always be loaded to a predefined value by the test sequence so that a test generator would never be allowed to specify an arbitrary value for it.

Intent: Most scannable latches are used to supply input vectors to the logic which they feed, and therefore their values are not predetermined by the user-supplied sequence. The use within OPC logic of a latch, either scannable or not scannable, which is not deterministically controlled solely by the events specified in a given user-supplied clocking template, precludes Encounter Test from inferring the behavior of the OPC logic from the clocking template, and will very likely cause invalid tests to be generated.

Test Guideline OPC.5 - OPC RAMs

A RAM output cannot feed OPC logic.

While in theory, a RAM could be treated the same as a floating latch, and therefore used in OPC logic in accordance with “Guideline OPC.3 - OPC Floating Latches” on page 64, Encounter Test does not support this due to the added complexity.

Intent: You may deviate from this guideline and still conform with the spirit of “Test Guideline OPC.1 - OPC Inputs” on page 63, but this will make the sequence definitions more complicated and you will be plagued with severe error messages from Test Structure Verification and OPC Sequence Verification. Because of this, there is high exposure to undetected errors and deviation from this guideline is strongly discouraged.

Test Guideline OPC.6 - Oscillators

An oscillating signal (a primary input connected to a free-running oscillator) must not be observable except through OPC logic.

In this context, the oscillating signal is not considered to be “observed” by feeding a latch within OPC logic. In fact, it is the function of such latches to synchronize other external events with the oscillating signal and produce well-behaved signals on the outputs of the OPC logic.

Intent: When simulating tests, Encounter Test sets oscillating signals to X (unknown). If an oscillator is observable (bypasses the OPC logic), then the X value will give pessimistic, but possibly usable, results for stored pattern tests will corrupt the signatures for WRPT and LBIST, rendering these tests worthless. The signals coming out of the OPC logic are specified by user-supplied sequences, and should properly take into account the influence of the oscillator on the OPC logic behavior.

Outcome

The following messages may be produced by this test:

TSV-120	TSV-121	TSV-122
TSV-123	TSV-124	TSV-125

Ensure Compatibility between Parent and Child Modes

This section describes the test guideline and intent for this check.

Test Guideline

Check for conditions that preclude the conversion of child mode test patterns into parent mode test patterns.

For additional reference, refer to "Multiple Test Modes" in the *Encounter Test: Guide 2: Testmodes.*

Intent

This mode compatibility check verifies the latch types/values across the child/parent test modes.

Outcome

The following messages may be produced by this test:

TSV-245	TSV-246	TSV-247
TSV-248	TSV-249	TSV-250
TSV-251	TSV-252	TSV-253
TSV-254	TSV-255	TSV-256
TSV-257		

Test Coverage Guidelines

The following guidelines describe design configurations which cause Encounter Test applications to run longer, achieve lower test coverage or both.

These guidelines apply to both LSSD and GSD

Guideline TBT.1 - Combinational Logic Functions

Inputs to a combinational logic function should not be fed by latches which are adjacent in the scan path.

Consequence: Encounter Test test generation applications allow for both normal and skewed LSSD scan chain operation. However, both operations can result in each L2 having the same data as its preceding L1 (normal shift) or its succeeding L1 (skewed shift). Violation of this guideline can impact test coverage due to the inability to obtain necessary latch control values by an LSSD load operation.

Guideline TBT.2 - Reconvergence of Clock Signals

Clock signals originating from the same clock source must not reconverge in-phase. Out-of-phase reconvergence is allowed only within a clock chopper (see "Guideline TBT.7 - Three-State Drivers" on page 69). Clock signals originating from different sources must not be logically ANDed together.

Consequence: A deviation from this guideline as stated above will almost surely result in limiting the test coverage that can be obtained.

If different phases of the same clock are ANDed, some faults will remain untested in static test environment. If different clocks are logically ANDed, faults will remain untested due to ATPG restrictions that normally allow only one clock ON at a time.

Some static faults on the ANDing circuit or feeding it may not be observable due to the restrictions imposed by the logic in the case of reconvergence or imposed by ATPG (only one clock allowed ON at a time) in the case of ANDing different clock sources.

Note: ANDing one clock with the invert of another clock prevents observation of "clock stuck OFF" faults in the second clock's tree that do not have an alternate observation path.

When two or more clock signals converge at an AND/NAND gate and at least one of the clock signals presents a 0 to the input of the NAND gate when that clock is OFF, this is considered to be a logical clock ANDing. When two or more clock signals converge at an OR/NOR gate and at least one of the clock signals presents a 1 to the input of the OR/NOR gate, this is also considered logical clock ANDing. Logic clock ANDing occurs whenever the clock OFF state of at least one of the convergent clocks presents the controlling value to the input of the convergent gate.

When two or more clock signals converge at a gate and all clock inputs to the gate are at the non-controlling value when the clocks are all at their OFF state, this is considered to be logical clock ORing, which is not a DFT concern.

Guideline TBT.3 - Data inputs to RAMs and Latches

Clock pins should not feed the data inputs of latches, or the address or data inputs of RAMs either directly or through combinational logic, whether they are chopped or not.

Consequence: Some faults may not be tested since Encounter Test activates only one clock at a time.

Guideline TBT.4 - Scan Function Controls

If the clock network that controls the scan function of an L2 or L3 (the Scan B and Scan P controlled ports respectively) is other than a simple powering tree fed from a single product clock input, then the derived (or gated) scan clock signal should feed to a product output pin or to a system clock input of an L1 or L2. (If the signal feeds to a latch, the Encounter Test Guidelines TB.3 and TB.5 must not be violated). Refer to "Guideline TB.3 - Memory Elements" and "Guideline TB.5 - Scan Chain" in the *Encounter Test: Reference: Legacy Functions*.

Consequence: Lower test coverage.

Guideline TBT.5 - RAM Input Controls

If the clock network that controls the read enable or write clock input of a RAM is other than a simple powering tree fed from a single product clock input, then the derived (or gated) signal should feed to a product output pin or to a clock input of an L1 or L2. (If the signal feeds to a latch, the Encounter Test Guidelines TB.3 and TB.5 must not be violated). Refer to "Guideline TB.3 - Memory Elements" and "Guideline TB.5 - Scan Chain" in the *Encounter Test: Reference: Legacy Functions*.

Consequence: Lower test coverage.

Guideline TBT.6 - Clock Choppers

The following testability guidelines exist for clock choppers:

1. To ensure complete testability of a clock chopper, the chopped signal should be fed to a product output pin.
2. If the signal feeding into a clock chopper is gated (combined with non-clock data), then the gated signal should be fed to a product output pin.

3. A chopped clock should not act solely as the scan B clock for L2/L4s, the scan P clock for L3s, the write clock input to a RAM, or as the clock input to an L5. For test purposes, it must also be used as the system clock input of an L1 or L2 and not violate Encounter Test Guidelines TB.3 and TB.5. Refer to “Guideline TB.3 - Memory Elements” and “Guideline TB.5 - Scan Chain” in the *Encounter Test: Reference: Legacy Functions*.

Consequence: Lower test coverage.

Guideline TBT.7 - Three-State Drivers

The following testability guidelines exist for networks with Three-State Drivers (TSDs) and transistors (NFET and PFET):

1. Each net driven by a TSD or a transistor primitive must be terminated to a logical one or zero or have a resistor dotted to the net unless it is directly connected to a package output pin.
2. There must exist some product input pins (those identified as BI test function pins), which force all TSDs that directly feed product output pins to high impedance.

Consequence

1. If net driven by TSD and transistor primitives is not terminated, then if all sources on the net are inhibited to high impedance, then this is treated as an unknown (X) by down stream logic and results in faults that cannot be definitely tested. This is particularly serious when using signature analysis (WRP).
2. Failure to properly identify a BI pin may cause problems for the manufacturer in verifying that the Three-State Drivers can reach a high impedance state.

Guideline TBT.8 - Three-State Contention

The enable inputs of Three-State Drivers (TSDs) and the gate inputs of transistor primitives (NFET or PFET) that drive the same net must be controlled so that no two sources can be actively driving conflicting values during the application of the test data, unless the sources are burnout proof and no product damage can occur. Refer to

Refer to “Analyze Three-state Drivers for Contention” on page 48 for additional information.

Limitation Guidelines

The conditions outlined below are program limitations of the applications within Encounter Test. These limitations exist to allow Encounter Test to process designs efficiently and effectively. If a design does not adhere to these guidelines the test data is suspect.

These guidelines apply to both LSSD and GSD.

Guideline TBL.1 - Clock Isolation (CI) Pins

If multi-function clock pins are specified, having both a scan and a system function, at most one CI pin is allowed. The CI pin enables distinction between the scan function and the system function of multi-function clock pins, and can greatly improve the performance of the race checking in TSV.

If more than one CI pin is specified, Encounter Test arbitrarily picks one and treats the rest as scan enables (SEs).

Note: Multi-function clock pins are identified as either AS, BS, ES, or PS.

Consequence: Only 1 clock isolation pin will be applied. If it is not the correct pin, you may lower test coverage and cause TSV to identify severe errors.

Guideline TBL.2 - Redundant Signals

Redundant signals in the scan path logic are not allowed because they get in the way of the tracing that identifies the scan path. Without this limitation TSV would either have to exhaustively simulate all combinations of signals feeding a latch or perform a test-generation-like path sensitization function to verify the operability of the scan logic.

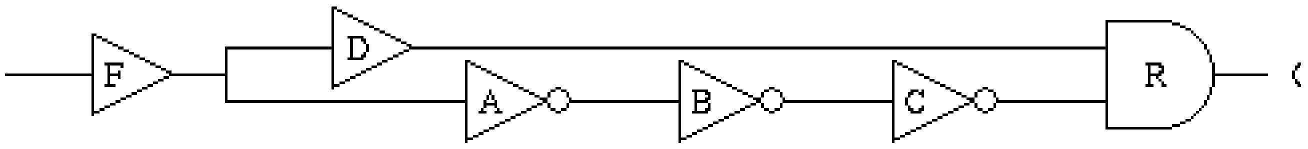
Consequence: TSV may become confused by the redundant logic and would not know which path to follow when tracing the scan path. This means it would be unable to identify the scan chains.

Guideline TBL.4 - Clock Choppers

This guideline identifies restrictions associated with clock choppers. Encounter Test supports both leading-edge and trailing-edge clock choppers. In both cases the clock chopper should be explicitly identified by the presence of a so-called CHOP block in its model to denote that the technology provider is assuring that the inherent race of a chopped signal has been correctly timed and will not cause zero-yield problems. Designs that behave as clock choppers but do not include the special Encounter Test CHOP primitive blocks may be used, but without the technology provider's assurance, the generated test data will be suspect. For those customers who must use technology cell definitions which do not have the special Encounter Test CHOP primitives in them, an alternative cell definition can be specified for those cells which use clock choppers.

See "Clock Chopper Primitives" in the *Encounter Test: Guide 1: Models* for additional information.

Figure 1-24 Example of a "Legal" Clock Chopper Circuit (TBL.4)



The following rules apply to clock chopper circuits (refer to Figure 1-24 on page 71).

1. There must exist a single fan-out node (*F*) and a single reconvergent node (*R*). The reconvergent node (*R*) must be an AND, NAND, OR or NOR gate - other gate functions are not supported.
2. There must be exactly 2 inputs of the reconvergent node (*R*) which are fed from the fan-out node (*F*) and the two paths between the fan-out node (*F*) and the reconvergent node (*R*) must have different inversion levels (i.e. one path has an even number of inversions and the other has an odd number of inversions).
3. One of the two paths between the fan-out node (*F*) and the reconvergent node (*R*) must be at least one unit of delay longer (counting unit- and zero-delay flatModel nodes) such that this path can be designated the "long path".
4. The long path's last node (*C*), which feeds the reconvergent node (*R*) must feed only to the reconvergent block (*R*). For explicitly defined leading-edge choppers this node (*C*) should be a CHOPL or NCHOPL Encounter Test primitive; otherwise it is an implicitly defined clock chopper.
5. For clock choppers which behave as trailing-edge choppers, the short path must have at least one node on it and the last node in the short path (*D*) must feed only to the reconvergent block (*R*). For explicitly defined trailing-edge clock choppers this node (*D*) should be a CHOPT or NCHOPT Encounter Test primitive; otherwise it is an implicitly defined clock chopper.

6. If any of the nodes on the long or short paths of a clock chopper have multiple inputs, all inputs which are not in the short or long path must be set to the non-controlling value for that node when the TIE/TI circuit state is applied such that if a 0/1 is simulated at the output of the fan-out node (*F*) when in the TIE/TI state, that a 0/1 or 1/0 is seen at the reconvergent node's inputs for the short and long paths. The sole exception to this rule is that for trailing-edge clock choppers, it is legal for nodes in the long path (*A* , *B,* or *C*) to have another clock signal logically ORed in if and only if the clock OFF state for this other clock signal yields the non-controlling value at the long path node. This exception simply allows clock ORing logic to be built using nodes which already exist to create delay within the long path of the clock chopper. It can apply similarly to the short path for leading-edge clock choppers, but there is no advantage to adding delay in the short path of a clock chopper.
7. If the reconvergent node (*R*) has more than 2 inputs, the other inputs are considered gating signals to the chopped clock as if the reconvergent gate (*R*) were logically constructed from a cascade of a 2-input gate serving as the reconvergent node for the clock chopper and another gate which logically gates the chopped clock signal.
8. The clock chopper output (*R*) must be controlled to the OFF state when the clock primary inputs are in their OFF state.
9. Clock choppers can be cascaded such that the output of one clock chopper feeds to the input of another clock chopper. The lone exception is that it is not legal to cascade the output of a trailing-edge clock chopper to the input of another trailing-edge clock chopper regardless of whether there are intervening leading-edge clock choppers.

Consequence: A clock chopper is a sequential circuit which emits a short width pulse when it receives an input clock transition in a specific direction. This behavior makes it necessary that these guidelines be followed in order for Encounter Test to reliably and efficiently process a circuit containing clock choppers. Failure to observe these guidelines may result in the generation of invalid test data, or at the very least it can prevent correct simulation of clock signals resulting in very poor test coverage.

Guideline TBL.5 - Output Inhibit (OI) Test Function

There may be at most one OI test function input pin which forces all the primary outputs of the design under test, except SO, to the high impedance state when held at the specified logic value.

The OI pin identifies the signal which drives all TSDs feeding product outputs to their high impedance state independently. This removes the noise problem if many output devices simultaneously change circuit values.

Consequence: The skewing between product outputs is easier to control by the circuit designer if there is only 1 OI pin. The use of multiple OI's is not guaranteed to effectively suppress noise from simultaneous output switching.

Verifying an Assumed Scan Test Mode

For assumed scan modes, care must be taken when overriding the default tests. Any added test may fail due to the use of assumed scan.

Design Attributes for Verify Test Structures

TSV Message Controls

The TSV Message Control attributes provide the capability to tailor the results of the Encounter Test Verify Test Structures application for special cases. It is not always possible to model the logic in a design exactly as the hardware will perform, and it is not practical for Encounter Test to simulate each potential ground rule violation in every possible state, therefore, there are occasions where Verify Test Structures will report a structural ground rule violation that is acceptable. In such a case, manual analysis is required to prove that this is, indeed, an acceptable violation.

After you have analyzed a specific TSV message or a group of messages and assessed their impact or severity, you can control how TSV reports and reacts to this logic. The TSV Message Control attribute allows you to:

- Increase the severity of a message by one level. This allows the flexibility that some errors are more severe if they occur in a specific part of the design.
- Suppress a specific TSV message for an instance or all instances of a module or cell type. Encounter Test does not report suppressed messages but notes their severity and generates the summary message WARNING (TSV-593) to notify the user about the suppressed messages.
- Completely eliminate a specific TSV message for an instance or all instances of a module or cell type. There is no summary of eliminated messages and their severity does not affect the final TSV run status. In other words, if the only "Severe" status messages are for those instances that are to be completely ignored, then the final TSV status will not report any severe message. Subsequent ATPG runs will not report to the user that severe TSV messages were issued. Please note that you will still see the TSV-593 messages for the suppressed messages.

Suppress or Eliminate TSV Messages with SUPPRESS_MSG Attribute

Use the `SUPPRESS_MSG` attribute to suppress or completely eliminate specific TSV messages for an instance or all instances of a module or cell type. This attribute is specified within the Verilog net list or technology library. To suppress or eliminate a message on all instances of a module or cell, apply the attribute immediately after the module statement. To suppress or eliminate a message only for an instance within a specific module, apply this attribute immediately after the instance within a cell.

To suppress a message, use the `TSV` prefix. To eliminate a message, precede the TSV message number with the `CSV` prefix.

During the Build Model process, Encounter Test reads the values of the `SUPPRESS_MSG` attribute and creates a message override file (`tbdata/TEImsgOverrides`). This message override file is used by Test Structure Verification to inhibit reporting of ground rule violations of the types specified in the TSV message names.

The following is an example of suppressing message TSV-101 for all instances of module REG5, eliminating TSV-051 and TSV-052 messages for instance I_0 within module REG5, and raising the severity of message TSV-118 for instance I_4 within REG5:

```
module REG5 (D_out, D_in, CLK, EN, RST) ;
//! TYPE="CELL"

//! SUPPRESS_MSG= "TSV101"
output [4:0] D_out;
input [4:0] D_in ;
input CLK, EN ;
input RST ;  //! TIE = "1"
SDFF  I_0 (D_out[0], D_in[0], CLK, EN, RST) ; //! SUPPRESS_MSG= "CSV051,CSV052"
SDFF  I_1 (D_out[1], D_in[1], CLK, EN, RST) ;
SDFF  I_2 (D_out[2], D_in[2], CLK, EN, RST) ;
SDFF  I_3 (D_out[3], D_in[3], CLK, EN, RST) ;
SDFF  I_4 (D_out[4], D_in[4], CLK, EN, RST) ; //! RAISE_MSG_SEVERITY= "TSV118"
endmodule
```

Override TSV Message Inhibition with KEEP_MSG Attribute

The `KEEP_MSG` attribute can be specified on either an entire cell, or a single instance within a cell. The value of the `KEEP_MSG` attribute contains a list of TSV message names.

The `KEEP_MSG` attribute is used for those cases where `SUPPRESS_MSG` is coded for a DEF block, but it is nonetheless desired that a particular usage of that DEF block not suppress the message. Starting from the lowest level in the hierarchy and preceding to the highest level, the last `SUPPRESS_MSG` or `KEEP_MSG` attribute encountered for a particular message prevails. The default is always to keep, which is resorted to only if no explicit `KEEP_MSG` or `SUPPRESS_MSG` was encountered in climbing the hierarchy.

During the Build Model process, Encounter Test will read the values of the `KEEP_MSG` attribute, and create a message override file. The message override file is used by Test Structure Verification to inhibit reporting of ground rule violations.

Raise TSV Message Severity with RAISE_MSG_SEVERITY Attribute

The `RAISE_MSG_SEVERITY` attribute can be specified on either an entire cell, or a single instance within a cell. The value of the `RAISE_MSG_SEVERITY` attribute contains a list of TSV message names, e.g., `RAISE_MSG_SEVERITY=TSV052, TSV051`. During the Build Model process, Encounter Test will read the values of the `RAISE_MSG_SEVERITY` attribute, and create a message override file. The message override file is used by Test Structure Verification to raise the severity level when reporting of ground rule violations of the types specified in the TSV message names, and only for the logic inside the cell or instance which contains the `RAISE_MSG_SEVERITY` attribute.

Flag Data Races with CLOCKED_FROM and CLOCKED_TO Attributes

The CLOCKED_FROM and CLOCKED_TO attributes are used to flag clock vs. data races which are known to be simulated correctly by Encounter Test. There are two ways to flag a sanctioned race: either place the CLOCKED_FROM attribute on the instance of the LATCH or RAM primitive which is the starting point for the race, or else place the CLOCKED_TO attribute on the instance of the LATCH or RAM primitive which is the ending point for the race.

For instance, if a RAM is clocked by the same clock as the latch which captures the output of the RAM, there is a race condition. When the clock turns on, data is written into the RAM and may also be read from the RAM. It is possible that this data then travels through several levels of combinational logic before it reaches the capture latch. If the clock pulse is wide enough, then the clock at the capture latch will still be on at the time of arrival of the new RAM data. The capture latch will then capture the data newly written into the RAM. On the other hand, if the clock pulse is not wide enough, the clock may turn off before the new RAM data arrives at the latch, and the latch thus captures the old RAM data. By default, Encounter Test flags this condition as a data vs. clock race during test structure verification. If the RAM primitive and its capture latches are designed in a single technology library cell, the technology rules coder can guarantee that the new data will arrive at the capture latch before the clock turns off. In this case the technology rules coder can indicate to Encounter Test that this race condition should not be flagged as a problem by test structure verification. To accomplish this, the technology rules coder has two options: either place the CLOCKED_FROM attribute on the RAM instance, with a value of NO, or else code the CLOCKED_TO attribute on the capture LATCH instance, once again with a value of NO. Clearly, for this simple example, it is probably easier to use the CLOCKED_FROM attribute, applied to just the single RAM instance, than the CLOCKED_TO attribute, applied to many LATCH primitive instances, one for each data-out bit of the RAM.

When running Verify Test Structures, the violations that check for the CLOCKED_FROM and CLOCKED_TO attributes are:

- TSV-050
- TSV-059

2 Verify 1149.1 Boundary Scan

Verify 1149.1 Boundary determines compliance with the IEEE 1149.1a 1993 and IEEE 1149.1 2001 standards. It also verifies that the Boundary Scan Definition Language Description (BSDL) for a given design is syntactically and semantically compliant with the IEEE 1149.1b-1994 and 2001 standards.

For information on verification suite and the test modes required for verification, refer to "Verification Suite" on page 83.

The verification patterns are designed to:

- Exercise the test access port (TAP) controller Finite State Machine (FSM), the instruction decode and the instruction register
- Exercise the instructions and test data registers defined by the BSDL
- Exercise the synchronous and asynchronous reset logic

BSV performs the verification by generating verification patterns and simulating them on the design being verified. The verification patterns and the detailed results of the verification are written to a readable results file. Refer to "Verify 1149.1 Boundary Results File" on page 185 for details.

The verification patterns can be written to TBD and viewed and analyzed via the graphical user interface. Refer to "Viewing Vectors" in the *Encounter Test: Reference: GUI* for additional information.

Encounter Test 1149.1 Boundary Scan Verification (BSV) does not verify every aspect of every rule defined by the IEEE 1149.1 standard. BSV does however, verify many of those aspects of the standard deemed necessary to ensure a high quality test at higher levels of packaging (boards, module, and so on). Refer to the "1149.1 Boundary Scan Verification

Checks” on page 87 for details on the scope of 1149.1 verification performed by Encounter Test and the rules that are checked for compliance.

Notes

1. To perform *Verify 1149.1 Boundary* using the graphical interface, refer to “Verify 1149.1 Boundary” in the *Encounter Test: Reference: GUI.*
2. To perform *Verify 1149.1 Boundary* using command lines, refer to “verify_11491_boundary” in the *Encounter Test: Reference: Commands*.
3. Refer to “BSDL File” in the *Encounter Test: Guide 2: Testmodes* for details on the use and configuration of BSDL.

Creating an 1149.1 Test Mode for Boundary Scan Verification

One BSDL file is required for defining an 1149.1 mode for Boundary Scan Verification. Refer to “Creating BSDL” in the *Encounter Test: Guide 2: Testmodes* for more information. *Build Test Mode* uses the BSDL `COMPLIANCE_PATTERNS` and `scan port identification` statements to define the 1149.1 test function pins for the design.

Note: Refer to Verify 1149.1 Boundary Test Modes on page 184 for details on BSV test modes.

Following is a sample mode definition file for Boundary Scan Verification:

```
Tester_Description_Rule =
dummy_tester;
scan type = 1149.1 boundary=no

     in = pi
     out = po;

implicit choppers = yes;
test_type = none;
faults = none;
```

Note: Test function pins are defined in the BSDL. Refer to “Test Function Pins for an 1149.1 Mode” in the *Encounter Test: Guide 2: Testmodes* for additional information.

Performing Verify 1149.1 Boundary

To perform *Verify 1149.1 Boundary* using the graphical interface, refer to "Verify 1149.1 Boundary" in the *Encounter Test: Reference: GUI.*

To perform *Verify 1149.1 Boundary* using commands, refer to "verify_11491_boundary" in the *Encounter Test: Reference: Commands.*

A license for either Encounter Test Architect, Encounter True-Time, or Diagnostics Engine is required to run Verify 1149.1 Boundary. Refer to "Encounter Test and Diagnostics Product License Configuration" in *What's New for Encounter® Test and Diagnostics* for details on the licensing structure.

Prerequisite Tasks

1. Prepare (if necessary), then import a gate-level Encounter Test model.
2. Define or select a test mode for the design. In the SCAN statement of the mode definition file, TYPE must be 1149.1. Refer to "Performing Build Test Mode" in the *Encounter Test: Guide 2: Testmodes* for more information about defining or selecting a test mode.

Input Files

The suffix *.gz,* indicating file compression, is attached to the files listed below if they were created with the storage versus performance environment variable set to `TB_MODE=COMPRESS`*.* Refer to "Using Commands to Manage Files and Disk Space" in the *Encounter Test: Reference: Commands* for additional information.

Boundary Scan Verification requires the following input files:

Logic Model Files

The following files resulting from *Build Model* are used as input to Boundary Scan Verification:

- `hierModel`

  This file contains the basic logical block, pin and net connections and defines how the hierarchy fits together. It supports an arbitrary number of levels, as required, to describe the logic hierarchy. This model contains all levels of logic down to and including the primitive gates.

- `hierAttributes`

The hierarchical model attribute file contains all the block, pin and net attributes.

- `flatModel`

  This file contains a flattened, primitive gate-level logic model which is the logical equivalent of the hierModel.

- `globalData`

  This file keeps track of all the Encounter Test-generated data and files. It stores audit information and registers experimental files and pattern statistics. It is also used for locking to support parallel processing.

Test Mode Files

The following files resulting from *Build Test Mode* are used as input to Boundary Scan Verification:

- modeInfo. *modename*

  This file contains model tables specific to the test mode.

- activeNodeMap. *modename*

  This file identifies which pins in the hierarchical model are active in the test mode.

- TBDseq. *modename*

  This file contains initialization sequences.

BSDL File

This BSDL file defines the 1149.1 test attributes for the design being verified. The file may come from a Test Insertion tool, the `write_bsdl` tool, or be manually created. Refer to the following for additional information:

- "Customizing the JTAG Macro" in the *Encounter Test: Reference: Legacy Functions*
- "BSDL File" in *Encounter Test: Guide 2: Testmodes*

BSDL Package File

A package file is an ASCII file that describes boundary scan cell definitions. The package file is in the form of a standard VHDL package file definition.

A package file path and filename can be optionally specified as input to IEEE 1149.1 BSV. Refer to "Input Files" for "Creating BSDL" in the *Encounter Test: Guide 2: Testmodes* for additional information.

Using Verify 1149.1 Boundary

To invoke Boundary Scan Verification using the Graphical User Interface, do the following:

1. Select *Verification - Verify - 1149.1 Boundary* from the main menu. The *Verify 1149.1 Boundary* window is displayed.
2. Click *Tests* to view and select the instructions or operations you wish to verify.

   If you are unsure which 1149.1 test logic is implemented for the design, select the *Select All* button and *Verify 1149.1 Boundary* will verify all instructions defined in the input BSDL file. Selecting *Public Instructions* will verify all public instructions defined in the input BSDL file. Likewise, selecting *Private Instructions* will verify all private instructions defined in the BSDL. The non-standard instruction list may be used to verify a specific list of non-standard public or private instructions. The IEEE 1149.1 Standard defines the standard instruction and describes what BSDL information is necessary in support of all instructions.
3. Select *OK* on the *Tests* panel and click Run to begin processing.
4. When *Verify 1149.1 Boundary* processing is complete, Encounter Test issues a beep.

Analyzing Verify 1149.1 Boundary Messages

1. Select the *Messages* Tab and click the *View Messages* icon; or *Window - Messages - Verify 11491 Boundary.*
2. The *IEEE 1149.1 Check Summary* window lists the total number of messages (for which analysis is supported) per test mode per check, resulting from the run. Select a row, then select *View.* The *1149.1 Boundary Scan Verification Message Summary* window is displayed listing total occurrences for each numbered message resulting from the check and abbreviated versions of the associated message text.
3. Select a message from the list, then select *View.* The *Verify 1149.1 Boundary Specific Message List* is displayed listing every occurrence of the selected message and the associated message text in entirety.

   The *Id* column indicates the breadth of message analysis capability for a message:

   ❑ An asterisk (*) preceding the message ID indicates a message with no analysis available.

- ❑ A pound sign (#) preceding the message ID indicate a message with minimal analysis available.
- ❑ The absence of a symbol indicates full analysis support is available.

4. To begin message analysis, select the desired message from the list, then select *Analyze.* The View Schematic window is invoked containing the portion of the design causing the message.
5. To assist in analysis, Explanation and User Response for messages can be viewed by selecting a message, then selecting *Message Help.*

Refer to "Message Analysis Windows" in the *Encounter Test: Reference: GUI* for additional information.

Note: A user-specified experiment is not valid when interactively analyzing 1149.1 Boundary Scan verification messages. The only valid experiment is the `11491expt` experiment name generated by the `verify_11491_boundary` command. If the GUI message analysis cannot find this experiment, it issues a message and provides the message analysis without simulating the corresponding circuit state.

Output Files

The suffix *.gz,* indicating file compression, is attached to the files listed below if the storage versus performance environment variable is set to `TB_MODE=COMPRESS`. Refer to "Using Commands to Manage Files and Disk Space" in the *Encounter Test: Reference: Commands* for additional information.

A summary of Verify 1149.1 Boundary results is written to the log file. In addition, Verify 1149.1 Boundary creates:

- IEEE11491ver.*`modename.experiment`*

  This is a readable file that contains the verification sequences, the actual results, and points of simulation miscompares (expected values vs. actual values). This file can be used to assist in identifying the cause of the miscompare (non compliance). Refer to "Verify 1149.1 Boundary Results File" on page 185 for additional information.

- TBDbin.*`testmode.experiment`*

  An experimental Vectors file or files containing verification sequences. A Vectors file created by Verify 1149.1 Boundary can be simulated for purposes of analyzing failing sequences. The file may also be exported to other pattern formats for use outside of Encounter Test (e.g. Verilog).

- TJCmessageFile.*`testmode`*

A file containing messages resulting from running Verify 1149.1 Boundary. A TJCmessageFile is created for each test mode that has messages.

- Test Mode files are optionally created if verification includes the iomapping checks (`TB_EXTEST_CAP_UPDT`) or register identification was requested or failed (TB_ *instruction name* `_SHIFT_DR`). Refer to "Verify 1149.1 Boundary Test Modes" on page 184.

Verification Suite

In general, a verification suite is a collection of verification patterns that are simulated against the design being verified. The verification patterns are encapsulated into test procedures referred to as Absolute Procedure Numbers (APN). Each APN test procedure represents a predefined BSV check that verifies one or more 1149.1 rules. BSV customizes the checks based on the BSDL for the design. BSV encapsulates the APN test procedures into test groups referred to as Absolute Group Numbers (AGN). Each AGN represents a predefined collection of related but independent BSV checks. A verification group is created for each verification request. For example, a verification request of `verify=reset,bypass` results in two AGNs each containing multiple APNs which contain patterns for verifying reset or bypass.

The test groups and test procedures are ordered with the intent of simplifying analysis/diagnosis of failures in the verification patterns. For example, testing the ability to reset the TAP is performed before verifying boundary register functionality. However, the checks themselves are design to be independent of one another. This simplifies the analysis process by reducing simulation overhead when interactively analyzing failures.

Verification Modes

Presently, two test modes are required for a complete verification. One mode is created by the user and is referred to herein as the '`BASE`' mode, the other mode is typically created by Encounter Test BSV/BSDLgen and is referred to as the '`TB_EXTEST_CAP_UPDT`' or '`EXTEST`' mode. Based on the way the `TB_EXTEST_CAP_UPDT` mode is defined, checking is limited to checks related to the `EXTEST` instruction and its use of the boundary register. The only register which is scannable in this mode is the boundary register (cannot scan in a different `EXTEST` opcode since the instruction register cannot be scanned). The `BASE` mode is the traditional 1149.1 BSV test mode where there is no restriction on the 1149.1 registers which may be scanned. Given that multiple modes are required for complete verification, it follows that multiple uncommitted tests are required to contain the complete set of verification patterns.

All BSV checks begin in one of two known states which is established by the mode init `Test_Procedure`:

If the test mode is the `BASE` mode, then:

- the home state is the `TestLogicReset` state
- the home instruction is defined as the `RESET` instruction (`BYPASS` or `IDCODE`)
- the home register is defined as the `RESET` register (`BYPASS` or `DEVICE_ID`)

If the test mode is the `TB_EXTEST_CAP_UPDT` mode, then:

  - the home state is the `CaptureDR` state
  - the home instruction is `EXTEST`
  - the home register is the boundary register

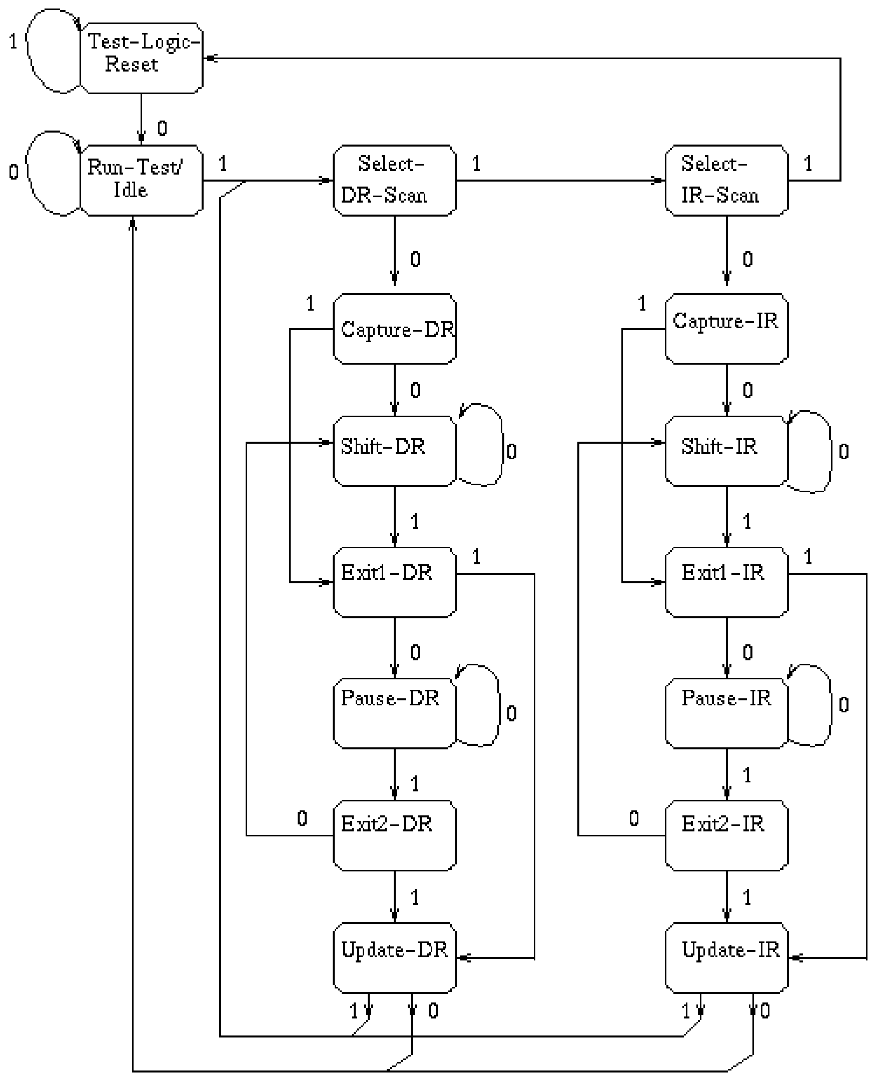
The *home state* describes what state the TAP FSM is in.

The *home instruction* describes what instruction is latched in the parallel instruction register update latches.

The *home register* is the register that is selected for scan.

Figure 2-1 on page 84 represents the 1149.1 Finite State Machine Diagram:

Figure 2-1 Finite State Machine Diagram



This figure is a representation of the state diagram for the 1149.1 TAP controller contained in IEEE standard 1149.1-1990, IEEE Standard Test Access Port and Boundary Scan Architecture.

The TAP controller is a synchronous finite state machine that responds to changes at the TMS and TCK inputs to the TAP to control the operation of the circuitry defined by the 1149.1 standard.

NOTES:

1. The value shown adjacent to each state transition in this figure represents the signal present on TMS at the time of a rising edge of TCK

2. All state transitions of the TAP controller occur based on the value of TMS at the time of a rising edge of TCK.

Mapping the Verification Suite to TBD Patterns

This section describes how the verification patterns are mapped to a TBD uncommitted test by BSV. BSV will produce at most two uncommitted tests for a complete verification. One experimental TBD contains the patterns for checking done in the `BASE` mode, and one experimental TBD contains the verification patterns for the checking done in the `TB_EXTEST_CAP_UPDT` mode. BSV maps the verification patterns to the TBD hierarchy as follows:

- `uncommitted test` - contains all verification patterns based on verification request applicable to the test mode to which they are applied
- `Test_Section` - contains verification patterns for a BSV defined group (e.g Bypass, iomappinginpin). These are referred to as BSV's AGNs.
- `Tester_Loop` - contains verification patterns for a BSV defined check (e.g. HighZ check). These are referred to as BSV's APNs.
- `Test_Procedure` - contains either the mode init sequence patterns or the sequences and patterns that make up the BSV check.

By default, the verification suite is constructed such that each `Tester_Loop` contains one BSV check so that all checks and groups of checks are independent of one another. The test mode's mode init `Test_Procedure` is used to initialize the design to a known state.

1149.1 Boundary Scan Verification Checks

This section describes the elements that comprise the 1149.1 BSV verification suite. The suite is produced from a set of predefined verification groups and procedures based on the verification request. The checks in the verification suite are customized based on the BSDL for the design being verified. Each verification group is assigned an AGN for referencing purposes. Likewise, each verification procedure is assigned an APN for referencing purposes. An APN constitutes one BSV check which verifies one or more 1149.1 Boundary Scan Rules defined in the IEEE 1149.1b-1994 and IEEE 1149. 2001 standards.

The BSDL `COMPONENT_CONFORMANCE` statement is referenced by `write_bsdl` and `verify_11491_boundary` to determine which level of the standard to use for compliance checking. The `conformance` is used by t`write_bsdl` to determine which level of the standard it will use when producing a BSDL file.

"1149.1 BSV Verification Groups (AGNs)" on page 88 lists the AGNs. "1149.1 BSV Verification Procedures (APNs)" on page 103 lists the APNs."IEEE 1149.1 BSDL Rules Verification" on page 144 and "IEEE 1149.1 Rules Verification Tables" on page 155 shows cross-reference of 1149.1 rules, verification procedures, semantic checks, and Encounter Test messages.

1149.1 BSV Verification Groups (AGNs)

This section lists the AGNs for Verify 1149.1 Boundary. This information is displayed as a function of help when displaying test descriptions using the *View Test Data* window. For related information, refer to "Viewing Vectors" in the *Encounter Test: Reference: GUI*.

AGN: 1 - Reset

Purpose: The reset group is a collection of tests which verify reset behaviors of the 1149.1 logic.

Explanation: Reset verification is done in the BASE test mode. Reset verification can be selected for verification by specifying `verify=reset` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 1 - Asynchronous Reset
2. APN: 2 - Synchronous Reset
3. APN: 3 - Test Logic Reset Zero Transition - Reset/0
4. APN: 4 - SelectIR One Transition - SelectIR/1

AGN: 2 - TAP FSM

Purpose: The TAP FSM group is a collection of tests which verifies that a TAP FSM behaves according to the Standard.

Explanation: TAP FSM verification is done in the BASE test mode. Verification requires the use of the `BYPASS` register. TAP FSM verification can be selected for verification by specifying `verify=tap` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 3 - Test Logic Reset Zero Transition - Reset/0
2. APN: 2 - Synchronous Reset
3. APN: 10 - Exit2IR One Transition - Exit2IR/1
4. APN: 4 - SelectIR One Transition - SelectIR/1
5. APN: 12 - Instruction Decode-Data Register Access

6. APN: 6 - Instruction Register Capture and Shift
7. APN: 8 - Instruction Register Capture Without Immediate Shift
8. APN: 7 - Instruction Shift without Capture
9. APN: 9 - PauseIR Zero Transition - PAUSEIR/0
10. APN: 11 - UpdateIR Zero Transition - UpdateIR/0
11. APN: 21 - Run Test Idle Zero Transition - RTI/0
12. APN: 19 - UpdateDR Zero Transition
13. APN: 14 - Data Register Capture and Shift
14. APN: 16 - Data Register Capture Without Immediate Shift
15. APN: 15 - Data Register Shift Without Capture
16. APN: 18 - Data Register Update
17. APN: 17 - PauseDR Zero Transition
18. APN: 20 - Exit2DR One Transition

AGN: 3 - Instruction Register

Purpose: The instruction register group is a collection of tests which verify whether the instruction register behaves according to the Standard.

Explanation: Instruction register verification is done in the BASE test mode. Verification requires the use of the `BYPASS` register. Instruction register verification can be selected for verification by specifying `verify=instreg` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 5 - Instruction Register Shift
2. APN: 7 - Instruction Shift without Capture
3. APN: 9 - PauseIR Zero Transition - PAUSEIR/0
4. APN: 6 - Instruction Register Capture and Shift
5. APN: 8 - Instruction Register Capture Without Immediate Shift
6. APN: 4 - SelectIR One Transition - SelectIR/1
7. APN: 10 - Exit2IR One Transition - Exit2IR/1

8. APN: 11 - UpdateIR Zero Transition - UpdateIR/0

AGN: 4 - Bypass

Purpose: The `BYPASS` group is a collection of tests which verifies the `BYPASS` instruction. Verification of the `BYPASS` instruction verifies whether the `BYPASS` register behaves according to the Standard when accessed by the `BYPASS` instruction.

Explanation: `BYPASS` verification is done in the BASE test mode. `BYPASS` verification can be selected for verification by specifying `verify=bypass` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 13 - Data Register Shift
2. APN: 13 - Data Register Shift
3. APN: 17 - PauseDR Zero Transition
4. APN: 14 - Data Register Capture and Shift
5. APN: 16 - Data Register Capture Without Immediate Shift
6. APN: 18 - Data Register Update
7. APN: 19 - UpdateDR Zero Transition
8. APN: 20 - Exit2DR One Transition
9. APN: 21 - Run Test Idle Zero Transition - RTI/0
10. APN: 12 - Instruction Decode-Data Register Access (any unexercised `BYPASS` op codes)

Note: These tests result in the traversing of all states and edges in the RTI and `SelectDR-UpdateDR` chain of states while the target data register is selected for scan.

AGN: 5 - IDCODE

Purpose: The `IDCODE` group is a collection of tests which verifies the `IDCODE` instruction. Verification of the `IDCODE` instruction verifies whether the `DEVICE_ID` register behaves according to the Standard when accessed by the `IDCODE` instruction.

Explanation: `IDCODE` verification is done in the BASE test mode. `IDCODE` verification can be selected for verification by specifying `verify=idcode` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 13 - Data Register Shift

2. APN: 15 - Data Register Shift Without Capture
3. APN: 17 - PauseDR Zero Transition
4. APN: 14 - Data Register Capture and Shift
5. APN: 16 - Data Register Capture Without Immediate Shift
6. APN: 18 - Data Register Update
7. APN: 19 - UpdateDR Zero Transition
8. APN: 20 - Exit2DR One Transition
9. APN: 21 - Run Test Idle Zero Transition - RTI/0
10. APN: 12 - Instruction Decode-Data Register Access (any unexercised `IDCODE` opcodes)

Note: These tests result in the traversing of all states and edges in the RTI and `SelectDR-UpdateDR` chain of states while the target data register is selected for scan.

AGN: 6 - USERCODE

Purpose: The `USERCODE` group is a collection of tests which verify the `USERCODE` instruction. Verification of the `USERCODE` instruction verifies whether the `DEVICE_ID` register behaves according to the Standard when accessed by the `USERCODE` instruction.

Explanation: `USERCODE` verification is done in the BASE test mode. `USERCODE` verification can be selected for verification by specifying `verify=usercode` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 13 - Data Register Shift
2. APN: 15 - Data Register Shift Without Capture
3. APN: 17 - PauseDR Zero Transition
4. APN: 14 - Data Register Capture and Shift
5. APN: 16 - Data Register Capture Without Immediate Shift
6. APN: 18 - Data Register Update
7. APN: 19 - UpdateDR Zero Transition
8. APN: 20 - Exit2DR One Transition
9. APN: 21 - Run Test Idle Zero Transition - RTI/0

10. APN: 12 - Instruction Decode-Data Register Access (any unexercised `USERCODE` opcodes)

Note: These tests result in the traversing of all states and edges in the RTI and `SelectDR-UpdateDR` chain of states while the target data register is selected for scan.

AGN: 7 - User Instruction

Purpose: The User Instruction group is a collection of tests which verify a user defined instruction. Verification of a user instruction verifies whether the register defined as being accessed by the user instruction behaves according to the Standard and its definition when accessed by the user instruction. The User Instruction group is used to verify public or private user instructions.

Explanation: `USERCODE` verification is done in the BASE test mode. User instruction verification can be selected for verification by specifying `verify= user instruction name` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 13 - Data Register Shift
2. APN: 15 - Data Register Shift Without Capture
3. APN: 17 - PauseDR Zero Transition
4. APN: 14 - Data Register Capture and Shift
5. APN: 16 - Data Register Capture Without Immediate Shift
6. APN: 18 - Data Register Update
7. APN: 19 - UpdateDR Zero Transition
8. APN: 20 - Exit2DR One Transition
9. APN: 21 - Run Test Idle Zero Transition - RTI/0
10. APN: 12 - Instruction Decode-Data Register Access (any unexercised User Instruction opcodes)

The User Instruction group is given a name which is the name given on the verify option. For example, if a user defined instruction is named `SETBIST` and a user requests `verify=setbist,` then the group of tests to verify the `SETBIST` instruction is called `SETBIST`.

Requesting verification of user defined instructions is based on information provided in the BSDL. BSV uses the instruction name given on the verify option to find the instruction and register information in the BSDL. For-user defined instructions, BSV expects to find the instruction name given on the *verify* option in the `INSTRUCTION_OPCODE` statement. To determine the register definition for a user-defined instruction, BSV expects to find the instruction name in the `REGISTER_ACCESS` statement wherein the register name, length and optional capture value is defined.

You can request verification of user defined instructions by providing an explicit list of instruction names on the verify option or by specifying `verify=public`, which verifies all public user defined instructions or by specifying `verify=private`, which verifies all private user-defined instruction. Any user-defined instruction defined under the `INSTRUCTION_PRIVATE` statement is considered a private instruction. To enable verification of private user instructions, a user will need to temporarily modify the BSDL to include the register accessed by the private instruction in the `REGISTER_ACCESS` statement. Once verification is complete, the private register can be removed from the `REGISTER_ACCESS` statement.

Note: These tests result in the traversing of all states and edges in the RTI and `SelectDR-UpdateDR` chain of states while the target data register is selected for scan.

AGN: 8 - HIGHZ

Purpose: The HIGHZ group is a collection of tests which verify whether the HIGHZ instruction behaves according to the Standard.

Explanation: HIGHZ verification is done in the BASE test mode. HIGHZ verification can be selected for verification by specifying `verify=highz` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 13 - Data Register Shift
2. APN: 15 - Data Register Shift Without Capture
3. APN: 17 - PauseDR Zero Transition
4. APN: 14 - Data Register Capture and Shift
5. APN: 16 - Data Register Capture Without Immediate Shift
6. APN: 18 - Data Register Update
7. APN: 19 - UpdateDR Zero Transition
8. APN: 29 - Boundary Register - All System Output Drivers Retain Active State
9. APN: 21 - Run Test Idle Zero Transition - RTI/0
10. APN: 41 - All System Output Pins Inactive

11. APN: 12 - Instruction Decode-Data Register Access (any unexercised HIGHZ opcodes)

Note: These tests result in the traversing of all states and edges in the RTI and `SelectDR-UpdateDR` chain of states while the target data register is selected for scan.

AGN: 9 - CLAMP

Purpose: The `CLAMP` group is a collection of tests which verify whether the `CLAMP` instruction behaves according to the Standard.

Explanation: `CLAMP` verification is done in the BASE test mode. `CLAMP` verification requires use of the `SAMPLE`/`PRELOAD` instruction to precondition the output and control cells to drive a '`clamp`' state when the `CLAMP` instruction becomes active. `CLAMP` verification can be selected for verification by specifying `verify=clamp` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 13 - Data Register Shift
2. APN: 15 - Data Register Shift Without Capture
3. APN: 17 - PauseDR Zero Transition
4. APN: 14 - Data Register Capture and Shift
5. APN: 16 - Data Register Capture Without Immediate Shift
6. APN: 18 - Data Register Update
7. APN: 19 - UpdateDR Zero Transition
8. APN: 20 - Exit2DR One Transition
9. APN: 21 - Run Test Idle Zero Transition - RTI/0
10. APN: 12 - Instruction Decode-Data Register Access (any unexercised `CLAMP` opcodes)

Note: These tests result in the traversing of all states and edges in the RTI and `SelectDR-UpdateDR` chain of states while the target data register is selected for scan.

AGN: 10 - SAMPLE/PRELOAD

Purpose: The `SAMPLE`/`PRELOAD` group is a collection of tests which verify whether the `SAMPLE`/`PRELOAD` instruction behaves according to the Standard.

Explanation: `SAMPLE`/`PRELOAD` verification is done in the BASE test mode. `SAMPLE`/`PRELOAD` verification requires use of the EXTEST instruction to verify the `SAMPLE` and `PRELOAD` phases works as expected. `SAMPLE`/`PRELOAD` verification can be selected for verification by specifying `verify=sample` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 13 - Data Register Shift
2. APN: 15 - Data Register Shift Without Capture
3. APN: 17 - PauseDR Zero Transition
4. APN: 14 - Data Register Capture and Shift
5. APN: 16 - Data Register Capture Without Immediate Shift
6. APN: 18 - Data Register Update
7. APN: 19 - UpdateDR Zero Transition
8. APN: 20 - Exit2DR One Transition
9. APN: 21 - Run Test Idle Zero Transition - RTI/0
10. APN: 23 - SAMPLE
11. APN: 22 - PRELOAD
12. APN: 12 - Instruction Decode-Data Register Access (any unexercised `SAMPLE`/`PRELOAD` opcodes)

Note: These tests result in the traversing of all states and edges in the RTI and `SelectDR-UpdateDR` chain of states while the target data register is selected for scan.

AGN: 11 - EXTEST

Purpose: The `EXTEST` group is a collection of tests which support the verification of the EXTEST instruction. `EXTEST` verification verifies whether certain aspects of the boundary register behaves according to the Standard when accessed by the `EXTEST` instruction. The primary intent of this group of tests is to verify that the boundary register is properly accessed (shifted) across the data register TAP states and by the various `EXTEST` opcodes. Verification of the correspondence between the boundary register cells and device I/Os is verified by the `IOMapping` groups (AGN: 13 and AGN: 14).

Explanation: `EXTEST` verification is done in the BASE test mode. `EXTEST` verification can be selected by specifying `verify=extest` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 13 - Data Register Shift
2. APN: 15 - Data Register Shift Without Capture

3. APN: 17 - PauseDR Zero Transition
4. APN: 14 - Data Register Capture and Shift
5. APN: 16 - Data Register Capture Without Immediate Shift
6. APN: 18 - Data Register Update
7. APN: 19 - UpdateDR Zero Transition
8. APN: 20 - Exit2DR One Transition
9. APN: 21 - Run Test Idle Zero Transition - RTI/0
10. APN: 12 - Instruction Decode-Data Register Access (any unexercised EXTEST opcodes)

Note: These tests result in the traversing of all states and edges in the RTI and `SelectDR-UpdateDR` chain of states while the target data register is selected for scan.

AGN: 12 - RUNBIST

Purpose: The `RUNBIST` group is a collection of tests which verify whether the `RUNBIST` instruction behaves according to the Standard. Verification of `RUNBIST` is limited to data register access and treatment of the package pins when the `RUNBIST` instruction is active.

Explanation: `RUNBIST` verification is done in the BASE test mode. `RUNBIST` verification can be selected for verification by specifying `verify=runbist` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 13 - Data Register Shift
2. APN: 15 - Data Register Shift Without Capture
3. APN: 17 - PauseDR Zero Transition
4. APN: 14 - Data Register Capture and Shift
5. APN: 16 - Data Register Capture Without Immediate Shift
6. APN: 18 - Data Register Update
7. APN: 19 - UpdateDR Zero Transition
8. APN: 20 - Exit2DR One Transition
9. APN: 21 - Run Test Idle Zero Transition - RTI/0

10. APN: 43 - RUNBIST - All System Output Pins Inactive or APN: 44 - RUNBIST - All System Output Pins Active
11. APN: 12 - Instruction Decode-Data Register Access (any unexercised RUNBIST opcodes)

The `RUNBIST_EXECUTION` statement defines the behavior of the system output pins when the `RUNBIST` instruction is active. If the `RUNBIST_EXECUTION` statement defines the system output pins to go to the state defined by the state of the boundary register, then verification of the `RUNBIST` instruction requires use of the `BOUNDARY_REGISTER` statement and the `SAMPLE`/`PRELOAD` instruction.

Note: These tests result in the traversing of all states and edges in the RTI and `SelectDR-UpdateDR` chain of states while the target data register is selected for scan.

AGN: 13 - Boundary Register - I/O Mapping System Output Pins

Purpose: The Boundary Register - I/O Mapping System Output Pins group is a collection of tests which verify the functionality of all system output pins when the `EXTEST` instruction is active. This collection of tests is part of the verification for the `EXTEST` instruction. This group of tests verifies each output cell and each control cell both individually and collectively as a group. Boundary register accessing and scanning when the `EXTEST` instruction is active is verified via the `EXTEST` group. System input pins are verified via Boundary Register - I/O Mapping System Input Pins.

Explanation: Verification of output and control cells defined for system output pins is done in the EXTEST (`TB_EXTEST_CAP_UPDT`) test mode. Output pin verification can be selected by specifying `verify=iomappingoutpin` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 25 - Boundary Register I/O Mapping - System Output Pin (for each output capable pin)
2. APN: 26 - Boundary Register I/O Mapping - Share Enable (for each share enable pin)
3. APN: 31 - Boundary Register I/O Mapping - Share Disable (for each share cell)
4. APN: 27 - Boundary Register - Enable All System Output Drivers (Logic 0)
5. APN: 27 - Boundary Register - Enable All System Output Drivers(Logic 1)
6. APN: 29 - Boundary Register - All System Output Drivers Retain Active State
7. APN: 28 - Boundary Register - Disable All System Output Drivers

The `BOUNDARY_REGISTER` statement defines the mapping of boundary register cells to system pins. BSV uses the `BOUNDARY_REGISTER` statement to which pins are considered system output pins. Any pin defined with a cell function of `OUTPUT2`, `OUTPUT3` or `BIDIR` is considered a system output pin. The `BOUNDARY_REGISTER` statement also defines the enable/disable properties for output drivers with an inactive state. This group of tests also verifies each output pin can be driven to each of its active and inactive states as appropriate. A `CONTROL`/`CONTROLR` cell which controls more than one output pin is considered a shared enable. This group of tests also verifies that multiple system output pins can be driven to active and inactive states.

AGN: 14 - Boundary Register - I/O Mapping System Input Pins

Purpose: The Boundary Register - I/O Mapping System Input Pins group is a collection of tests which verify the functionality of all system input pins when the `EXTEST` instruction is active. This collection of tests is part of the verification for the `EXTEST` instruction. This group of tests verifies each input capable cell. Boundary register accessing and scanning when the `EXTEST` instruction is active is verified via the `EXTEST` group. System output pins are verified via Boundary Register - I/O Mapping System Output Pins.

Explanation: Verification of input capable (observe) cells defined for system input pins is done in the EXTEST (`TB_EXTEST_CAP_UPDT`) test mode. Input pin verification can be selected by specifying `verify=iomappinginpin` on the BSV command line interface. Verification is accomplished by the ordered application of the following:

1. APN: 24 - Boundary Register I/O Mapping - System Input Pin (for each input capable pin)
2. APN: 40 - Capture All System Input Pins (Logic 0)
3. APN: 40 - Capture All System Input Pins (Logic 1)
4. APN: 39 - Boundary Register - BiDir IOWRAP (for all bidirectional pins)

The `BOUNDARY_REGISTER` statement defines the mapping of boundary register cells to system pins. BSV uses the `BOUNDARY_REGISTER` statement to which pins are considered system input pins. Any pin defined with a cell function of `INPUT`, `BIDIR`, `CLOCK`, or `OBSERVE_ONLY` is considered a system input pin. The `BOUNDARY_REGISTER` statement also defines the enable/disable properties for bidirectional pins. These tests use the `BOUNDARY_REGISTER` disable specification to disable a driver on a bidirectional system pin.

AGN: 15 - SAMPLE

Purpose: The `SAMPLE` group is a collection of tests to verify whether the `SAMPLE` instruction behaves as expected.

Explanation: The `SAMPLE` verification is done in the BASE mode by using the EXTEST instruction. Select this collection of tests by specifying `verify=sample` on the BSV command interface. Verification is accomplished by the ordered application of the following:

1. APN: 13 - Data Register Shift
2. APN: 15 - Data Register Shift Without Capture
3. APN: 17 - PauseDR Zero Transition
4. APN: 14 - Data Register Capture and Shift
5. APN: 16 - Data Register Capture Without Immediate Shiftt
6. APN: 18 - Data Register Update
7. APN: 19 - UpdateDR Zero Transition
8. APN: 20 - Exit2DR One Transition
9. APN: 21 - Run Test Idle Zero Transition - RTI/0
10. APN: 23 - SAMPLE
11. APN: 12 - Instruction Decode-Data Register Access)

Note: These tests result in the traversing of all states and edges in the `RTI` and `SelectDR-UpdateDR` chain of states while the target data register is selected for scan.

AGN: 16 - PRELOAD

Purpose: The `PRELOAD` group is a collection of tests to verify whether the `PRELOAD` instruction behaves as expected.

Explanation: The `PRELOAD` verification is done in the BASE mode by using the EXTEST instruction. Select this collection of tests can by specifying `verify=preload` on the BSV command interface. Verification is accomplished by the ordered application of the following:

1. APN: 13 - Data Register Shift
2. APN: 15 - Data Register Shift Without Capture
3. APN: 17 - PauseDR Zero Transition
4. APN: 14 - Data Register Capture and Shift
5. APN: 16 - Data Register Capture Without Immediate Shift
6. APN: 18 - Data Register Update

7. APN: 19 - UpdateDR Zero Transition
8. APN: 20 - Exit2DR One Transition
9. APN: 21 - Run Test Idle Zero Transition - RTI/0
10. APN: 22 - PRELOAD
11. APN: 12 - Instruction Decode-Data Register Access

Note: These tests result in the traversing of all states and edges in the `RTI` and `SelectDR-UpdateDR` chain of states while the target data register is selected for scan.

AGN: 17 - EXTEST_PULSE

Purpose: The `EXTEST_PULSE` group is a collection of tests to verify whether certain aspects of the boundary register behave as expected when accessed by the `EXTEST_PULSE` instruction. The primary intent of the `EXTEST_PULSE` test group is to verify that the boundary register is properly accessed (shifted) across the data register TAP states and by the various `EXTEST_PULSE` opcodes. The correspondence between the boundary register cells and device I/Os is verified by the IOMapping groups (AGN: 13 and AGN: 14).

AGN 17 checks for conformance to the IEEE 1149.6 standard.

Explanation: The `EXTEST_PULSE` verification is done in the BASE mode and is selected by specifying `verify=extest_pulse` on the BSV command interface. Verification is accomplished by the ordered application of the following:

1. APN: 13 - Data Register Shift
2. APN: 15 - Data Register Shift Without Capture
3. APN: 17 - PauseDR Zero Transition
4. APN: 14 - Data Register Capture and Shift
5. APN: 16 - Data Register Capture Without Immediate Shift
6. APN: 18 - Data Register Update
7. APN: 19 - UpdateDR Zero Transition
8. APN: 20 - Exit2DR One Transition
9. APN: 21 - Run Test Idle Zero Transition - RTI/0
10. APN: 12 - Instruction Decode-Data Register Access)

Note: These tests result in the traversing of all states and edges in the `RTI` and `SelectDR-UpdateDR` chain of states while the target data register is selected for scan.

AGN: 18 - EXTEST_TRAIN

Purpose: The `EXTEST_TRAIN` group is a collection of tests to verify whether certain aspects of the boundary register behave as expected when accessed by the `EXTEST_TRAIN` instruction. The primary intent of this group of tests is to verify that the boundary register is properly accessed (shifted) across the data register TAP states and by the various `EXTEST_TRAIN` opcodes. The correspondence between the boundary register cells and device I/Os is verified by the IOMapping groups (AGN: 13 and AGN: 14).

AGN 18 checks for conformance to the IEEE 1149.6 standard.

Explanation: The `EXTEST_TRAIN` verification is done in the BASE test t mode and can be selected by specifying `verify=extest_train` on the BSV command interface. Verification is accomplished by the ordered application of the following:

1. APN: 13 - Data Register Shift
2. APN: 15 - Data Register Shift Without Capture
3. APN: 17 - PauseDR Zero Transition
4. APN: 14 - Data Register Capture and Shift
5. APN: 16 - Data Register Capture Without Immediate Shift
6. APN: 18 - Data Register Update
7. APN: 19 - UpdateDR Zero Transition
8. APN: 20 - Exit2DR One Transition
9. APN: 21 - Run Test Idle Zero Transition - RTI/0
10. APN: 12 - Instruction Decode-Data Register Access)

Note: These tests result in the traversing of all states and edges in the RTI and SelectDR-UpdateDR chain of states while the target data register is selected for scan.

AGN: 19 - Boundary Register - I/O Mapping System Output Pins

Purpose: This is the 1149.6 equivalent for the system output pins group of tests. This collection of tests uses the `EXTEST_PULSE` instruction. Verification is done in the

`EXTEST_PULSE (TB_EXTEST_PULSE_CAP_UPDT)` test mode by the ordered application of the following:

1. APN: 32 - Boundary Register I/O Mapping - System Output Pin
2. APN: 33 - Boundary Register I/O Mapping - Share Enable
3. APN: 34 - Boundary Register I/O Mapping - Share Disable
4. APN: 35 - Boundary Register I/O Mapping - Enable All System Output Drivers (Logic 0)
5. APN: 35 - Boundary Register I/O Mapping - Enable All System Output Drivers (Logic 1)
6. APN: 36 - Boundary Register I/O Mapping - All System Output Drivers Retain Active State
7. APN: 37 - Boundary Register I/O Mapping - Disable All System Output Drivers

Refer to AGN: 13 - Boundary Register - I/O Mapping System Output Pins on page 97 for more information.

AGN: 20 - Boundary Register - I/O Mapping System Input Pins

Purpose: This is the 1149.6 equivalent for the system input pins group of tests. This collection is part of the verification for the `EXTEST_PULSE` instruction. Verification is done in the `EXTEST_PULSE (TB_EXTEST_PULSE_CAP_UPDT)` test mode by the ordered application of the following:

1. APN: 48 - Boundary Register I/O Mapping - System Input Pin
2. APN: 50 - Capture All System Input Pins (Logic 0)
3. APN: 50 - Capture All System Input Pins (Logic 1)
4. APN: 49 - Boundary Register BiDir IOWRAP

Refer to AGN: 14 - Boundary Register - I/O Mapping System Input Pins on page 98 for more information.

AGN: 21- Boundary Register - I/O Mapping System Output Pins

Purpose: This is the 1149.6 equivalent for the system output pins group of tests. This collection of tests uses the `EXTEST_TRAIN` instruction. Verification is done in the `EXTEST_TRAIN (TB_EXTEST_TRAIN_CAP_UPDT)` test mode by the ordered application of the following:

1. APN: 52 - Boundary Register I/O Mapping - System Output Pin

2. APN: 53 - Boundary Register I/O Mapping - Share Enable
3. APN: 54 - Boundary Register I/O Mapping - Share Disable
4. APN: 55 - Boundary Register I/O Mapping - Enable All System Output Drivers (Logic 0)
5. APN: 55 - Boundary Register I/O Mapping - Enable All System Output Drivers (Logic 1)
6. APN: 56 - Boundary Register I/O Mapping - All System Output Drivers Retain Active State
7. APN: 57 - Boundary Register I/O Mapping - Disable All System Output Drivers

Refer to AGN: 13 - Boundary Register - I/O Mapping System Output Pins on page 97for more information

AGN: 22- Boundary Register - I/O Mapping System Input Pins

Purpose: This is the 1149.6 equivalent for the system input pins group of tests. This collection is part of the verification for the `EXTEST_TRAIN` instruction. Verification is done in the `EXTEST_TRAIN (TB_EXTEST_TRAIN_CAP_UPDT)` test mode by the ordered application of the following:

1. APN: 58- Boundary Register I/O Mapping - System Input Pin
2. APN: 60- Capture All System Input Pins (Logic 0)
3. APN: 60- Capture All System Input Pins (Logic 1)
4. APN: 59 - Boundary Register BiDir IOWRAP

Refer to AGN: 14 - Boundary Register - I/O Mapping System Input Pins on page 98 for more information

1149.1 BSV Verification Procedures (APNs)

This section lists the Absolute Test Procedure Numbers (APNs) for Verify 1149.1 Boundary. Complete detail of each APN is displayed as a function of help when displaying test descriptions using the *View Test Data* window. For related information, refer to “Viewing Vectors” in the *Encounter Test: Reference: GUI*.

APN: 1 - Asynchronous Reset

Purpose: The asynchronous reset test verifies that designs with a TRST* pin can be asynchronously reset to the TLR state. This check is done in the BASE mode.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load the instruction register with an instruction other than the home instruction. This step is omitted if there is no other instruction defined.
2. Set TRST* to zero with the assumption that the TAP will go to the TLR state.
3. Performing a `CaptureDR` and `ShiftDR` on the data register selected for scan upon entering the TLR state. Expect appropriate capture and shift behavior on data register selected for scan.
4. Return to the home state.

The data register selected for scan must be either the `BYPASS` register or the 32-bit DEVICE IDENTIFICATION register. BSV uses its input BSDL to determine which register should be selected for scan. If an `IDCODE` instruction is defined in the BSDL, then BSV expects that the DEVICE IDENTIFICATION register will be selected for scan, otherwise BSV expects that the `BYPASS` register will be selected for scan. If the DEVICE IDENTIFICATION register is selected for scan, then BSV will expect the `IDCODE_REGISTER` value is captured into the DEVICE IDENTIFICATION register and is subsequently scanned out via a `CaptureDR` / `ShiftDR` on the selected register. If the `BYPASS` register is selected for scan, then BSV will expect that a capture value of one is captured into the `BYPASS` register and is subsequently scanned out via a `CaptureDR`/`ShiftDR` on the selected register. In both cases, the register is over-shifted with a shift signature of five bits (10011) which results in a shift length of six bits or 37 bits for the `BYPASS` and `DEVICE_IDENTIFICATION` registers respectively.

Rules Supported: 3.6.1(a), 3.8.1(a), 3.8.1(b), 5.2.1(a.ii), 5.3.1(c), 6.1.1(b), 6.2.1(b), 6.2.1(f), 7.2.1(c), 7.12.1(a).

Messages Produced: TJC-217

APN: 2 - Synchronous Reset

Purpose: The synchronous reset test verifies that designs can be synchronously reset (i.e. clocked) to the TLR state. This test also verifies the one transition out of the `TestLogicReset` (TLR) state of the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load the instruction register with an instruction other than the home instruction. This step is omitted if there is no other instruction defined.
2. Traverse to the TLR state and apply the one transition out of the TLR state several times. This step will result in at least five TCK pulses with TMS at one.
3. Perform a `CaptureDR` and `ShiftDR` on the data register selected for scan upon entering the TLR state. Expect appropriate capture and shift behavior on data register selected for scan.
4. Return to the home state.

The data register selected for scan must be either the `BYPASS` register or the 32-bit DEVICE IDENTIFICATION register. BSV uses its input BSDL to determine which register should be selected for scan. If an `IDCODE` instruction is defined in the BSDL, then BSV expects that the DEVICE IDENTIFICATION register will be selected for scan, otherwise BSV expects that the `BYPASS` register will be selected for scan. If the DEVICE IDENTIFICATION register is selected for scan, then BSV will expect the `IDCODE_REGISTER` value is captured into the DEVICE IDENTIFICATION register and is subsequently scanned out via a `CaptureDR` / `ShiftDR` on the selected register. If the `BYPASS` register is selected for scan, then BSV will expect that a capture value of one is captured into the `BYPASS` register and is subsequently scanned out via a `CaptureDR` / `ShiftDR` on the selected register. In both cases, the register is over-shifted with a shift signature of five bits (10011) which results in a shift length of six bits or 37 bits for the `BYPASS` and `DEVICE_IDENTIFICATION` registers respectively.

Rules Supported: 3.8.1(a), 3.8.1(b), 5.1.1(a) - TLR/1, 6.1.1(b), 6.2.1(b), 6.2.1(e), 7.2.1(c), 7.12.1(a).

Messages Produced: TJC-210, TJC-217

APN: 3 - Test Logic Reset Zero Transition - Reset/0

Purpose: The reset zero transition test verifies the zero transition out of the `TestLogicReset` (TLR) state in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Traverse to the TAP Test Logic Reset state and apply target (zero) transition
2. Perform a `CaptureDR` and `ShiftDR` on the data register selected for scan upon entering the TLR state. Expect appropriate capture and shift behavior on data register selected for scan.
3. Return to the home state.

The data register selected for scan must be either the `BYPASS` register or the 32-bit DEVICE IDENTIFICATION register. BSV uses its input BSDL to determine which register should be selected for scan. If an `IDCODE` instruction is defined in the BSDL, then BSV expects that the DEVICE IDENTIFICATION register will be selected for scan, otherwise BSV expects that the `BYPASS` register will be selected for scan. If the DEVICE IDENTIFICATION register is selected for scan, then BSV will expect the `IDCODE_REGISTER` value is captured into the DEVICE IDENTIFICATION register and is subsequently scanned out via a `CaptureDR` / `ShiftDR` on the selected register. If the `BYPASS` register is selected for scan, then BSV will expect that a capture value of one is captured into the `BYPASS` register and is subsequently scanned out via a `CaptureDR` / `ShiftDR` on the selected register. In both cases, the register is over-shifted with a shift signature of five bits (10011) which results in a shift length of six bits or 37 bits for the `BYPASS` and `DEVICE_IDENTIFICATION` registers respectively.

Rules Supported: 5.1.1(a) - TLR/0

Messages Produced: TJC-210, TJC-217

APN: 4 - SelectIR One Transition - SelectIR/1

Purpose: The `SelectIR` one transition test verifies the one transition out of the `SelectIR` state in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Traverse to the TAP `SelectIR` state and apply target (one) transition

2. Capture and shift the data register selected for scan upon entering the `TLRstate`. Expect appropriate capture and shift behavior on data register selected for scan.
3. Return to the home state.

The data register selected for scan must be either the `BYPASS` register or the 32-bit DEVICE IDENTIFICATION register. BSV uses its input BSDL to determine which register should be selected for scan. If an `IDCODE` instruction is defined in the BSDL, then BSV expects that the DEVICE IDENTIFICATION register will be selected for scan, otherwise BSV expects that the `BYPASS` register will be selected for scan. If the DEVICE IDENTIFICATION register is selected for scan, then BSV will expect the `IDCODE_REGISTER` value is captured into the DEVICE IDENTIFICATION register and is subsequently scanned out via a `CaptureDR` / `ShiftDR` on the selected register. If the `BYPASS` register is selected for scan, then BSV will expect that a capture value of one is captured into the `BYPASS` register and is subsequently scanned out via a `CaptureDR` / `ShiftDR` on the selected register. In both cases, the register is over-shifted with a shift signature of five bits (10011) which results in a shift length of six bits or 37 bits for the `BYPASS` and `DEVICE_IDENTIFICATION` registers respectively.

Notes

This check is performed in the `BASE` test mode. Therefore the home state is TLR and the home instruction is based on the presence or absence of the `IDCODE` instruction.

If the `IDCODE` instruction is defined, then home instruction is `IDCODE`.

If the `IDCODE` instruction is not defined then the home instruction is `BYPASS`.

Like all procedures, this procedure begins with the assumption that the design is in the BSV home state. This test is a Unique Input Output (UIO)-based test.

Rules Supported: 5.1.1(a) - `SelectIR`/1, 6.2.1(a), 6.2.1(b)

Messages Produced: TJC-210, TJC-217

APN: 5 - Instruction Register Shift

Purpose: The instruction register shift test verifies that the instruction register can be scanned without inversion and is the appropriate length.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Capture and shift the instruction register. Over shift instruction register with shift signature. CaptureIR behavior is not verified by this test.
2. Expect appropriate shift behavior on the instruction register.
3. Return to home state.

   BSV over-shifts the instruction register with a shift signature of five bits (10011). The `INSTRUCTION_LENGTH` statement defines the length of the instruction register. BSV expects the five bit signature to appear at TDO after the appropriate number of shift clock cycles.

Rules Supported: 6.1.1(a), 6.1.1(c)

Messages Produced: TJC-215, TJ-C217

APN: 6 - Instruction Register Capture and Shift

Purpose: The instruction register capture and shift test verifies the `CaptureIR` - `ShiftIR` behavior of the instruction register. This test also verifies the one transition out of the `SelectDR` state and the zero transitions out of the `SelectIR` and `CaptureIR` states in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Capture and shift the instruction register. Over shift the instruction register with shift signature.
2. Expect appropriate instruction register capture-shift behavior.
3. Return to the home state

This test verifies the `CaptureIR` - `ShiftIR` behavior of the instruction register. The `INSTRUCTION_CAPTURE` value defines the expected instruction register capture state. The

`INSTRUCTION_LENGTH` statement defines the length of the instruction register. BSV over-shifts the instruction register with a shift signature of five bits (10011). BSV expects the five bit signature to appear at TDO after the appropriate number of shift clock cycles for the instruction register.

Rules Supported: 5.1.1(a) - `SelectIR`/0, `CaptureIR`/0, `SelectDR`/1, 6.1.1(d), 6.2.1(a), 6.2.1(c)

Messages Produced: TJC-206, TJC-217

APN: 7 - Instruction Shift without Capture

Purpose: The instruction register shift without capture test verifies that the instruction register can be scanned without being preceded by a `CaptureIR`. This test also verifies the following transitions in the TAP FSM: the zero transition out of the `Exit1IR`, the zero and one transitions out of the `ShiftIR` state, the one transition out of the `PauseIR` state and the zero transition out of the `Exit2IR` state.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Capture and shift the instruction register. Load instruction register with a known state. `CaptureIR` behavior is not verified.
2. Traverse to the `PauseIR`, `Exit2IR` states following instruction register load
3. Load instruction register with five bit shift signature. Unload instruction register and expect previously loaded known state followed by five bit shift signature.
4. Return to home state

This test performs two back-to-back scans on the instruction register. The `INSTRUCTION_LENGTH` statement defines the length of the instruction register. BSV over-shifts the instruction register with a shift signature of five bits (10011). BSV expects the five bit signature to appear at TDO after the appropriate number of shift clock cycles for the instruction register. The second scan of the instruction register should result in a scan out value of the previous state of the instruction register followed by the five bit shift signature.

Rules Supported: 5.1.1(a) - `Exit1IR`/0, `ShiftIR`/0, `ShiftIR`/1, `PauseIR`/1, `Exit2IR`/0, 6.2.1(a)

Messages Produced: TJC-207, TJC-217

APN: 8 - Instruction Register Capture Without Immediate Shift

Purpose: The instruction register capture without shift test verifies the `CaptureIR` behavior of the instruction register when `ShiftIR` is not immediately preceded by a `CaptureIR`. This test also verifies the one transition out of the `CaptureIR` state in the TAP FSM

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Traverse from the home state (TLR) to `CaptureIR` and from `CaptureIR` to `PauseIR` bypassing `ShiftIR`.
2. Traverse from `PauseIR` to `ShiftIR`. Load instruction register with five bit shift signature. Unload instruction register and expect the `CaptureIR` state followed by five bit shift signature.
3. Return to home state

BSV uses this test to verify the capture behavior of the instruction register. BSV will expect the instruction register capture value is captured into the instruction register and is subsequently scanned out via `CaptureIR` - `Exit1IR` - `PauseIR` - `Exit2IR` - `ShiftIR` path through the TAP. The `INSTRUCTION_CAPTURE` statement defines the `CaptureIR` value. The `INSTRUCTION_LENGTH` statement defines the length of the instruction register. BSV over-shifts the instruction register with a shift signature of five bits (10011) and expects the five bit signature to appear at TDO after the appropriate number of shift clock cycles for the instruction register.

Rule Supported: 5.1.1(a) - `CaptureIR`/1

Messages Produced: TJC-216, TJC-217

APN: 9 - PauseIR Zero Transition - PAUSEIR/0

Purpose: The `PauseIR` zero transition test verifies the zero transition out of the `PauseIR` state in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Capture and shift the instruction register. Load instruction register with a known state. `CaptureIR` behavior is not verified.
2. Traverse to the `PauseIR` state following instruction register load.
3. Traverse twice on the `PauseIR`/0 transition
4. Load instruction register with five bit shift signature. Unload instruction register and expect previously loaded known state followed by five bit shift signature.
5. Return to home state

This test performs two back-to-back scans on the instruction register. The `INSTRUCTION_LENGTH` statement defines the length of the instruction register. BSV over-shifts the instruction register with a shift signature of five bits (10011). BSV expects the five bit signature to appear at TDO after the appropriate number of shift clock cycles for the instruction register.

Rule Supported: 5.1.1(a) - `PauseIR`/0

Messages Produced: TJC-207, TJC-217

APN: 10 - Exit2IR One Transition - Exit2IR/1

Purpose: The `Exit2IR` one transition test verifies the one transition out of the `Exit2IR` state in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load instruction register with an instruction which does not select a one bit register. If home instruction is `IDCODE` or no suitable instruction is defined, this step is omitted.
2. Capture and shift the instruction register. Load instruction register with the `BYPASS` instruction. `CaptureIR` behavior is not verified.

3. Traverse to the `UpdateIR` via the `Exit2IR` state
4. Capture and shift the `BYPASS` register. Overshift the `BYPASS` register with the shift signature.
5. Expect appropriate capture-shift behavior on scanout of the `BYPASS` register
6. Return to home state.

This test verifies the one transition out of the `Exit2IR` state. Verification is accomplished by using the `BYPASS` instruction and `BYPASS` register. The `INSTRUCTION_LENGTH` statement defines the length of the instruction register. BSV over-shifts the `BYPASS` register with a shift signature of five bits (10011) and expects the five bit signature to appear at TDO after the appropriate number of shift clock cycles for the register being scanned. BSV expects the first bit scanned out of the `BYPASS` register is one followed by the five bit shift signature.

Rules Supported: 5.1.1(a) - `Exit2IR`/1, 6.1.1(b), 6.2.1(d)

Messages Produced: TJC-209, TJC-217

APN: 11 - UpdateIR Zero Transition - UpdateIR/0

Purpose: The `UpdateIR` zero transition test verifies the zero transition out of the `UpdateIR` state in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load instruction register with an instruction which does not select a one bit register. If home instruction is `IDCODE` or no suitable instruction is defined, this step is omitted.
2. Capture and shift the instruction register. Load instruction register with the `BYPASS` instruction. `CaptureIR` behavior is not verified.
3. Traverse to the target state (`UpdateIR`) via the `Exit1IR` state and apply the target transition (zero).
4. Capture and shift the `BYPASS` register. Overshift the `BYPASS` register with the shift signature.
5. Expect the `BYPASS` register capture value (1), followed by the five bit shift signature on scan out of the `BYPASS` register.

6. Return to home state.

This test verifies the zero transition out of the `UpdateIR` state. Verification is accomplished by using the `BYPASS` instruction and `BYPASS` register. The `INSTRUCTION_LENGTH` statement defines the length of the instruction register. BSV over-shifts the instruction register and the `BYPASS` register with a shift signature of five bits (10011) and expects the five it signature to appear at TDO after the appropriate number of shift clock cycles for the register being scanned.

Rule Supported: 5.1.1(a) - `UpdateIR`/0

Messages Produced: TJC-209, TJC-217

APN: 12 - Instruction Decode-Data Register Access

Purpose: The instruction decode data register access test verifies that an instruction decode selects the appropriate register for scan. This test also verifies the one transition out of `Exit1IR` and the one transition out of `UpdateIR` in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Capture and shift instruction register. Load instruction register with an instruction accessing a register other than the target register. `CaptureIR` behavior is not verified.
2. Capture and shift the instruction register. Load instruction register with the target instruction opcode by traversing to `UpdateIR` via `Exit1IR`. `CaptureIR` behavior is not verified.
3. Traverse to `CaptureDR` via the one transition out of `UpdateIR`
4. Capture and shift the data register selected for scan. Overshift data register with shift signature.
5. Expect appropriate capture-shift behavior on data register selected for scan. `CaptureDR` behavior not verified for boundary register.
6. Return to home state

   For standard instructions: Standard instructions are expected to access the appropriate standard register. In particular, the `BYPASS`, `HIGHZ`, and `CLAMP` instructions are expected to access the one bit `BYPASS` register. The `SAMPLE`/`PRELOAD`, and `EXTEST` instructions are expected to access the `BOUNDARY` register whose length is defined by the BSDL attribute `BOUNDARY_LENGTH`. The `IDCODE` and `USERCODE` instructions are expected to access the 32-bit `DEVICE_IDENTIFICATION` register. When the `BYPASS` register should be selected for scan, a `CaptureDR` values of one (1) is expected. When the `DEVICE_IDENTIFICATION` register should be selected for scan via the `IDCODE` instruction, then the `IDCODE` defines the expected `CaptureDR` value. When the `DEVICE_IDENTIFICATION` register should be selected for scan via the `USERCODE` instruction, then the `USER_CODE` defines the expected `CaptureDR` value. Boundary register `CaptureDR` behavior is verified by other checks.

For non-standard instructions: BSV uses the `INSTRUCTION` name and `REGISTER_ACCESS` statement to determine which data register should be selected for scan. The `REGISTER_ACCESS` defines the register length and which instructions may access it. The `REGISTER_ACCESS` may also define a '`captures`' value for certain instruction opcodes. If there '`capture`' value is defined for an opcode and it contains at least one known state, then it will be uses as an expected `CaptureDR` value in this test.

In all cases, BSV uses a five bit shift signature of 10011 to verify that the target register has been selected for scan. This results in a five bit over-shift of the target register. For example, BSV will expect the five bit signature to appear on TDO after 32 TCK clock cycles when verifying an instruction which accesses the standard `DEVICE_IDENTIFICATION` register.

Rules Supported: 5.1.1(a) - `Exit1IR`/1, `UpdateIR`/1, 6.1.1(a), 6.1.1(b), 6.2.1(a), 6.2.1(b), 7.7.1(c), 7.4.1(c), 7.10.1(a), 7.12.1(b), 7.12.1(c), 7.13.1(b), 7.13.1(c), 7.14.1(a), 8.2.1(c), 9.1.1(b)

Messages Produced: TJC-209, TJC-217

APN: 13 - Data Register Shift

Purpose: The data register shift test verifies that the data register selected for scan can be scanned without inversion and is the appropriate length.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Capture and shift the instruction register. Load instruction register with instruction selecting target register. `CaptureIR` behavior is not verified. This step is omitted if target register is the home register.

2. Capture and shift the target data register. Load the target register with the shift signature. If the target register is the boundary register, follow load of shift signature with load of safe state. Expect five bit shift signature on unload of register. `CaptureDR` state is not verified.
3. Return to home state.

   For standard registers: `BYPASS`, `DEVICE_IDENTIFICATION`, `BOUNDARY`

   In all cases, BSV over-shifts the data register with a shift signature of five bits (10011) which results in a shift length of: six bits for the `BYPASS` register, 37 bits for the `DEVICE_IDENTIFICATION` register, and `BOUNDARY_LENGTH`+5 bits for the `BOUNDARY` register. BSV expects the five bit signature to appear at TDO after the appropriate number of shift clock cycles for the target register.

   For non-standard registers:

   In all cases, BSV over-shifts the data register with a shift signature of five bits (10011). The `REGISTER_ACCESS` statement defines the length of the register. BSV expects the five bit signature to appear at TDO after the appropriate number of shift clock cycles for the target register.

Rules Supported: 7.6.1(c), 7.7.1(c), 8.2.1(b), 8.3.1(b), 9.1.1(a), 10.2.1(c), 10.2.1(i), 10.3.1(a)

Messages Produced: TJC-212, TJC-217

APN: 14 - Data Register Capture and Shift

Purpose: The data register capture and shift test verifies the `CaptureDR` behavior of the data register selected for scan. This test also verifies the zero transitions out of the `SelectDR` and `CaptureDR` states in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Capture and shift the instruction register. Load instruction register with instruction selecting target register. `CaptureIR` behavior is not verified. This step is omitted if target register is home register.
2. Capture and shift the target data register if a data register capture value is defined for the instruction or if the target data register is the boundary register. If the target register is the

boundary register, load a safe state into the boundary register. `CaptureDR` is not verified in this step. If no capture value is defined for the target register or the target register is not the boundary register, this step is omitted.

3. Capture and shift the target data register. Load the target register with the shift signature. Expect appropriate capture-shift behavior. `CaptureDR` state is not verified if target register is boundary register.
4. Return to home state.

For standard registers: `BYPASS`, `DEVICE_IDENTIFICATION`

If the DEVICE IDENTIFICATION register is selected for scan, then BSV will expect the `IDCODE` or `USERCODE` value is captured into the DEVICE IDENTIFICATION register and is subsequently scanned out via a `CaptureDR`/`ShiftDR`. If an `IDCODE` instruction is active, then BSV will expect the `IDCODE_REGISTER` value is scanned out. Conversely, if a `USERCODE` instruction is active, then BSV will expect the `USERCODE_REGISTER` value will be scanned out. If the `BYPASS` register is selected for scan, then BSV will expect that a capture value of one is captured into the `BYPASS` register and is subsequently scanned out via a `CaptureDR` / `ShiftDR` on the selected register. The `CaptureDR` behavior for the `BOUNDARY_REGISTER` is verified by one or more of the `BOUNDARY_REGISTER` checks.

For non-standard registers

BSV will expect to scan out the capture value defined in the `REGISTER_ACCESS` statement provided there is at least one bit which is expected to capture a known state (0 or 1). BSV scans out the capture value by performing a `CaptureDR`/ `ShiftDR` on the selected data register.

In all cases, BSV uses a five bit shift signature of 10011 to verify that the target register has been selected for scan. This results in a five bit over-shift of the target register. For example, BSV will expect the five bit signature to appear on TDO after 32 TCK clock cycles when verifying an instruction which accesses the standard `DEVICE_IDENTIFICATION` register.

Rules Supported: 5.1.1(a) - `SelectDR`/0, `CaptureDR`/0, 7.1.1(c), 7.4.1(c), 8.3.1(d)

Messages Produced: TJC-209, TJC-217

APN: 15 - Data Register Shift Without Capture

Purpose: The data register scan without capture test verifies that the data register selected for scan can be scanned without being preceded by a `CaptureDR`. This test also verifies the following transitions in the TAP FSM: the zero transition out of the `Exit1DR` state, the zero and one transitions out of the `ShiftDR` state, the one transition out of the `PauseDR` state and the zero transition out of the `Exit2IR` state.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load instruction register with instruction selecting target register. If home instruction selects target register, transfer to the `RunTestIdle` state.
2. Load the target register with the known state. There no expects on the unload of the target register.
3. Traverse from `Exit1DR`: `PauseDR`: `Exit2DR`: `ShiftDR`
4. Unload the target register. Reload register with five bit shift signature. Expect previous known state followed by five bit shift signature on register unload.
5. Return to home state

This test performs two back-to-back scans on the data register selected for scan. The first load of the data register preconditions the target register to a known state. There are no expects during the preconditioning. The second load/unload of the target register is expected to result in a TDO unload of the previously loaded known state followed by a five bit shift signature (10011). The shift signature is expected to appear at TDO after N shift clock cycles where N is the length of the target register. The total number of shift cycles in the second load/ unload is N+5 cycles.

For non-standard registers, the `REGISTER_ACCESS` statement defines the length of the register.

The TDO output pin is expected to remain in an inactive state when the TAP is not in a `ShiftDR` or `ShiftIR` state.

Rules Supported: 5.1.1(a) - `Exit1DR`/0, `ShiftDR`/0, `ShiftDR`/1, `PauseDR`/1, `Exit2DR`/0

Messages Produced: TJC-213, TJC-217

APN: 16 - Data Register Capture Without Immediate Shift

Purpose: The data register capture without shift test verifies the `CaptureDR` behavior of the selected data register when `ShiftDR` is not immediately preceded by a `CaptureDR` on the selected data register. This test also verifies the one transition out of the `CaptureDR` state in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load instruction register with instruction selecting target register. If home instruction selects target register, transfer to the `RunTestIdle` state.
2. Capture and shift the target data register if a data register capture value is defined for the instruction or if the target data register is the boundary register. If the target register is the boundary register, load a safe state into the boundary register otherwise load the complement of the defined capture value. `CaptureDR` is not verified in this step. If no capture value is defined for the target register and the target register is not the boundary register, this step is omitted.
3. Capture and shift the target data register. Enter the `ShiftDR` state by transferring from the `CaptureDR` state to `Exit1DR`: `PauseDR`: `Exit2DR`. Overshift the target register with the shift signature. Expect appropriate capture-shift behavior on data register unload. No `CaptureDR` expects if the data register is the boundary register or if no `CaptureDR` value is defined.
4. Return to home state.

This test verifies that the `CaptureDR` behavior of the logic when a `ShiftDR` does not immediately follow a `CaptureDR`. This test uses the following state traversal to capture and unload the target data register: `CaptureDR` to `Exit1DR` to `PauseDR` to `ShiftDR`.

For standard registers: `BYPASS`, `DEVICE_IDENTIFICATION`

If the DEVICE IDENTIFICATION register is selected for scan, then BSV will expect the `IDCODE` or `USERCODE` value is captured into the DEVICE IDENTIFICATION register and is subsequently scanned out via a `CaptureDR` / `ShiftDR`. If an `IDCODE` instruction is active, then BSV will expect the `IDCODE_REGISTER` value is scanned out. Conversely, if a `USERCODE` instruction is active, then BSV will expect the `USERCODE_REGISTER` value will be scanned out. If the `BYPASS` register is selected for scan, then BSV will expect that a capture value of one is captured into the `BYPASS` register and is subsequently scanned out via a `CaptureDR` / `ShiftDR` on the selected register. The `CaptureDR` behavior for the `BOUNDARY_REGISTER` is verified by one or more of the `BOUNDARY_REGISTER` checks.

For non-standard registers:

BSV will expect to scan out the capture value defined in the `REGISTER_ACCESS` statement provided there is at least one bit which is expected to capture a known state (0 or 1). BSV scans out the capture value by performing a `CaptureDR` / `ShiftDR` on the selected data register.

In all cases, BSV uses a five bit shift signature of 10011 to verify that the target register has been selected for scan. This results in a five bit over-shift of the target register. For example, BSV will expect the five bit signature to appear on TDO after 32 shift clock cycles when verifying an instruction which accesses the standard `DEVICE_IDENTIFICATION` register.

The TDO output pin is expected to remain in an inactive state when the TAP is not in a `ShiftDR` or `ShiftIR` state.

Note: The test may use different traversals from `Exit1DR` to `ShiftDR`

Rule Supported: 5.1.1(a) - `CaptureDR`/1

Messages Produced: TJC-217, TJC-218

APN: 17 - PauseDR Zero Transition

Purpose: The `PauseDR` zero transition test verifies the zero transition out of the `PauseDR` state in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load instruction register with instruction selecting target register. If home instruction selects target register, transfer to the RunTestIdle state.
2. Load the target register with the known state. There no expects on the unload of the target register.
3. Traverse from `Exit1DR`: `PauseDR`: `PauseDR`: `Exit2DR`: `ShiftDR` (includes target transition)
4. Unload the target register. Reload register with five bit shift signature. Expect previous known state followed by five bit shift signature on register unload.
5. Return to home state

This test performs two back-to-back scans on the data register selected for scan. The first load of the data register preconditions the target register to a known state. There are no expects during the preconditioning. The second load/unload of the target register is expected to result in a TDO unload of the previously loaded known state followed by a five bit shift signature (10011). The shift signature is expected to appear at TDO after N shift clock cycles where N is the length of the target register. The total number of shift cycles in the second load/ unload is N+5 cycles.

For non-standard registers, the `REGISTER_ACCESS` statement defines the length of the register.

The TDO output pin is expected to remain in an inactive state when the TAP is not in a `ShiftDR` or `ShiftIR` state.

Rule Supported: 5.1.1(a) - `PauseDR`/0

Messages Produced: TJC-213, TJC-217

APN: 18 - Data Register Update

Purpose: The data register update tests verifies the `UpdateDR` behavior of the data register selected for scan. It also verifies the one transitions out of the `Exit1DR` and `UpdateDR` states in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load instruction register with instruction selecting target register. If home instruction selects target register, transfer to the `RunTestIdle` state.
2. Load the target register with the known state. If the target register is the boundary register, load a safe state into the boundary register. There no expects on the unload of the target register.
3. Traverse to `UpdateDR` via `Exit1DR`.
4. Capture and shift the target data register. Load the target data register with the shift signature. Expect appropriate capture-shift behavior. No `CaptureDR` expects if the target register is the boundary register or if no capture value is defined.
5. Return to home state.

The first load of the data register preconditions the target register to a known state. There are no expects during the preconditioning. The second load/unload of the target register is expected to result in a TDO unload a defined register capture value for the target instruction followed by a five bit shift signature (10011). The shift signature is expected to appear at TDO after N shift clock cycles where N is the length of the target register. The total number of shift cycles in the second load/unload is N+5 cycles.

For standard registers: `BYPASS`, `DEVICE_IDENTIFICATION`

If the DEVICE IDENTIFICATION register is selected for scan, then BSV will expect the `IDCODE` or `USERCODE` value is captured into the DEVICE IDENTIFICATION register and is subsequently scanned out via a `CaptureDR` / `ShiftDR`. If an `IDCODE` instruction is active, then BSV will expect the `IDCODE_REGISTER` value is scanned out. Conversely, if a `USERCODE` instruction is active, then BSV will expect the `USERCODE_REGISTER` value will be scanned out. If the `BYPASS` register is selected for scan, then BSV will expect that a capture value of one is captured into the `BYPASS` register and is subsequently scanned out via a `CaptureDR` / `ShiftDR` on the selected register. The `CaptureDR` behavior for the `BOUNDARY_REGISTER` is verified by one or more of the `BOUNDARY_REGISTER` checks.

BSV will expect to scan out the capture value defined in the `REGISTER_ACCESS` statement provided there is at least one bit which is expected to capture a known state (0 or 1). BSV scans out the capture value by performing a `CaptureDR` / ShiftDR on the selected data register. The `REGISTER_ACCESS` statement defines the length of the register.

The TDO output pin is expected to remain in an inactive state when the TAP is not in a `ShiftDR` or `ShiftIR` state.

Rule Supported: 5.1.1(a) - `Exit1DR`/1, `UpdateDR`/1

Messages Produced: TJC-214, TJC-217

APN: 19 - UpdateDR Zero Transition

Purpose: The data register update tests verifies the `UpdateDR` behavior of the data register selected for scan. It also verifies the zero transitions out of `UpdateDR` state and the one transition out of the `RunTestIdle` (RTI) state in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load instruction register with instruction selecting target register. If home instruction selects target register, transfer to the `RunTestIdle` state.
2. Capture and shift the target data register if a data register capture value is defined for the instruction or if the target data register is the boundary register. If the target register is the boundary register, load a safe state into the boundary register otherwise load the completion of the defined capture value. `CaptureDR` is not verified in this step. If no capture value is defined for the target register and the target register is not the boundary register, this step is omitted.
3. Traverse to `CaptureDR` from `Exit1DR`: `PauseDR`: `Exit2DR`: `UpdateDR`: `RunTestIdle`
4. Capture and shift the target data register. Load the target data register with the shift signature. Expect appropriate capture-shift behavior. No `CaptureDR` expects if the target register is the boundary register or if no capture value is defined.
5. Return to home state.

Rule Supported: 5.1.1(a) - `UpdateDR`/0, `RunTestIdle`/1

Messages Produced: TJC-214, TJC-217

APN: 20 - Exit2DR One Transition

Purpose: The `Exit2DR` tests verifies the one transition out of `Exit2DR` state in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load instruction register with instruction selecting target register. If home instruction selects target register, transfer to the `RunTestIdle` state.
2. Capture and shift the target data register if a data register capture value is defined for the instruction or if the target data register is the boundary register. If the target register is the boundary register, load a safe state into the boundary register otherwise load the completion of the defined capture value. `CaptureDR` is not verified in this step. If no

capture value is defined for the target register and the target register is not the boundary register, this step is omitted.

3. Traverse to `CaptureDR` from `Exit1DR`: `PauseDR`: `Exit2DR`: `UpdateDR`: `SelectDR`
4. Capture and shift the target data register. Load the target data register with the shift signature. Expect appropriate capture-shift behavior. No `CaptureDR` expects if the target register is the boundary register or if no capture value is defined.
5. Return to home state.

Rule Supported: 5.1.1(a) - `Exit2DR`/1

Messages Produced: TJC-214, TJC-217

APN: 21 - Run Test Idle Zero Transition - RTI/0

Purpose: The Run Test Idle zero transition test verifies the zero transition out of the Run Test Idle state in the TAP FSM.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load instruction register with instruction selecting target register. If home instruction selects target register, transfer to the `RunTestIdle` state (include target transition in `RunTestIdle` State).
2. Capture and shift the target data register. Load the target data register with the shift signature. Expect appropriate capture-shift behavior. No `CaptureDR` expects if the target register is the boundary register or if no capture value is defined.
3. Return to home state.

The load/unload of the target register is expected to result in a TDO unload a defined register capture value for the target instruction followed by a five bit shift signature (10011). The shift signature is expected to appear at TDO after N shift clock cycles where N is the length of the target register. The total number of shift cycles in the second load/ unload is N+5 cycles.

For standard registers: `BYPASS`, `DEVICE_IDENTIFICATION`, `BOUNDARY`

If the DEVICE IDENTIFICATION register is selected for scan, then BSV will expect the `IDCODE` or `USERCODE` value is captured into the DEVICE IDENTIFICATION register and is subsequently scanned out via a `CaptureDR` / `ShiftDR`. If an `IDCODE` instruction is active, then BSV will expect the `IDCODE_REGISTER` value is scanned out. Conversely, if a `USERCODE` instruction is active, then BSV will expect the `USERCODE_REGISTER` value will be scanned out. If the `BYPASS` register is selected for scan, then BSV will expect that a capture value of one is captured into the `BYPASS` register and is subsequently scanned out via a `CaptureDR` / `ShiftDR` on the selected register. The `CaptureDR` behavior for the `BOUNDARY_REGISTER` is verified by one or more of the `BOUNDARY_REGISTER` checks.

For non-standard registers:

BSV will expect to scan out the capture value defined in the `REGISTER_ACCESS` statement provided there is at least one bit which is expected to capture a known state (0 or 1). BSV scans out the capture value by performing a `CaptureDR` / `ShiftDR` on the selected data register. The `REGISTER_ACCESS` statement defines the length of the register.

The TDO output pin is expected to remain in an inactive state when the TAP is not in a `ShiftDR` or `ShiftIR` state.

Rule Supported: 5.1.1(a) - RTI/0

Messages Produced: TJC-209, TJC-217

APN: 22 - PRELOAD

Purpose: The `PRELOAD` test verifies that boundary register parallel (update) latches defined for system output pins are loaded during the preload phase of the `SAMPLE/PRELOAD` instruction and that the state of the these update latches is subsequently observed at the system pin when `EXTEST` becomes the active instruction.

Explanation:

Home state is: TestLogicReset
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load the instruction register with an instruction opcode for the `SAMPLE/PRELOAD` instruction.

2. Load the boundary register such that all system output pins will be driven to an active (known state). There are no expects on the unload of the boundary register (i.e. Capture and shift is not verified in this step).
3. Load the instruction register with an instruction opcode for the `EXTEST` instruction. There are no expects on the unload of the instruction register. Traverse to `UpdateIR` via `Exit1IR`. Expect all system output pins to change to their respective preload state on the falling edge of the TCK clock pulse which triggers entry into the `UpdateIR` state.
4. Return to the home state.

This test verifies that the boundary register parallel (update) latches defined for output cells can be loaded via the `PRELOAD` instruction and their values subsequently observed at the system output pins when the `EXTEST` instruction becomes the active instruction (`EXTEST` instruction is latched into the instruction register parallel (update) latches). In particular, all system output pins are expected to change to their respective preload state on the falling edge of the TCK clock pulse which triggers entry into the `UpdateIR` state. The boundary register is preloaded as follows: all output cells are set to an active (known) state. Typically, this will be the same state for all system output pins. However, if an output is asymmetrical, then it must be driven to active state even if it differs from all other outputs. All control cells are set to enable their respective drivers. All other cells are set to their safe state.

Rules Supported: 7.6.1(e), 10.3.1(b), 10.6.1(g), 10.6.1(i)

Messages Produced: TJC-217, TJC3-10

APN: 23 - SAMPLE

Purpose: The `SAMPLE` test verifies that boundary register latches defined for system input pins capture the value of their respective system input pin during the sample phase of the `SAMPLE`/`PRELOAD` instruction.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load the instruction register with an instruction opcode for the `SAMPLE` instruction. There are no expects on the unload of the instruction register.
2. Load the boundary register such that all bidirectional system output pins will be driven to an inactive state. Load all input capable cells to the complement of the capture value.

There are no expects on the unload of the boundary register (that is, Capture and shift is not verified in this step).

3. Stim all system input pins to a known state. Load the instruction register with an instruction opcode for the `SAMPLE/PRELOAD` instruction. There are no expects on the unload of the instruction register.
4. Unload the boundary register. Expect all input capable cells to have the value of their respective input pin followed by the load and unload of the shift signature. Leave the boundary register in a safe state.
5. Return to the home state.

This test verifies that the boundary register capture/scan latches defined for system input pins are capable of observing the value of their respective input pins during the `SAMPLE` phase of the `SAMPLE/PRELOAD` instruction. The boundary register is first preconditioned to: disable all drivers associated with bidirectional system pins, to set the capture latches to the complement of the expected capture state and to set all other boundary cells to their safe state. The system input pins are set to a known state prior to the `CaptureDR` event while the `SAMPLE/PRELOAD` instruction is active. After the `SAMPLE/PRELOAD CaptureDR` event, the boundary register is unloaded with the expected results as follows: All input capable cells are expected to have observed (captured) the state of their respective input pin. As the boundary register capture state is unloaded, it is reloaded with a five bit shift signature (10011), followed by a boundary register safe state.

Rules Supported: 7.6.1(d), 10.5.1(i), 10.5.1(f)

Messages Produced: TJC-217, TJC-320

APN: 24 - Boundary Register I/O Mapping - System Input Pin

Purpose: The boundary register I/O mapping - system input pin test verifies that a system input pin is observed in the appropriate boundary register cells. This check is done in EXTEST mode.

Explanation:

Home state is: `CaptureDR`
Home instruction is: `EXTEST`
Home register is: `BOUNDARY_REGISTER`

Verification Procedure:

1. Traverse from the home state to the `ShiftDR` state

2. Directly precondition the boundary register such that the expected change is seen in the observe cells for the target input pin and that any unexpected change can be seen in all other input cells.
3. Traverse from `ShiftDR` to back to `CaptureDR` to complete boundary register preconditioning.
4. Set all system input pins to a known state. Set the target input pin to the target pin state and set all non-target input pins are set to the complement of the target pin state.
5. Traverse to the `ShiftDR` state.
6. Directly observe the state of all boundary scan chain cells (scan/capture latches) associated with system input pins to determine whether they changed or did not change state as expected. The target input cell is expected to observe the target pin state. All other input cells are expected to observe the complement of the target pin state.
7. Return to home state (includes returning bidirectional system pins to their inactive state)

This test verifies that the state of a single input pin (target pin) can be observed in the boundary register latches which are defined as observe cells (target cells) for the input pin. The test also verifies that the state of the input pin is not seen in boundary register cells which are not defined as observe cells for the input pin. BSV uses the BSDL `BOUNDARY_REGISTER` definition to determine which cells are observe cells for a particular pin. Any `INPUT`, `BIDIR`, `OBSERVE_ONLY`, or `CLOCK` cell defined for a pin is considered an observe cell for the pin and the pin is considered a system input pin. All system input pins are exercised by this test. The boundary register latches are structurally identified by Encounter Test and are positionally mapped to the cells defined in the `BOUNDARY_REGISTER` definition.

Rules Supported: 10.5.1(a), 10.5.1(i), 10.5.1(f), 10.7.1(a.i), 10.7.1(a.iii)

Messages Produced: TJC-217, TJC-300

APN: 25 - Boundary Register I/O Mapping - System Output Pin

Purpose: The boundary register I/O mapping - system output pin test verifies that the state of a system output pin is determined by the appropriate boundary register cells.

Explanation:

Home state is: `CaptureDR`
Home instruction is: `EXTEST`
Home register is: `BOUNDARY_REGISTER`

Verification Procedure:

1. Traverse from the home state to the `ShiftDR` state
2. Directly precondition the state of the boundary register such that the target system output pin will be driven to a known target state (state K) and all other system output pins will be driven to the complement of the target state (state L) or an inactive state (state M).
3. Traverse from `ShiftDR` to `CaptureDR` to complete boundary register preconditioning. This step includes traversing through `UpdateDR` allowing the system output pins to go to the state determined by the preconditioning of the boundary register.
4. Observe the state of all system output pins to determine whether the target system pin changed to the expected state (state K) and all other system output pins changed to the their expected state (state L or M).

This test verifies that a single system output pin can be driven to a state defined by data held in the appropriate boundary register latches. BSV uses the `BOUNDARY_REGISTER` definition to determine which pins are considered system output pins. Any pin defined with a cell function of `OUTPUT2`, `OUTPUT3`, or `BIDIR` is considered a system output pin. All system output pins are exercised by this test. BSV also use the `BOUNDARY_REGISTER` definition to determine the mapping of output and `CONTROL/CONTROLR` cells to system output pins. The boundary register latches are structurally identified by Encounter Test and are positionally mapped to the cells defined in the `BOUNDARY_REGISTER` definition.

Rules Supported: 10.5.1(e.i), 10.5.1(e.11), 10.6.1(a), 10.6.1(b), 10.6.1(c), 10.6.1(d), 10.6.1(g), 10.6.1(m), 10.7.1(a.ii), 10.7.1(a.iii)

Messages Produced: TJC-217, TJC-302

APN: 26 - Boundary Register I/O Mapping - Share Enable

Purpose: The boundary register I/O mapping - share enable test verifies that a `BOUNDARY_REGISTER` control cell which controls more than one system output controls the appropriate system output pins.

Explanation:

Home state is: `CaptureDR`
Home instruction is: `EXTEST`
Home register is: `BOUNDARY_REGISTER`

Verification Procedure:

1. Traverse from the home state to the `ShiftDR` state.

2. Directly precondition the state of the boundary register such that the target system output pins will be driven to a known target state (state K) and all other system output pins will be driven to the complement of the target state (state L) or an inactive state (state M).
3. Traverse from `ShiftDR` to `CaptureDR` to complete boundary register preconditioning. This step includes traversing through `UpdateDR` allowing the system output pins to go to the state determined by the preconditioning of the boundary register.
4. Observe the state of all system output pins to determine whether the target system pins changed to the expected state (state K) and all other system output pins changed to the their expected state (state L or M).

A shared enable is a control cell which controls more than one system output pin as defined in the `BOUNDARY_REGISTER` statement. This test verifies that all system output pins controlled by a shared enable can be simultaneously driven to a state defined by data held in the boundary register output cells for each output controlled by the target control cell. BSV uses the `BOUNDARY_REGISTER` to determine which pins are considered system output pins and which pins are controlled by a shared enable (`CONTROL`/`CONTROLR` cell). Any pin defined with a cell function of `OUTPUT2`, `OUTPUT3`, or `BIDIR` is considered a system output pin. Any `CONTROL`/`CONTROLR` cell which is referenced by more than one system output pin is considered a shared enable and will be exercised by this test. The boundary register latches are structurally identified by Encounter Test and are positionally mapped to the cells defined in the `BOUNDARY_REGISTER` definition.

Rules Supported: 7.7.1(e), 10.6.1(e), 10.6.1(d), 10.6.1(g)

Messages Produced: TJC-217, TJC-304

APN: 27 - Boundary Register - Enable All System Output Drivers

Purpose: The boundary register enable all output drivers test verifies that all system output pins can be simultaneously driven to an active (known) state on the appropriate falling edge of TCK.

Explanation:

Home state is: `CaptureDR`
Home instruction is: `EXTEST`
Home register is: `BOUNDARY_REGISTER`

Verification Procedure:

1. Traverse from the home state to the `ShiftDR` state.

2. Precondition the state of the boundary register such that the system output pins will be driven to a known target state (state K).

3. Traverse from `ShiftDR` to `CaptureDR` to complete boundary register preconditioning. This step includes traversing through `UpdateDR` allowing the system output pins to go to the state determined by the preconditioning of the boundary register. Observe the state of all system output pins to determine whether the target system pins changed to the expected state (state K) and that the change occurs on the falling edge of the TCK clock event which triggers entry into the `UpdateDR` state.

4. Return to the home state.

This test verifies that all system output pins can be simultaneously driven to a state defined by data held in the boundary register on the falling edge of the TCK clock event which triggers entry into the `UpdateDR` state. BSV uses the `BOUNDARY_REGISTER` to determine which pins are considered system output pins. Any pin which has a `BOUNDARY_REGISTER` cell function of `OUTPUT2`, `OUTPUT3`, or `BIDIR` is considered a system output pin. The boundary register latches are structurally identified by Encounter Test and are positionally mapped to the cells defined in the `BOUNDARY_REGISTER` definition.

Rules Supported: 7.7.1(e), 8.3.1(e.i), 10.3.1(b), 10.6.1(d), 10.6.1(e) 10.6.1(g), 10.6.1(j)

Messages Produced: TJC-217, TJC-310

APN: 28 - Boundary Register - Disable All System Output Drivers

Purpose: The boundary register disable all output drivers test verifies that all system output pins can be simultaneously driven to an inactive state on the appropriate falling edge of TCK via the `EXTEST` instruction.

Explanation:

Home state is: `CaptureDR`
Home instruction is: `EXTEST`
Home register is: `BOUNDARY_REGISTER`

Verification Procedure:

1. Traverse from the home state to the `ShiftDR` state

2. Directly precondition the state of the boundary register such that the all system output pins will be driven to a known target state (state K).

3. Traverse from `ShiftDR` to `CaptureDR` to complete boundary register preconditioning. This step includes traversing through `UpdateDR` allowing the system output pins to go to the state determined by the preconditioning of the boundary register.

4. Traverse from `CaptureDR` to `ShiftDR`
5. Precondition the state of the boundary register such that each system output pin which has an inactive state will be driven to its inactive state on the falling edge of the TCK clock event which triggers entry into the `UpdateDR` state.
6. Return to the home state.

This test verifies that all system output pins which have inactive states can be simultaneously disabled based on data held in the boundary register on the falling edge of the TCK clock event which triggers entry into the `UpdateDR` state. BSV uses the `BOUNDARY_REGISTER` to determine which pins are considered system output pins with an inactive state. Any pin which has a `BOUNDARY_REGISTER` cell function of `OUTPUT2`, `OUTPUT3`, or `BIDIR` and is defined with a disable specification is considered a system output pin with and inactive state. The boundary register latches are structurally identified by Encounter Test and are positionally mapped to the cells defined in the `BOUNDARY_REGISTER` definition.

Rules Supported: 7.7.1(e), 8.3.1(e.i), 10.6.1(e), 10.6.1(g), 10.6.1(j)

Messages Produced: TJC-217, TJC-322

APN: 29 - Boundary Register - All System Output Drivers Retain Active State

Purpose: This boundary register test verifies that all system output pins retain their state while the `EXTEST` instruction is while traversing through various states in the TAP FSM.

Explanation:

Home state is: `CaptureDR`
Home instruction is: `EXTEST`
Home register is: `BOUNDARY_REGISTER`

Verification Procedure:

1. Traverse from the home state to the `ShiftDR` state
2. Precondition the state of the boundary register such that the system output pins will be driven to a known target state (state K).
3. Traverse to `UpdateDR`. Observe the state of all system output pins to determine whether each pin changed to its active state on the falling edge of the TCK clock event which triggers entry into the `UpdateDR` state.
4. Expect each system output pin retains the pin state set while in the previous `UpdateDR` state, while traversing through the TAP states as follows:

5. `RunTestIdle`, `SelectDR`, `CaptureDR`, `Exit1DR`, `PauseDR`, `Exit2DR`, `ShiftDR`

This test verifies that all system output pins are capable of holding the state determined by their parallel (update) latches. The boundary register parallel latches associated with system output pins are preloaded (preconditioned) via the `EXTEST` instruction. Once the system output pins change state, they are expected to hold that state until either another `EXTEST` - `UpdateDR` event occurs, or a a new instruction becomes active, or a reset occurs. BSV uses the `BOUNDARY_REGISTER` to determine which pins are considered system output pins. Any pin which has a `BOUNDARY_REGISTER` cell function of `OUTPUT2`, `OUTPUT3`, or `BIDIR` will participate in this test. The system output pins which are participating in this test are expected to change state on the falling edge for the TCK clock pulse which triggers entry into the `UpdateDR` state.

Rules Supported: 7.7.1(e), 10.6.1(g), 10.6.1(i)

Messages Produced: TJC-217, TJC-311

APN: 31 - Boundary Register I/O Mapping - Share Disable

Purpose: The boundary register I/O mapping - share disable test verifies that a `BOUNDARY_REGISTER` control cell which controls more than one system output, controls the appropriate system output drivers and is capable of disabling them.

Explanation:

Home state is: `CaptureDR`
Home instruction is: `EXTEST`
Home register is: `BOUNDARY_REGISTER`

Verification Procedure:

1. Traverse from the home state (`CaptureDR`) to `ShiftDR`
2. Precondition the state of the boundary register such that the system output pins will be driven to a known target state (state X).
3. Traverse from `ShiftDR` to `CaptureDR` to complete boundary register preconditioning. This step includes traversing through `UpdateDR` allowing the system output pins to go to the state determined by the preconditioning of the boundary register.
4. Traverse from the `CaptureDR` to `ShiftDR`
5. Precondition the state of the boundary register such that the target system output pins will be driven to their inactive state while all other system output pins will remain at the target state (state X).

6. Traverse through `UpdateDR` to `RunTestIdle`. Expect all system output to change to the appropriate state on the falling edge of the TCK clock event which triggers entry into the `UpdateDR` state.
7. Return to the home state.

A shared enable is a control cell which controls more than one system output pin as defined in the `BOUNDARY_REGISTER` statement. This test verifies that all system output pins controlled by a shared enable can be simultaneously disabled. BSV uses the `BOUNDARY_REGISTER` to determine which pins are considered system output pins and which pins are controlled by a shared enable (`CONTROL`/`CONTROLR` cell). Any pin defined with a cell function of `OUTPUT2`, `OUTPUT3`, or `BIDIR` is considered a system output pin. All shared enable cells are exercised by this test. The boundary register latches structurally identified by Encounter Test are positionally mapped to the cells defined in the `BOUNDARY_REGISTER` definition.

Rules Supported: 10.6.1(d.i), 10.6.1(d.ii), 10.6.1(e), 10.6.1(f)

Messages Produced: TJC-217, TJC-321

APN: 32 - Boundary Register I/O Mapping - System Output Pin

This is the 1149.6 equivalent for the system output pin test. Refer to APN: 25 - Boundary Register I/O Mapping - System Output Pin on page 127 for more information.

APN: 33 - Boundary Register I/O Mapping - Share Enable

This is the 1149.6 equivalent for the share enable test. Refer to APN: 26 - Boundary Register I/O Mapping - Share Enable on page 128 for more information.

APN: 34 - Boundary Register I/O Mapping - Share Disable

This is the 1149.6 equivalent for the share disable test. Refer to APN: 31 - Boundary Register I/O Mapping - Share Disable on page 132 for more information.

APN: 35 - Boundary Register I/O Mapping - Enable All System Output Drivers

This is the 1149.6 equivalent for enable all output drivers test. Refer to APN: 27 - Boundary Register - Enable All System Output Drivers on page 129 for more information.

APN: 36 - Boundary Register I/O Mapping - All System Output Drivers Retain Active State

This is the 1149.6 equivalent for verifying that all system output pins retain their state while the `EXTEST_PULSE` instruction is traversing through various states in the TAP FSM. Refer to APN: 29 - Boundary Register - All System Output Drivers Retain Active State on page 131 for more information.

APN: 37 - Boundary Register I/O Mapping - Disable All System Output Drivers

This is the 1149.6 equivalent for verifying that all system output pins can be simultaneously driven to an inactive state on the appropriate falling edge of TCK through the `EXTEST_PULSE` instruction. Refer to APN: 28 - Boundary Register - Disable All System Output Drivers on page 130 for more information.

APN: 39 - Boundary Register - BiDir IOWRAP

Purpose: The boundary register bidir `IOWRAP` test verifies that all bidirectional system pins can observe the state of their on-product driver.

Explanation:

Home state is: `CaptureDR`
Home instruction is: `EXTEST`
Home register is: `BOUNDARY_REGISTER`

Verification Procedure:

1. Traverse from the home state (`CaptureDR`) to `ShiftDR`
2. Precondition the state of the boundary register such that the system output pins will be driven to a known target state (state X).
3. Traverse from `ShiftDR` to `CaptureDR` to complete boundary register preconditioning. This step includes traversing through `UpdateDR` allowing the system output pins to go to the state determined by the preconditioning of the boundary register.
4. Traverse from the `CaptureDR` to `ShiftDR`
5. Directly observe the state of all input cells associated with bidirectional system pins
6. Return to the home state.

This test verifies that the state of the on product driver of a bidirectional system pin can be observed in the observe cells defined for the pin. The test enables all drivers on all bidirectional pins and then looks to see if the active state of the all active drivers is observed in observe cells defined for the bidirectional pins. BSV uses the `BOUNDARY_REGISTER` to determine which pins will participate in this test. Any bidirectional system pin which has both an output capable (data) cell (`OUTPUT2`, `OUTPUT3`, `BIDIR`) and one or more input capable cells (`INPUT`, `CLOCK`, `OBSERVE_ONLY`) participates in this test. The boundary register latches structurally identified by Encounter Test are positionally mapped to the cells defined in the `BOUNDARY_REGISTER` definition.

Rules Supported: 10.7.1(a.i), 10.7.1(a.ii)

Messages Produced: TJC-217, TJC-321

APN: 40 - Capture All System Input Pins

Purpose: The boundary register capture all input pins test verifies that all system input pins are simultaneously observed in the appropriate boundary register latches.

Explanation:

Home state is: `CaptureDR`
Home instruction is: `EXTEST`
Home register is: `BOUNDARY_REGISTER`

Verification Procedure:

1. Preconditioning the state of the boundary register such that the expected change is seen in the observe latches for all system input pins and that any unexpected change can be seen in all other boundary latches.
2. Set the target input pins to a known state
3. Traverse to `CaptureDR` and then to `ShiftDR`
4. Directly observe the state of all scan/capture latches in the boundary register to determine whether they changed or did not change state as expected.

This test verifies that the state of all system inputs can be observed in the boundary register latches which are defined as observe cells for the input pins. Any `INPUT`, `OBSERVE_ONLY`, or `CLOCK` cell defined for a pin is considered an observe cell. The test also verifies that states of the input pins are not seen in boundary register cells which are not defined as observe cells for the input pins. BSV uses the BSDL `BOUNDARY_REGISTER` definition to determine which cells are observe cells for a particular pin. The boundary register latches identified by Encounter Test are positionally mapped to the cells defined in the `BOUNDARY_REGISTER` definition.

Rules Supported: 7.6.1(d), 7.7.1(f)

Messages Produced: TJC-217, TJC-320

APN: 41 - All System Output Pins Inactive

Purpose: The HIGHZ all system output pins inactive test verifies that all system output pins are driven to an inactive state when the HIGHZ becomes the active instruction.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load an instruction opcode for the HIGHZ instruction.
2. Traverse to `UpdateIR` via `Exit1IR`. Expect all system output pins to go to their inactive state on the falling edge of the TCK clock pulse which triggers entry into `UpdateIR`.
3. Expect all system output pins will remain at their inactive state as this test traverses through the following sequence of TAP states:

   `RunTestIdle`, `SelectDR`, `CaptureDR`, `ShiftDR`,`Exit1DR`, `PauseDR`, `Exit2DR`, `UpdateDR`, `SelectDR`, `SelectIR`, `CaptureIR`, `ShiftIR`, `Exit1IR`, `PauseIR`, `Exit2IR`, `UpdateIR`, `SelectDR`, `CaptureDR`, `Exit1DR`, `PauseDR`

   Note:

   - When traversing through `ShiftIR` in the above sequence, the instruction register is loaded with an opcode for the HIGHZ instruction.
   - When traversing through `ShiftDR` in the above sequence, the data register is loaded with a known state. One bit is shifted in/out of the register.
   - No expects data is included when in either the `ShiftIR` or `ShiftDR` states.
4. Return to the home state.

This test verifies that the all system output pins are driven to an inactive state when the HIGHZ instruction is selected. First, all bidirectional system pins are preconditioned to their inactive state so that simulation will not produce a unknown logic state when computing the state for such output pins. Here, the inactive state for a pin determined by logic model for the I/O pin. The initial state of all system output drivers is unknown. The HIGHZ instruction is selected when one of its defined opcodes is latched in the instruction register parallel (update) latches. The `INSTRUCTION_OPCODE` statement defines the opcodes for an instruction. The `BOUNDARY_REGISTER` statement defines what pins have a system output capability. Any pin associated with a cell function of `OUTPUT2`, `OUTPUT3`, or `BIDIR` will participate in this test. A pin's inactive state is explicitly defined by the `BOUNDARY_REGISTER` disable result of the disable specification for the data cell of the pin or it is implicitly defined as HIGHZ for a two state output pin whose pin type is buffer. Support for the HIGHZ instruction requires that all system output pins be driven to an inactive state while the HIGHZ instruction is active. BSV will expect that the output pins switch to an inactive state on the falling edge of the TCK clock pulse which triggers entry into the `UpdateIR` state. Thereafter BSV will expect all system output pins to remain in their inactive state for the duration of this test.

Rules Supported: 7.14.1(b), 10.6.1(g), 10.6.1(h), 10.6.1(k)

Messages Produced: TJC-217, TJC-327, TJC-328

APN: 42 - CLAMP - All System Output Pins Active

Purpose: The `CLAMP` all system output pins active test verifies that all system output pins are driven to an active state when the `CLAMP` becomes the active instruction. This test also verifies that all system output pins remain at the '`clamped`' state while the `CLAMP` instruction is active.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load the instruction register with an instruction opcode for the `SAMPLE`/`PRELOAD` instruction.
2. Load the boundary register with values which will enable all output drivers and drive all system output pins to an active state. There are no expects on the unload of the boundary register (i.e. Capture and shift is not verified in this step).
3. Load the instruction register with an instruction opcode for the `CLAMP` instruction. There are no expects on the unload of the instruction register. Traverse to `UpdateIR` via `Exit1IR`. Expect all system output pins to change to their respective preload state on the falling edge of the TCK clock pulse which triggers entry into the `UpdateIR` state.
4. Expect all system output pins will remain at their respective active state as this test traverses through the following sequence of TAP states:

`RunTestIdle, SelectDR, CaptureDR, ShiftDR,Exit1DR, PauseDR, Exit2DR, UpdateDR, SelectDR, SelectIR, CaptureIR, ShiftIR, Exit1IR, PauseIR, Exit2IR, UpdateIR, SelectDR, CaptureDR, Exit1DR, PauseDR`.

Note:

- When traversing through `ShiftIR` in the above sequence, the instruction register is loaded with an opcode for the `CLAMP` instruction.
- When traversing through `ShiftDR` in the above sequence, the data register is loaded with a known state. One bit is shifted in/out of the register.
- No expects data is included when in either the `ShiftIR` or `ShiftDR` states.

5. Return to the home state.

This test verifies that the all system output pins are driven to an active state when the `CLAMP` instruction is selected. This test also verifies that all system output pins hold their '`clamped`' state while the `CLAMP` instruction is active. The `CLAMP` instruction is selected when one of its defined opcodes is latched in the instruction register parallel (update) latches. The `INSTRUCTION_OPCODE` statement defines the opcodes for an instruction. The `BOUNDARY_REGISTER` statement defines what pins have a system output capability. Any pin associated with a cell function of `OUTPUT2`, `OUTPUT3`, or `BIDIR` will participate in this test. BSV will expect that the output pins switch to their preconditioned (active) state on the falling edge of the TCK clock pulse which triggers entry into the `UpdateDR` state. Thereafter BSV will expect all system output pins to remain in their respective active state for the duration of this test. BSV will precondition each output cell to an active (known) state via the `PRELOAD` phase of the `SAMPLE`/`PRELOAD` instruction.

Rules Supported: 7.10.1(b), 7.10.1(c), 10.6.1(g), 10.6.1(i), 10.6.1(k)

Messages Produced: TJC-217, TJC-310, TJC-311

APN: 43 - RUNBIST - All System Output Pins Inactive

Purpose: The `RUNBIST` all system output pins inactive test verifies that all system output pins are driven to an inactive state when `RUNBIST` becomes the active instruction.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load an instruction opcode for the `RUNBIST` instruction.
2. Traverse to `UpdateIR` via `Exit1IR`. Expect all system output pins to go to their inactive state on the falling edge of the TCK clock pulse which triggers entry into `UpdateIR`.
3. Expect all system output pins will remain at their inactive state as this test traverses through the following sequence of TAP states:

   ```
   RunTestIdle, SelectDR, CaptureDR, ShiftDR,Exit1DR, PauseDR,
   Exit2DR, UpdateDR, SelectDR, SelectIR, CaptureIR, ShiftIR,
   Exit1IR, PauseIR, Exit2IR, UpdateIR, SelectDR, CaptureDR,
   Exit1DR, PauseDR.
   ```

   Note:

   - When traversing through `ShiftIR` in the above sequence, the instruction register is loaded with an opcode for the `RUNBIST` instruction.
   - When traversing through `ShiftDR` in the above sequence, the data register is loaded with a known state. One bit is shifted in/out of the register.
   - No expects data is included when in either the `ShiftIR` or `ShiftDR` states.
4. Return to the home state.

This test verifies that the all system output pins are driven to an inactive state when the `RUNBIST` instruction is selected. BSV uses the `BSDL RUNBIST_EXECUTION` statement to determine how the system output pins should behave when the `RUNBIST` instruction is active. This test is included in the verification suite when BSV determines that the `RUNBIST_EXECUTION` statement defines `'OBSERVING HIGHZ AT_PINS'`.

First, all bidirectional system pins are preconditioned to their inactive state so that simulation will not produce a unknown logic state when computing the state for such output pins. Here, the inactive state for a pin determined by logic model for the I/O pin. The initial state of all system output drivers is unknown. The `RUNBIST` instruction is selected when one of its defined opcodes is latched in the instruction register parallel (update) latches. The `INSTRUCTION_OPCODE` statement defines the opcodes for an instruction. The `BOUNDARY_REGISTER` statement defines what pins have a system output capability. Any pin associated with a cell function of `OUTPUT2`, `OUTPUT3`, or `BIDIR` will participate in this test. A pin's inactive state is explicitly defined by the `BOUNDARY_REGISTER` disable result of the disable specification for the data cell of the pin or it is implicitly defined as `HIGHZ` for a two state output pin whose pin type is buffer. Support for the `RUNBIST` which observes `HIGHZ` at all system output pins requires that all system output pins be driven to an inactive state while the `RUNBIST` instruction is active. BSV will expect that the output pins switch to an inactive state on the falling edge of the TCK clock pulse which triggers entry into the `UpdateIR` state. Thereafter BSV will expect all system output pins to remain in their inactive state for the duration of this test.

Rules Supported: 7.9.k(ii), 10.6.1(g), 10.6.1(i)

Messages Produced: TJC-217, TJC-327, TJC-328

APN: 44 - RUNBIST - All System Output Pins Active

Purpose: The `RUNBIST` all system output pins active test verifies that all system output pins are driven to an active state when `RUNBIST` becomes the active instruction.

Explanation:

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load the instruction register with an instruction opcode for the `SAMPLE`/`PRELOAD` instruction.
2. Load the boundary register with values which will enable all output drivers and drive all system output pins to an active state. There are no expects on the unload of the boundary register (i.e. Capture and shift is not verified in this step).
3. Load the instruction register with an instruction opcode for the `RUNBIST` instruction. There are no expects on the unload of the instruction register. Traverse to `UpdateIR` via `Exit1IR`. Expect all system output pins to change to their respective preload state on the falling edge of the TCK clock pulse which triggers entry into the `UpdateIR` state.
4. Expect all system output pins will remain at their respective active state as this test traverses through the following sequence of TAP states:

```
RunTestIdle, SelectDR, CaptureDR, ShiftDR, Exit1DR, PauseDR,
Exit2DR, UpdateDR, SelectDR, SelectIR, CaptureIR, ShiftIR,
Exit1IR, PauseIR, Exit2IR, UpdateIR, SelectDR, CaptureDR,
Exit1DR, PauseDR.
```

Note:

- When traversing through `ShiftIR` in the above sequence, the instruction register is loaded with an opcode for the `RUNBIST` instruction.
- When traversing through `ShiftDR` in the above sequence, the data register is loaded with a known state. One bit is shifted in/out of the register.
- No expects data is included when in either the `ShiftIR` or `ShiftDR` states.

5. Return to the home state.

This test verifies that the all system output pins are driven to an active state when the `RUNBIST` instruction is selected. BSV uses the `BSDL RUNBIST_EXECUTION` statement to determine how the system output pins should behave when the `RUNBIST` instruction is active. This test is included in the verification suite when BSV determines that the `RUNBIST_EXECTUION` statement defines `'OBSERVING BOUNDARY AT_PINS'`.

First, all bidirectional system pins are preconditioned to their inactive state so that simulation will not produce a unknown logic state when computing the state for such output pins. Here, the inactive state for a pin determined by logic model for the I/O pin. The initial state of all system output drivers is unknown. The `RUNBIST` instruction is selected when one of its defined opcodes is latched in the instruction register parallel (update) latches. The `INSTRUCTION_OPCODE` statement defines the opcodes for an instruction. The `BOUNDARY_REGISTER` statement defines what pins have a system output capability. Any pin associated with a cell function of `OUTPUT2`, `OUTPUT3`, or `BIDIR` will participate in this test. Support for the `RUNBIST` which uses the `BOUNDARY_REGISTER` to precondition the state of all system output pin requires that all system output pins be capable of retaining their state while the `RUNBIST` instruction is active. BSV will expect that the output pins switch to their preconditioned (active) state on the falling edge of the TCK clock pulse which triggers entry into the `UpdateDR` state. Thereafter BSV will expect all system output pins to remain in their respective active state for the duration of this test. BSV will precondition each output cell to an active (known) state via the `PRELOAD` phase of the `SAMPLE/PRELOAD` instruction.

Rules Supported: 7.9.1(k.i), 7.9.1(k.ii), 10.6.1(g), 10.6.1(k)

Messages Produced: TJC-217, TJC-310, TJC-311

APN: 45 - Large Register Verification

Purpose: Large register verification uses a single capture and shift sequence to verify a register whose size would require a prohibitively long runtime in simulation to complete the full verification suite. If `scanonlysim` is enabled, this APN is performed in addition to the mode build structure verification that would otherwise be exclusively used to verify a register whose size exceeds the value for `maxsimreglength`. See “verify_11491_boundary” in the *Encounter Test: Reference: Commands* for details on these keywords.

Explanation

Home state is: `TestLogicReset`
Home instruction is: `BYPASS` or `IDCODE`
Home register is: `BYPASS` or `DEVICE_ID`

Verification Procedure:

1. Load the instruction register with an instruction opcode for the register to be verified.
2. Capture and shift the target data register. Load the target register with the shift signature. Expect appropriate capture-shift behavior.
3. Return to the home state.

The purpose of this check is to provide additional simulation-based checking over and above the structural checks which are performed for a large register. This check is only performed when *scanonlysim* has been specified. It is applicable to all instructions; however, it offers no additional check completeness for BSV's predefined checking groups.

For user-defined registers, BSV will expect to scan out the capture value defined in the `REGISTER_ACCESS` statement provided there is at least one bit which is expected to capture a known state (0 or 1). BSV scans out the capture value by performing a `CaptureDR` / `ShiftDR` on the selected data register. For data registers defined by the Standard, BSV will expect to scan out the capture value defined by the BSDL or Standard. For example, when this check is applied to the `BYPASS` register the expected capture value is always one.

BSV uses a five bit shift signature of 10011 to verify that the target register has been selected for scan. This results in a five bit over-shift of the target register.

Rules Supported: 5.1.1(a) - `SelectDR`/0, `CaptureDR`/0, `ShiftDR`/0, `ShiftDR`/1, 7.1.1(c), 8.3.1(d)

APN: 48 - Boundary Register I/O Mapping - System Input Pin

This is the 1149.6 equivalent for the system input pin test. This check is done in the `EXTEST_PULSE` mode. Refer to APN: 24 - Boundary Register I/O Mapping - System Input Pin on page 126 for more information.

APN: 49 - Boundary Register BiDir IOWRAP

This is the 1149.6 equivalent for the boundary register BiDir IOWRAP test. Refer to APN: 39 - Boundary Register - BiDir IOWRAP on page 134 for more information.

APN: 50 - Capture All System Input Pins

This is the 1149.6 equivalent for the boundary register capture all input pins test. Refer to APN: 40 - Capture All System Input Pins on page 135 for more information.

APN: 52 - Boundary Register I/O Mapping - System Output Pin

This is the 1149.6 equivalent for the system output pin test. Refer to APN: 25 - Boundary Register I/O Mapping - System Output Pin on page 127 for more information.

APN: 53 - Boundary Register I/O Mapping - Share Enable

This is the 1149.6 equivalent for the share enable test. Refer to APN: 26 - Boundary Register I/O Mapping - Share Enable on page 128 for more information.

APN: 54 - Boundary Register I/O Mapping - Share Disable

This is the 1149.6 equivalent for the share disable test. Refer to APN: 31 - Boundary Register I/O Mapping - Share Disable on page 132 for more information.

APN: 55 - Boundary Register I/O Mapping - Enable All System Output Drivers

This is the 1149.6 equivalent for the enable all output drivers test. Refer to APN: 27 - Boundary Register - Enable All System Output Drivers on page 129 for more information.

APN: 56 - Boundary Register I/O Mapping - All System Output Drivers Retain Active State

This is the 1149.6 equivalent for verifying that all system output pins retain their state while the `EXTEST_TRAIN` instruction is traversing through various states in the TAP FSM. Refer to APN: 29 - Boundary Register - All System Output Drivers Retain Active State on page 131 for more information.

APN: 57 - Boundary Register I/O Mapping - Disable All System Output Drivers

This is the 1149.6 equivalent for the disable all output drivers test via the `EXTEST_TRAIN` instruction. Refer to APN: 28 - Boundary Register - Disable All System Output Drivers on page 130 for more information.

APN: 58- Boundary Register I/O Mapping - System Input Pin

This is the 1149.6 equivalent for the system input pin test. This check is done in the `EXTEST_PULSE` mode. Refer to APN: 24 - Boundary Register I/O Mapping - System Input Pin on page 126 for more information.

APN: 59 - Boundary Register BiDir IOWRAP

This is the 1149.6 equivalent for the boundary register BiDir IOWRAP test. Refer to APN: 39 - Boundary Register - BiDir IOWRAP on page 134 for more information.

APN: 60- Capture All System Input Pins

This is the 1149.6 equivalent for the boundary register capture all input pins test. Refer to APN: 40 - Capture All System Input Pins on page 135 for more information.

IEEE 1149.1 BSDL Rules Verification

Table 2-1 shows whether specific rules from IEEE 1149.1 Std 1149.1b-1994 are verified by Encounter Test and if verified, the resulting message(s).

Table 2-2 on page 149 shows equivalent information for IEEE 1149.1-2001.

Table 2-1 IEEE 1149.1 1994 BSDL Rules Verification Table

1149.1b-1993/94 Rule	Verified	Message(s)
The entity description		
B.8.1.1	Yes	TJB-210
B.8.1.2(a)	No	
Generic Parameter Statement		
B.8.2.3(a.1)	No	
B.8.2.3(a.2)	No	
B.8.2.3(a.3)	No	
Logical port description statement		
B.8.3	Yes	TJB-216
B.8.3.3(a)	Yes	TJB-316
B.8.3.3(b)	Yes	TJB-249
Standard use statement		
B.8.4	Yes	TJB-285

1149.1b-1993/94 Rule	Verified	Message(s)
B.8.4.1.3(a)	No	
B.8.4.2.2(a)	Yes	Various
Use statement		
B.8.5.3(a)	No	
B.8.5.3(b.1)	No	
B.8.5.3(b.2)	No	
Component conformance statement		
B.8.6.1	Yes	TJB-286
Device package pin mappings		
B.8.7.1	Yes	TJB-207
B.8.7.3(a)	Yes	TJB-264
B.8.7.3(c)	Yes	TJB-228
B.8.7.3(d)	Yes	TJB-228
B.8.7.3(e)	Yes	TJB-317
B.8.7.3(f)	Yes	TJB-218
Grouped port identification		
B.8.8.3(a)	Yes	TJB-250
B.8.8.3(b)	Yes	TJB-221
B.8.8.3(c)	Yes	TJB-256
B.8.8.3(d)	Yes	TJB-257
B.8.8.3(e)	Yes	TJB-319
B.8.8.3(f)	Yes	TJB-219
Scan port identification		
B.8.9	Yes	TJB-201
B.8.9.3(a.1)	Yes	TJB-247
B.8.9.3(a.2)	Yes	TJB2-47
B.8.9.3(a.3)	No	

1149.1b-1993/94 Rule	Verified	Message(s)
B.8.9.3(b)	Yes	TJB-258
B.8.9.3(c)	Yes	TJB-251
B.8.9.3(d)	Yes	TJB-221
B.8.9.3(e)	Yes	TJB-321
Compliance enable description		
B.8.10.3(a)	Yes	TJB-243
B.8.10.3(b)	Yes	TJB-245
B.8.10.3(c)	Yes	TJB-246
B.8.10.3(d.1)	Yes	TJB-218
B.8.10.3(d.2)	Yes	TJB-247
B.8.10.3(e)	Yes	TJB-258
B.8.10.3(f)	Yes	TJB-221
B.8.10.3(g)	Yes	TJB-241
Instruction register description		
B.8.11	Yes	TJB-287, TJB-238 ,TJB-239
B.8.11.3	Yes	TJB-208
B.8.11.3(a)	Yes	TJB-203
B.8.11.3(b)	Yes	TJB-209
B.8.11.3(c)	Yes	TJB-213
B.8.11.3(d)	Yes	TJB-211
B.8.11.3(e)	Yes	TJB-214
B.8.11.3(f)	Yes	TJB-326
B.8.11.3(g)	Yes	TJB-204
B.8.11.3(g.2)	Yes	TJB-230
B.8.11.3(h)	Yes	TJB-303
B.8.11.3(i)	Yes	TJB-310
Optional register description		

1149.1b-1993/94 Rule	Verified	Message(s)
B.8.12.3(a)	Yes	TJB-235
B.8.12.3(b)	Yes	TJB-312
B.8.12.3(b.2)	Yes	TJB-311
B.8.12.3(c)	Yes	TJB-237
B.8.12.3(d)	No	
Register access description		
B.8.13.3(a)	No	
B.8.13.3(b)	Yes	TJB-300
B.8.13.3(c)	Yes	TJB-301
B.8.13.3(d)	Yes	TJB-309
B.8.13.3(e)	No	
B.8.13.3(f)	Yes	TJB-302
Boundary scan chain description		
B.8.14.2	Yes	TJB-288,TJB-289
B.8.14.2(a)	Yes	TJB306
B.8.14.2(b)	Yes	TJB-225,TJB-307
B.8.14.2(c)	Yes	TJB308
B.8.14.2(d.1)	Yes	TJB-252,TJB-253
B.8.14.2(d.2)	Yes	TJB-322
B.8.14.2(d.3)	Yes	TJB-323
B.8.14.2(e)	Yes	TJB-324
B.8.14.2(f)	Yes	TJB-220
B.8.14.2(g)	Yes	TJB-221
B.8.14.2(h)	Yes	TJB-254,TJB-275
B.8.14.2(i)	Yes	TJB-276
B.8.14.2(j)	Yes	TJB-255
B.8.14.2(k)	Yes	TJB-313

1149.1b-1993/94 Rule	Verified	Message(s)
B.8.14.2(l)	Yes	TJB-262
B.8.14.2(m)	Yes	TJB-223,TJB-224
B.8.14.2(n)	Yes	TJB-314
B.8.14.2(o)	Yes	TJB-278
B.8.14.2(p.1)	Yes	TJB-277
B.8.14.2(p.2)	Yes	TJB-277
B.8.14.2(q)	Yes	TJB2-48
B.8.14.2(q.1)	Yes	TJB-257
B.8.14.2(q.2)	Yes	TJB-258
B.8.14.2(q.3)	Yes	TJB-258
B.8.14.2(r)	Not Required	
B.8.14.2(s.1)	Yes	TJB-265
B.8.14.2(s.2)	Yes	TJB-266
B.8.14.2(s.2.2)	Yes	TJB-270
B.8.14.2(s.3)	Yes	TJB-267
B.8.14.2(s.3.2)	Yes	TJB-268
B.8.14.2(s.3.3)	Yes	TJB-271
B.8.14.2(s.4.1)	Yes	TJB-269
B.8.14.2(s.4.2)	Yes	TJB-272
B.8.14.2(s.4.3)	Yes	TJB-273
B.8.14.2(s.4.4)	Yes	TJB-274
B.8.14.2(s.4.5)	Yes	TJB-290
B.8.14.2(t)	No	
B.8.14.2(u)	No	
RUNBIST description		
B.8.15.3(a)	Yes	TJB-304
B.8.15.3(b)	No	

1149.1b-1993/94 Rule	Verified	Message(s)
B.8.15.3(b.1)	No	
B.8.15.3(b.2)	No	
B.8.15.3(c)	Yes	TJB-305
B.8.15.3(d)	No	
B.8.15.3(e)	No	
INTEST description		
B.8.16.3(a)-(d)	No	
User extensions to BSDL		
B.8.17.3(a)-(b)	No	
User-Supplied VHDL Packages		
B.10.1.3(a)	No	
B.10.1.3(b)	Yes	TJB-325
B.10.1.3(c)	No	
B.10.2.2.1	No	

Table 2-2 IEEE 1149.1 - 2001 BSDL Rules Verification Table

1149.1 - 2001 Rule	Verified	Message(s)
BSDL Components		
B.7(a)	Yes	TJB-102
B.7(b.1)	Yes	TJB-285
B.7(b.2)	No	
The entity description		
B.8.1.1	Yes	TJB-210
B.8.1.2(a)	No	
Generic Parameter Statement		
B.8.2.3(a.1)	No	
B.8.2.3(a.2)	No	

1149.1 - 2001 Rule	Verified	Message(s)
B.8.2.3(a.3)	No	
Logical port description statement		
B.8.3	Yes	TJB-216
B.8.3.3(a)	Yes	TJB-316
B.8.3.3(b)	Yes	TJB-249
Standard use statement		
B.8.4	Yes	TJB-285
B.8.4.1.3(a)	No	
B.8.4.2.2(a)	Yes	Various
Use statement		
B.8.5.3(a)	No	
B.8.5.3(b.1)	No	
B.8.5.3(b.2)	No	
Component conformance statement		
B.8.6.1	Yes	TJB-286
Device package pin mappings		
B.8.7.1	Yes	TJB-207
B.8.7.3(a)	Yes	TJB-264
B.8.7.3(c)	Yes	TJB-228
B.8.7.3(d)	Yes	TJB-228
B.8.7.3(e)	Yes	TJB-317
B.8.7.3(f)	Yes	TJB-218
Grouped port identification		
B.8.8.3(a)	Yes	TJB-250
B.8.8.3(b)	Yes	TJB-221
B.8.8.3(c)	Yes	TJB-256
B.8.8.3(d)	Yes	TJB-257

1149.1 - 2001 Rule	Verified	Message(s)
B.8.8.3(e)	Yes	TJB-319
B.8.8.3(f)	Yes	TJB-219
Scan port identification		
B.8.9	Yes	TJB-201
B.8.9.3(a.1)	Yes	TJB-247
B.8.9.3(a.2)	Yes	TJB-247
B.8.9.3(a.3)	No	
B.8.9.3(b)	Yes	TJB-258
B.8.9.3(c)	Yes	TJB-251
B.8.9.3(d)	Yes	TJB-221
B.8.9.3(e)	Yes	TJB-321
Compliance enable description		
B.8.10.3(a)	Yes	TJB-243
B.8.10.3(b)	Yes	TJB-245
B.8.10.3(c)	Yes	TJB-246
B.8.10.3(d.1)	Yes	TJB-218
B.8.10.3(d.2)	Yes	TJB-247
B.8.10.3(e)	Yes	TJB-258
B.8.10.3(f)	Yes	TJB-221
B.8.10.3(g)	Yes	TJB-241
Instruction register description		
B.8.11	Yes	TJB-287, TJB-238, TJB-239
B.8.11.3	Yes	TJB-208
B.8.11.3(a)	Yes	TJB-203
B.8.11.3(b)	Yes	TJB-209
B.8.11.3(c)	Yes	TJB-213
B.8.11.3(d)	Yes	TJB-212

1149.1 - 2001 Rule	Verified	Message(s)
B.8.11.3(e)	Yes	TJB-211
B.8.11.3(f)	Yes	TJB-214
B.8.11.3(g)	Yes	TJB-215
B.8.11.3(h.1)	Yes	TJB-214
B.8.11.3(h.2)	Yes	TJB-217
B.8.11.3(i)	Yes	TJB-326
B.8.11.3(j)	Yes	TJB-318
B.8.11.3(k)	Yes	TJB-204
B.8.11.3(k.2)	Yes	TJB-230
B.8.11.3(l)	Yes	TJB-303
B.8.11.3(m)	Yes	TJB-310
Optional register description		
B.8.12.3(a)	Yes	TJB-235
B.8.12.3(b)	Yes	TJB-312
B.8.12.3(b.2)	Yes	TJB-311
B.8.12.3(c)	Yes	TJB-237
B.8.12.3(d)	No	
Register access description		
B.8.13.3(a)	No	
B.8.13.3(b)	Yes	TJB-300
B.8.13.3(c)	Yes	TJB-301
B.8.13.3(d)	Yes	TJB-309
B.8.13.3(e)	No	
B.8.13.3(f)	Yes	TJB-302
Boundary scan chain description		
B.8.14.4	Yes	TJB-288,TJB-289
B.8.14.4(a)	Yes	TJB-306

1149.1 - 2001 Rule	Verified	Message(s)
B.8.14.4(b)	Yes	TJB-225,TJB-307
B.8.14.4(c)	Yes	TJB-308
B.8.14.4(d.1)	Yes	TJB-252,TJB-253
B.8.14.4(d.2)	Yes	TJB-322
B.8.14.4(d.3)	Yes	TJB-323
B.8.14.4(d.4)	No	
B.8.14.4(e)	Yes	TJB-324
B.8.14.4(f)	Yes	TJB-220
B.8.14.4(g)	Yes	TJB-221
B.8.14.4(h)	Yes	TJB-254,TJB-275
B.8.14.4(i)	Yes	TJB-276
B.8.14.4(j)	Yes	TJB-255
B.8.14.4(k.1)	Yes	TJB-313
B.8.14.4(k.2)	Yes	TJB-328
B.8.14.4(l)	Yes	TJB-328
B.8.14.4(m)	Yes	TJB-262
B.8.14.4(n)	Yes	TJB-314
B.8.14.4(o)	Yes	TJB-223 , TJB-224
B.8.14.4(p)	Yes	TJB-314
B.8.14.4(q)	Yes	TJB-278
B.8.14.4(r.1)	Yes	TJB-277
B.8.14.4(r.2)	Yes	TJB-277
B.8.14.4(s.1)	Yes	TJB-257
B.8.14.4(s.2)	Yes	TJB-258
B.8.14.4(s.3)	Yes	TJB-258
B.8.14.4(t)	Not required	
B.8.14.4(u.1)	Yes	TJB-265

1149.1 - 2001 Rule	Verified	Message(s)
B.8.14.4(u.2)	Yes	TJB-266
B.8.14.4(u.2.2)	Yes	TJB-270
B.8.14.4(u.3)	Yes	TJB-267
B.8.14.4(u.3.2)	Yes	TJB-268
B.8.14.4(u.3.3)	Yes	TJB-271
B.8.14.4(u.4.1)	Yes	TJB-269
B.8.14.4(u.4.2)	Yes	TJB-272
B.8.14.4(u.4.3)	Yes	TJB-273
B.8.14.4(u.4.4)	Yes	TJB-274
B.8.14.4(u.4.5)	Yes	TJB-290
B.8.14.4(v)	No	
B.8.14.4(w)	No	
RUNBIST description		
B.8.15.3(a.1)	Yes	TJB-304
B.8.15.3(a.2)	Yes	TJB-304
B.8.15.3(b)	No	
B.8.15.3(b.1)	No	
B.8.15.3(b.2)	No	
B.8.15.3(c)	Yes	TJB-305
B.8.15.3(d)	No	
B.8.15.3(e)	No	
INTEST description		
B.8.16.3(a)-(d)	No	
User extensions to BSDL		
B.8.17.3(a)-(b)	No	
User-Supplied VHDL Packages		
B.10.1.3(a)	No	

1149.1 - 2001 Rule	Verified	Message(s)
B.10.1.3(b)	Yes	TJB-325
B.10.1.3(c)	No	
B.10.2.2.1	No	

IEEE 1149.1 Rules Verification Tables

Table 2-3 shows the rules from IEEE 1149.1 Std 1149.1-1990 and IEEE 1149.1 Std 1149.1a-1993 verified by Encounter Test and if verified, corresponding message(s), and associated APN and/or applicable BSDL semantic check that is performed.

Table 2-4 on page 170 shows equivalent information for the IEEE 1149.1 - 2001 standard.

Refer to the IEEE 1149.1 Standard and to “1149.1 BSV Verification Procedures (APNs)” on page 103.

Table 2-3 IEEE 1149.1 - 1990/1993 Rules Verification Table

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
Test Access Port			
3.1.1(a)	Yes	TJB-201,TTM-062	B.8.9
3.1.1(b)	No		
3.1.1(c)	Yes	TJB-258	B.8.9.3(b)
3.2.1	No		
3.3.1(a)	Yes	Various	Collectively Verified
3.3.1(b	No		
3.4.1(a)	Yes	Various	Collectively Verified
3.4.1(b)	No		
3.5.1(a)	Yes	Various	Collectively Verified
3.5.1(b)	Yes	TJC-217	Collectively Verified
3.6.1(a)	Yes	TJC-211	APN: 1 - Asynchronous Reset

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
3.6.1(b)	No		
3.6.1(c)	No		
3.8.1(a)	Yes	TJC-211	APN: 1 - Asynchronous Reset
	Yes	TJC-210	APN: 2 - Synchronous Reset.
3.8.1(b)	Yes	TJC-211	APN: 1 - Asynchronous Reset
	Yes	TJC-210	APN: 2 - Synchronous Reset
	Yes	TJC-141	
3.8.1(c)	No		
3.8.1(d)	No		
3.8.1(e)	Yes	TJB-241	B.8.10.3(g)
		TJB-258	B.8.10.3(e)
		TJB-117 -TJB-120	BSDL vs BSV test mode consistency checks
Test Logic Architecture			
4.1.1(a.i)	Yes	TJB-201,TTM-062	B.8.9
4.1.1(a.ii)	Yes	Various	Collectively Verified (re: 5.1.1(a))
4.1.1(a.iii)	Yes	Various	Collectively Verified (re: 6.1.1(a))
4.1.1(a.iv)	Yes	TJB-289	Re: 8.1.1(a)
4.2.1(a)	No		
4.2.1(b)	No		
TAP Controller			
5.1.1(a)	Yes		
TLR/0	Yes	TJC2-10	APN: 3 - Test Logic Reset Zero Transition - Reset/0
TLR/1	Yes	TJC-210	APN: 2 - Synchronous Reset
RTI/0	Yes	TJC-209	APN: 21 - Run Test Idle Zero Transition - RTI/0

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
RTI/1	Yes	TJC-214	APN: 19 - UpdateDR Zero Transition
SelectIR/0	Yes	TJC-206	APN: 6 - Instruction Register Capture and Shift
SelectIR/1	Yes	TJC-210	APN: 4 - SelectIR One Transition - SelectIR/1
CaptureIR/0	Yes	TJC-206	APN: 6 - Instruction Register Capture and Shift
CaptureIR/1	Yes	TJC-216	APN: 8 - Instruction Register Capture Without Immediate Shift
ShiftIR/0	Yes	TJC-207	APN: 7 - Instruction Shift without Capture
ShiftIR/1	Yes	TJC-207	APN: 7 - Instruction Shift without Capture
Exit1IR/0	Yes	TJC-207	APN: 7 - Instruction Shift without Capture
Exit1IR/1	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
PauseIR/0	Yes	TJC-207	APN: 9 - PauseIR Zero Transition - PAUSEIR/0
PauseIR/1	Yes	TJC-207	APN: 7 - Instruction Shift without Capture
Exit2IR/0	Yes	TJC-207	APN: 7 - Instruction Shift without Capture
Exit2IR/1	Yes	TJC-209	APN: 10 - Exit2IR One Transition - Exit2IR/1
UpdateIR/0	Yes	TJC-209	APN: 11 - UpdateIR Zero Transition - UpdateIR/0
UpdateIR/1	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
SelectDR/0	Yes	TJC-209	APN: 14 - Data Register Capture and Shift

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
SelectDR/1	Yes	TJC-206	APN: 6 - Instruction Register Capture and Shift
CaptureDR/0	Yes	TJC-209	APN: 14 - Data Register Capture and Shift
CaptureDR/1	Yes	TJC-218	APN: 16 - Data Register Capture Without Immediate Shift
ShiftDR/0	Yes	TJC-213	APN: 15 - Data Register Shift Without Capture
ShiftDR/1	Yes	TJC-213	APN: 15 - Data Register Shift Without Capture
Exit1DR/0	Yes	TJC-213	APN: 15 - Data Register Shift Without Capture
Exit1DR/1	Yes	TJC-214	APN: 18 - Data Register Update
PauseDR/0	Yes	TJC-213	APN: 17 - PauseDR Zero Transition
PauseDR/1	Yes	TJC-213	APN: 15 - Data Register Shift Without Capture
Exit2DR/0	Yes	TJC-213	APN: 15 - Data Register Shift Without Capture
Exit2DR/1	Yes	TJC-214	APN: 20 - Exit2DR One Transition
UpdateDR/0	Yes	TJC-214	APN: 19 - UpdateDR Zero Transition
UpdateDR/1	Yes	TJC-214	APN: 18 - Data Register Update
5.1.1(b)	Yes	Various	Collectively Verified (re: 5.1.1(a))
5.1.1(c)	Yes	Various	Partially Collectively Verified (re: 5.1.1(a) TDO edge expects)
5.2.1(a.i)	Yes	Various	Collectively Verified (re: 5.1.1(a))
5.2.1(a.ii)	Yes	TJC-211	APN: 1 - Asynchronous Reset
5.2.1(a.iii)	No		
5.2.1(b)	Yes	Various	Collectively Verified

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
5.2.1(c)	Yes	Various	Collectively Verified (re: 5.1.1(a) TDOexpects)
5.2.1(d)	Yes	Various	Collectively Verified (re: 5.1.1(a) TDO edge expects
5.3.1(a)	No		
5.3.1(b)	No		
5.3.1(c)	Yes	TJC-211	APN: 1 - Asynchronous Reset
5.3.1(d)	No		
Instruction Register			
6.1.1(a)	Yes	TJB-203	B.8.11.3(a)
		TJC-209	APN: 12 - Instruction Decode-Data Register Access
		TJC-215	APN: 5 - Instruction Register Shift
6.1.1(b)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
	Yes	TJC-211	APN: 1 - Asynchronous Reset
	Yes	TJC-210	APN: 2 - Synchronous Reset
6.1.1(c)	Yes	TJC-215	APN: 5 - Instruction Register Shift
6.1.1(d)	Yes		8.11.3(g.2)
		TJC-206	APN: 6 - Instruction Register Capture and Shift
6.2.1(a)	Yes	TJC-210	APN: 4 - SelectIR One Transition - SelectIR/1
		TJC-206	APN: 6 - Instruction Register Capture and Shift
		TJC-207	APN: 7 - Instruction Shift without Capture
		TJC-209	APN: 12 - Instruction Decode-Data Register Access

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
6.2.1(b)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
	Yes	TJC-210	APN: 4 - SelectIR One Transition - SelectIR/1
6.2.1(c)	Yes	TJC-206	APN: 6 - Instruction Register Capture and Shift.
6.2.1(d)	Yes	TJC-209	APN: 10 - Exit2IR One Transition - Exit2IR/1
6.2.1(e)	Yes	TJC-210	APN: 2 - Synchronous Reset
6.2.1(f)	Yes	TJC-211	APN: 1 - Asynchronous Reset
Instructions			
7.1.1(a)	No		
7.1.1(b)	No		
7.1.1(c)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
7.1.1(d)	No		
7.2.1(a)	No		
7.2.1(b)	Yes	TJB-213	B.8.11.3(c) (`BYPASS`)
		TJB-214	B.8.11.3(e) (`SAMPLE`)
		TJC-211	B.8.11.3(d) (`EXTEST`)
7.2.1(c)	Yes	TJB-312	B.8.12.3(b)
		TJC2-10	APN: 2 - Synchronous Reset
		TJC-211	APN: 1 - Asynchronous Reset
7.2.1(d)	No		
7.2.1(e)	Yes	TJB-211	B.8.11.3(d) (`EXTEST`)
		TJB-213	B.8.11.3(c) (`BYPASS`)
7.3.1(a)	No		
7.3.1(b)	No		

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
7.3.1(c)	No		
7.4.1(a)	Yes	TJB-213	B.8.11.3 (BYPASS)
7.4.1(b)	Yes	TJB-213	B.8.11.3(c) (BYPASS)
7.4.1(c)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
7.4.1(d)	No		
7.4.1(e)	No		
7.6.1(a)	Yes	TJB-214	B.8.11.3(e)
7.6.1(b)	Yes	TJC-212	APN: 13 - Data Register Shift
7.6.1(c)	No		
7.6.1(d)	Yes	TJC-320	APN: 23 - SAMPLE BdyReg Cap all Syst In Pins
7.6.1(e)	Yes	TJC-310	APN: 22 - PRELOAD
7.7.1(a)	Yes	TJB-211	B.8.11.3 (EXTEST)
7.7.1(b)	Yes	TJB-211	B.8.11.3(d) (EXTEST)
7.7.1(c)	Yes	TJC-212	APN: 13 - Data Register Shift
	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
7.7.1(d)	No		
7.7.1(e)	Yes	TJC-310	APN: 27 - Boundary Register - Enable All System Output Drivers
	Yes	TJC-322	APN: 28 - Boundary Register - Disable All System Output Drivers
	Yes	TJC-311	APN: 29 - Boundary Register - All System Output Drivers Retain Active State
7.7.1(f)	Yes	TJC-320	APN: 40 - Capture All System Input Pins
7.8.1(a)	No		

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
7.8.1(b)	No		
7.8.1(c.i)	No		
7.8.1(c.ii)	No		
7.8.1(d)	No		
7.8.1(e)	No		
7.9.1(a)	Yes	TJC-212	APN: 13 - Data Register Shift
7.9.1(b)	No		
7.9.1(c)	No		
7.9.1(d)	Yes	TJB-806	BSDL syntax error
7.9.1(e)	No		
7.9.1(f)	No		
7.9.1(g)	No		
7.9.1(h)	No		
7.9.1(i)	No		
7.9.1(j)	No		
7.9.1(k.i)	Yes	TJC-310	APN: 44 - RUNBIST - All System Output Pins Active
7.9.1(k.ii)	Yes	TJC-327	APN: 43 - RUNBIST - All System Output Pins Inactive
7.9.1(l)	Yes	TJC-311	APN: 44 - RUNBIST - All System Output Pins Active.
7.10.1(a)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
7.10.1(b)	Yes	TJC-310	APN: 42 - CLAMP - All System Output Pins Active
7.10.1(c)	Yes	TJC-311	APN: 42 - CLAMP - All System Output Pins Active
7.10.1(d)	No		
7.12.1(a)	Yes	TJB-311	B.8.12.3(b.2)

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
		TJB-312	B.8.12.3(b)
		TJC-211	APN: 1 - Asynchronous Reset
		TJC-210	APN: 2 - Synchronous Reset
7.12.1(b)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access.
7.12.1(c)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
7.12.1(d)	No		
7.12.1(e)	No		
7.13.1(a)	No		
7.13.1(b)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
7.13.1(c)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
7.13.1(d)	No		
7.13.1(e)	No		
7.14.1(a)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
7.14.1(b)	Yes	TJC-327,TJC-328	APN: 41 - All System Output Pins Inactive
7.14.1(c)	No		
Test Data Registers			
8.1.1(a)	Yes	TJB-289	B.8.14.2 (Boundary Register)
	Yes	TJB-209	APN: 12 - Instruction Decode-Data Register Access
8.1.1(b)	Yes	Various	Per `DEVICE_ID` checks
8.1.1(c)	Yes	Various	Collectively Verified
8.2.1(a)	Yes	TJB-301	B.8.13.3(c)
8.2.1(b)	Yes	TJC-293	BSV Structure Check

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
	Yes	TJC-212	dapn13.
8.2.1(c)	Yes	TJC-209	dapn12.
8.2.1(d)	No		
8.3.1(a)	Yes	TJB-300	B.8.13.3(b)
8.3.1(b)	Yes	TJC-289	BSV Structure Check
	Yes	TJC-212	APN: 13 - Data Register Shift
8.3.1(c)	No		
8.3.1(d)	Yes	TJC-209	APN: 14 - Data Register Capture and Shift
8.3.1(e.i)	Yes	TJC-310	APN: 27 - Boundary Register - Enable All System Output Drivers
		UpdateIR, Various	Highz, clamp, runbist, iomapping output pin checks
8.3.1(e.ii)	No		Collectively Verified
8.3.1(f)	No		
8.3.1(g)	No		
8.3.1(h)	No		
BYPASS Register			
9.1.1(a)	Yes	TJC-212	APN: 13 - Data Register Shift
9.1.1(b)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
9.1.1(c)	No		
9.1.1(d)	No		
Boundary Register			
10.2.1(a)	Yes	Various	Collectively Verified
10.2.1(b)	Yes	TJB-248	B.8.14.2(q), Collectively Verified
	Yes	Various	B.8.14.2(s)
10.2.1(c)	Yes	TJC-212	APN: 13 - Data Register Shift

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
10.2.1(d)	No		
10.2.1(e)	No		
10.2.1(f)	No		
10.2.1(g)	No		
10.2.1(h)	No		
10.2.1(i)	Yes	TJB-306	B.8.14.2(a)
	Yes	TJC-212	APN: 13 - Data Register Shift
10.2.1(j)	No		
10.2.1(k)	No		
10.3.1(a)	Yes	TJC-212	APN: 13 - Data Register Shift
10.3.1(b)	Yes	TJC-310	APN: 27 - Boundary Register - Enable All System Output Drivers
10.3.1(c.i)	No		
10.3.1(c.ii)	No		
10.3.1(d)	No		
10.3.1(e)	No		
10.4.1(a)	Yes	TJB-248	B.8.14.2(q)
10.4.1(b)	No		
10.4.1(c.i)	Yes	TJB-258	B.8.9.3(b)
10.4.1(c.ii)	Yes	TJB-258	B.8.10.3(e)
10.4.1(c.iii)	Yes	TJB-277	B.8.14.2(p)
10.4.1(d)	No		
10.4.1(e)	No		
10.5.1(a)	Yes	TJC-300	APN: 24 - Boundary Register I/O Mapping - System Input Pin.
		TJB-248	B.8.14.2(q)
10.5.1(b)	Yes	TJC-300	APN: 24 - Boundary Register I/O Mapping - System Input Pin

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
10.5.1(c)	No		
10.5.1(d)	No		
10.5.1(e.i)	No		
10.5.1(e.ii)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin.
10.5.1(e.iii)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin
10.5.1(f)	Yes	TJC-320	APN: 23 - SAMPLE
		TJC-300	APN: 24 - Boundary Register I/O Mapping - System Input Pin
10.5.1(g)	No		
10.5.1(h)	No		
10.5.1(i)	Yes	TJC-320	APN: 23 - SAMPLE
		TJC-300	APN: 24 - Boundary Register I/O Mapping - System Input Pin
10.5.1(j)	No		
10.5.1(k)	No		
10.6.1(a)	Yes	TJB-248	B.8.14.2(q)
		TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin
10.6.1(b)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin
		TJB-270 -TJB-274	B.8.14.2(s.2) - B.8.14.2(s.4)
10.6.1(c)	Yes	TJC-350	BSV Structure Check.
	Yes	TJC-351	BSV Structure Check.
	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin
		TJB-255,TJB-313	B.8.14.2(j), B.8.14.2(k)

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
10.6.1(d.i),(d.ii)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin
		TJC-304	APN: 26 - Boundary Register I/O Mapping - Share Enable
		TJC-304	APN: 31 - Boundary Register I/O Mapping - Share Disable
		TJC-310	APN: 27 - Boundary Register - Enable All System Output Drivers
10.6.1(e)(f)	Yes	TJC-304	APN: 26 - Boundary Register I/O Mapping - Share Enable
		TJC-304	APN: 31 - Boundary Register I/O Mapping - Share Disable
		TJC-310	APN: 27 - Boundary Register - Enable All System Output Drivers
		TJC-322	APN: 28 - Boundary Register - Disable All System Output Drivers
		TJB-314	B.8.14.2(n)
10.6.1(g)	Yes	TJC-310,TJC-311	APN: 42 - CLAMP - All System Output Pins Active
	Yes	Various	Various `EXTEST` checks
	Yes	TJC-327,TJC-328	APN: 41 - All System Output Pins Inactive
INTEST	No		
	Yes	TJC-310	APN: 44 - RUNBIST - All System Output Pins Active
	Yes	TJC-327	APN: 43 - RUNBIST - All System Output Pins Inactive
	Yes	TJC-310	APN: 22 - PRELOAD
BYPASS,IDCODE,USERCODE	No		

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
10.6.1(h)	Yes	TJC-327,TJC-328	APN: 41 - All System Output Pins Inactive
INTEST	No		
	Yes	TJC-327	APN: 43 - RUNBIST - All System Output Pins Inactive
10.6.1(i)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin
10.6.1(j)	Yes	TJC-310	APN: 22 - PRELOAD
	Yes	TJC-310	APN: 27 - Boundary Register - Enable All System Output Drivers
	Yes	TJC-322	APN: 28 - Boundary Register - Disable All System Output Drivers
INTEST	No		
10.6.1(k)	Yes	TJC-310	APN: 42 - CLAMP - All System Output Pins Active
	Yes	TJC-310	APN: 44 - RUNBIST - All System Output Pins Active
	Yes	TJC-327,TJC-328	APN: 41 - All System Output Pins Inactive
10.6.1(l)	No		
10.6.1(m)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin for data cell
10.6.1(n)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin for control cell
10.6.1(o)	No		
10.7.1(a.i)	Yes	TJC-300	APN: 24 - Boundary Register I/O Mapping - System Input Pin
10.7.1(a.ii)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin.

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
10.7.1(a.iii)	Yes	TJC-300	APN: 24 - Boundary Register I/O Mapping - System Input Pin
10.8.1(a)	No		
Device Identification Register			
11.1.1(a)	No		
11.1.1(b)	No		
11.1.1(c)	Yes	TJC-209	APN: 14 - Data Register Capture and Shift
11.1.1(d)	No		
11.1.1(e)	Yes	TJC-209	APN: 14 - Data Register Capture and Shift
11.1.1(f)	No		
11.2.1(a.i)	No		
11.2.1(a.ii)	No		
11.2.1(b)	No		
11.3.1(a)	No		
11.3.1(b)	No		
11.4.1(a)	No		
Conformance and Documentation			
12.1.1(a)	No		
12.1.1(b)	No		
12.2.1(a)	No		
12.3.1(a)	No		
12.3.1(b)	Yes	Various - BSDL messages	Various

Table 2-4 IEEE 1149.1 - 2001 Rules Verification Table

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
Test Access Port			
4.1.1(a)	Yes	TJB-201,TTM-062	B.8.9
4.1.1(b)	No		
4.1.1(c)	Yes	TJB-258	B.8.9.3(b)
4.2.1	No		
4.3.1(a)	Yes	Various	Collectively Verified
4.3.1(b	No		
4.4.1(a)	Yes	Various	Collectively Verified
4.4.1(b)	No		
4.5.1(a)	Yes	Various	Collectively Verified
4.5.1(b)	Yes	TJC-217	Collectively Verified
4.6.1(a)	Yes	TJC-211	APN: 1 - Asynchronous Reset
4.6.1(b)	No		
4.6.1(c)	No		
4.8.1(a)	Yes	TJC-211	APN: 1 - Asynchronous Reset
	Yes	TJC-210	APN: 2 - Synchronous Reset.
4.8.1(b)	Yes	TJC-211	APN: 1 - Asynchronous Reset
	Yes	TJC-210	APN: 2 - Synchronous Reset
	Yes	TJC-141	
4.8.1(c)	No		
4.8.1(d)	No		
4.8.1(e)	Yes	TJB-241	B.8.10.3(g)
		TJB-258	B.8.10.3(e)
		TJB-117 -TJB-120	BSDL vs BSV test mode consistency checks

Rule	**Verified**	**Message(s)**	**APN/1149.1 Semantic Check/ Structure Check**
Test Logic Architecture			
5.1.1(a.i)	Yes	TJB-201,TTM-062	B.8.9
5.1.1(a.ii)	Yes	Various	Collectively Verified (re: 5.1.1(a))
5.1.1(a.iii)	Yes	Various	Collectively Verified (re: 6.1.1(a))
5.1.1(a.iv)	Yes	TJB-289	Re: 8.1.1(a)
5.2.1(a)	No		
5.2.1(b)	No		

TAP Controller			
6.1.1(a)	Yes		
TLR/0	Yes	TJC-210	APN: 3 - Test Logic Reset Zero Transition - Reset/0
TLR/1	Yes	TJC-210	APN: 2 - Synchronous Reset
RTI/0	Yes	TJC-209	APN: 21 - Run Test Idle Zero Transition - RTI/0
RTI/1	Yes	TJC-214	APN: 19 - UpdateDR Zero Transition
SelectIR/0	Yes	TJC-206	APN: 6 - Instruction Register Capture and Shift
SelectIR/1	Yes	TJC-210	APN: 4 - SelectIR One Transition - SelectIR/1
CaptureIR/0	Yes	TJC-206	APN: 6 - Instruction Register Capture and Shift
CaptureIR/1	Yes	TJC-216	APN: 8 - Instruction Register Capture Without Immediate Shift
ShiftIR/0	Yes	TJC-207	APN: 7 - Instruction Shift without Capture
ShiftIR/1	Yes	TJC-207	APN: 7 - Instruction Shift without Capture

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
Exit1IR/0	Yes	TJC-207	APN: 7 - Instruction Shift without Capture
Exit1IR/1	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
PauseIR/0	Yes	TJC-207	APN: 9 - PauseIR Zero Transition - PAUSEIR/0
PauseIR/1	Yes	TJC-207	APN: 7 - Instruction Shift without Capture
Exit2IR/0	Yes	TJC-207	APN: 7 - Instruction Shift without Capture
Exit2IR/1	Yes	TJC-209	APN: 10 - Exit2IR One Transition - Exit2IR/1
UpdateIR/0	Yes	TJC-209	APN: 11 - UpdateIR Zero Transition - UpdateIR/0
UpdateIR/1	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
SelectDR/0	Yes	TJC-209	APN: 14 - Data Register Capture and Shift
SelectDR/1	Yes	TJC-206	APN: 6 - Instruction Register Capture and Shift
CaptureDR/0	Yes	TJC-209	APN: 14 - Data Register Capture and Shift
CaptureDR/1	Yes	TJC-218	APN: 16 - Data Register Capture Without Immediate Shift
ShiftDR/0	Yes	TJC-213	APN: 15 - Data Register Shift Without Capture
ShiftDR/1	Yes	TJC-213	APN: 15 - Data Register Shift Without Capture
Exit1DR/0	Yes	TJC-213	APN: 15 - Data Register Shift Without Capture
Exit1DR/1	Yes	TJC-214	APN: 18 - Data Register Update
PauseDR/0	Yes	TJC-213	APN: 17 - PauseDR Zero Transition

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
PauseDR/1	Yes	TJC-213	APN: 15 - Data Register Shift Without Capture
Exit2DR/0	Yes	TJC-213	APN: 15 - Data Register Shift Without Capture
Exit2DR/1	Yes	TJC-214	APN: 20 - Exit2DR One Transition
UpdateDR/0	Yes	TJC-214	APN: 19 - UpdateDR Zero Transition
UpdateDR/1	Yes	TJC-214	APN: 18 - Data Register Update
6.1.1(b)	Yes	Various	Collectively Verified (re: 5.1.1(a))
6.1.1(c)	Yes	Various	Partially Collectively Verified (re: 5.1.1(a) TDO edge expects)
6.2.1(a.i)	Yes	Various	Collectively Verified (re: 5.1.1(a))
6.2.1(a.ii)	Yes	TJC-211	APN: 1 - Asynchronous Reset
6.2.1(a.iii)	No		
6.2.1(b)	Yes	Various	Collectively Verified
6.2.1(c)	Yes	Various	Collectively Verified (re: 5.1.1(a) TDOexpects)
6.2.1(d)	Yes	Various	Collectively Verified (re: 5.1.1(a) TDO edge expects
6.3.1(a)	No		
6.3.1(b)	No		
6.3.1(c)	Yes	TJC-211	APN: 1 - Asynchronous Reset
6.3.1(d)	No		
Instruction Register			
7.1.1(a)	Yes	TJB-203	B.8.11.3(a)
		TJC-209	APN: 12 - Instruction Decode-Data Register Access
		TJC-215	APN: 5 - Instruction Register Shift

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
7.1.1(b)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
	Yes	TJC-211	APN: 1 - Asynchronous Reset
	Yes	TJC-210	APN: 2 - Synchronous Reset
7.1.1(c)	Yes	TJC-215	APN: 5 - Instruction Register Shift
7.1.1(d)	Yes		8.11.3(g.2)
		TJC-206	APN: 6 - Instruction Register Capture and Shift
7.2.1(a)	Yes	TJC-210	APN: 4 - SelectIR One Transition - SelectIR/1
		TJC-206	APN: 6 - Instruction Register Capture and Shift
		TJC-207	APN: 7 - Instruction Shift without Capture
		TJC-209	APN: 12 - Instruction Decode-Data Register Access
7.2.1(b)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
	Yes	TJC-210	APN: 4 - SelectIR One Transition - SelectIR/1
7.2.1(c)	Yes	TJC-206	APN: 6 - Instruction Register Capture and Shift.
7.2.1(d)	Yes	TJC-209	APN: 10 - Exit2IR One Transition - Exit2IR/1
7.2.1(e)	Yes	TJC-210	APN: 2 - Synchronous Reset
7.2.1(f)	Yes	TJC-211	APN: 1 - Asynchronous Reset
Instructions			
8.1.1(a)	No		
8.1.1(b)	No		

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
8.1.1(c)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
8.1.1(d)	No		
8.2.1(a)	No		
8.2.1(b)	Yes	TJB-213	B.8.11.3(c) (`BYPASS`)
		TJB-214	B.8.11.3(e) (`SAMPLE`)
		TJC-211	B.8.11.3(d) (`EXTEST`)
8.2.1(c)	Yes	TJB-312	B.8.12.3(b)
		TJC-210	APN: 2 - Synchronous Reset
		TJC-211	APN: 1 - Asynchronous Reset
8.2.1(d)	No		
8.2.1(e)	Yes	TJB-211	B.8.11.3(d) (`EXTEST`)
		TJB-213	B.8.11.3(c) (`BYPASS`)
8.3.1(a)	No		
8.3.1(b)	No		
8.3.1(c)	No		
8.4.1(a)	Yes	TJB-213	B.8.11.3 (`BYPASS`)
8.4.1(b)	Yes	TJB-213	B.8.11.3(c) (`BYPASS`)
8.4.1(c)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
8.4.1(d)	No		
8.4.1(e)	No		
8.6.1(a)	Yes	TJB-214	B.8.11.3(e)
8.6.1(b)	Yes	TJC-212	APN: 13 - Data Register Shift
8.6.1(c)	No		
8.6.1(d)	Yes	TJC-320	APN: 23 - SAMPLE BdyReg Cap all Syst In Pins
8.6.1(e)	Yes	TJC-310	APN: 22 - PRELOAD

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
8.7	Yes		Preload Instruction
8.8.1(a)	Yes	TJB-211	B.8.11.3 (`EXTEST`)
8.8.1(b)	Yes	TJB-211	B.8.11.3(d) (`EXTEST`)
8.8.1(c)	Yes	TJC-212	APN: 13 - Data Register Shift
	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
8.8.1(d)	No		
8.8.1(e)	Yes	TJC-310	APN: 27 - Boundary Register - Enable All System Output Drivers
	Yes	TJC-322	APN: 28 - Boundary Register - Disable All System Output Drivers
	Yes	TJC-311	APN: 29 - Boundary Register - All System Output Drivers Retain Active State
8.8.1(f)	Yes	TJC-320	APN: 40 - Capture All System Input Pins
8.9.1(a)	No		
8.9.1(b)	No		
8.9.1(c.i)	No		
8.9.1(c.ii)	No		
8.9.1(d)	No		
8.9.1(e)	No		
8.10.1(a)	Yes	TJC-212	APN: 13 - Data Register Shift
8.10.1(b)	No		
8.10.1(c)	No		
8.10.1(d)	Yes	TJB-806	BSDL syntax error
8.10.1(e)	No		
8.10.1(f)	No		
8.10.1(g)	No		

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
8.10.1(h)	No		
8.10.1(i)	No		
8.10.1(j)	No		
8.10.1(k.i)	Yes	TJC-310	APN: 44 - RUNBIST - All System Output Pins Active
8.10.1(k.ii)	Yes	TJC-327	APN: 43 - RUNBIST - All System Output Pins Inactive
8.10.1(l)	Yes	TJC-311	APN: 44 - RUNBIST - All System Output Pins Active.
8.11	Yes		CLAMP Instruction
8.12.1(a)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
8.12.1(b)	Yes	TJC-310	APN: 42 - CLAMP - All System Output Pins Active
8.12.1(c)	Yes	TJC-311	APN: 42 - CLAMP - All System Output Pins Active
8.12.1(d)	No		
8.13.1(a)	Yes	TJB-311	B.8.12.3(b.2)
		TJB-312	B.8.12.3(b)
		TJC-211	APN: 1 - Asynchronous Reset
		TJC-210	APN: 2 - Synchronous Reset
8.13.1(b)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access.
8.13.1(c)	Yes	TJC209	APN: 12 - Instruction Decode-Data Register Access
8.13.1(d)	No		
8.13.1(e)	No		
8.14.1(a)	No		
8.14.1(b)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
8.14.1(c)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
8.14.1(d)	No		
8.14.1(e)	No		
8.15.1(a)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
8.15.1(b)	Yes	TJC-327,TJC-328	APN: 41 - All System Output Pins Inactive
8.15.1(c)	No		
Test Data Registers			
9.1.1(a)	Yes	TJB-289	B.8.14.2 (Boundary Register)
	Yes	TJB-209	APN: 12 - Instruction Decode-Data Register Access
9.1.1(b)	Yes	Various	Per `DEVICE_ID` checks
9.1.1(c)	Yes	Various	Collectively Verified
9.2.1(a)	Yes	TJB-301	B.8.13.3(c)
9.2.1(b)	Yes	TJC-293	BSV Structure Check
	Yes	TJC-212	dapn13.
9.2.1(c)	Yes	TJC-209	dapn12.
9.2.1(d)	No		
9.3.1(a)	Yes	TJB-300	B.8.13.3(b)
9.3.1(b)	Yes	TJC-289	BSV Structure Check
	Yes	TJC-212	APN: 13 - Data Register Shift
9.3.1(c)	No		
9.3.1(d)	Yes	TJC-209	APN: 14 - Data Register Capture and Shift
9.3.1(e.i)	Yes	TJC-310	APN: 27 - Boundary Register - Enable All System Output Drivers

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
		UpdateIR, Various	Highz, clamp, runbist, iomapping output pin checks
9.3.1(e.ii)	No		Collectively Verified
9.3.1(f)	No		
9.3.1(g)	No		
9.3.1(h)	No		
BYPASS Register			
10.1.1(a)	Yes	TJC-212	APN: 13 - Data Register Shift
10.1.1(b)	Yes	TJC-209	APN: 12 - Instruction Decode-Data Register Access
10.1.1(c)	No		
109.1.1(d)	No		
Boundary Register			
11.2.1(a)	Yes	Various	Collectively Verified
11.2.1(b)	Yes	TJB-248	B.8.14.2(q), Collectively Verified
	Yes	Various	B.8.14.2(s)
11.2.1(c)	Yes	TJC-212	APN: 13 - Data Register Shift
11.2.1(d)	No		
11.2.1(e)	No		
11.2.1(f)	No		
11.2.1(g)	No		
11.2.1(h)	No		
11.2.1(i)	Yes	TJB-306	B.8.14.2(a)
	Yes	TJC-212	APN: 13 - Data Register Shift
11.2.1(j)	No		
11.2.1(k)	No		
11.3.1(a)	Yes	TJC-212	APN: 13 - Data Register Shift

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
11.3.1(b)	Yes	TJC-310	APN: 27 - Boundary Register - Enable All System Output Drivers
11.3.1(c.i)	No		
11.3.1(c.ii)	No		
11.3.1(d)	No		
11.3.1(e)	No		
11.4.1(a)	Yes	TJB-248	B.8.14.2(q)
11.4.1(b)	No		
11.4.1(c.i)	Yes	TJB-258	B.8.9.3(b)
11.4.1(c.ii)	Yes	TJB-258	B.8.10.3(e)
11.4.1(c.iii)	Yes	TJB-277	B.8.14.2(p)
11.4.1(d)	No		
11.4.1(e)	No		
11.5.1(a)	Yes	TJC-300	APN: 24 - Boundary Register I/O Mapping - System Input Pin.
		TJB-248	B.8.14.2(q)
11.5.1(b)	Yes	TJC-300	APN: 24 - Boundary Register I/O Mapping - System Input Pin
11.5.1(c)	No		
11.5.1(d)	No		
11.5.1(e.i)	No		
11.5.1(e.ii)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin.
11.5.1(e.iii)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin
11.5.1(f)	Yes	TJC-320	APN: 23 - SAMPLE
		TJC-300	APN: 24 - Boundary Register I/O Mapping - System Input Pin
11.5.1(g)	No		

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
11.5.1(h)	No		
11.5.1(i)	Yes	TJC-320	APN: 23 - SAMPLE
		TJC-300	APN: 24 - Boundary Register I/O Mapping - System Input Pin
11.5.1(j)	No		
11.5.1(k)	No		
11.6.1(a)	Yes	TJB-248	B.8.14.2(q)
		TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin
11.6.1(b)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin
		TJB-270 -TJB-274	B.8.14.2(s.2) - B.8.14.2(s.4)
11.6.1(c)	Yes	TJC-350	BSV Structure Check.
	Yes	TJC-351	BSV Structure Check.
	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin
		TJB-255,TJB-313	B.8.14.2(j), B.8.14.2(k)
11.6.1(d.i),(d.ii)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin
		TJC-304	APN: 26 - Boundary Register I/O Mapping - Share Enable
		TJC-304	APN: 31 - Boundary Register I/O Mapping - Share Disable
		TJC-310	APN: 27 - Boundary Register - Enable All System Output Drivers
11.6.1(e)(f)	Yes	TJC-304	APN: 26 - Boundary Register I/O Mapping - Share Enable
		TJC-304	APN: 31 - Boundary Register I/O Mapping - Share Disable

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
		TJC-310	APN: 27 - Boundary Register - Enable All System Output Drivers
		TJC-322	APN: 28 - Boundary Register - Disable All System Output Drivers
		TJB-314	B.8.14.2(n)
11.6.1(g)	Yes	TJC-310,TJC-311	APN: 42 - CLAMP - All System Output Pins Active
	Yes	Various	Various `EXTEST` checks
	Yes	TJC-327,TJC-328	APN: 41 - All System Output Pins Inactive
INTEST	No		
	Yes	TJC-310	APN: 44 - RUNBIST - All System Output Pins Active
	Yes	TJC-327	APN: 43 - RUNBIST - All System Output Pins Inactive
	Yes	TJC-310	APN: 22 - PRELOAD
BYPASS,IDCODE,USERCODE	No		
11.6.1(h)	Yes	TJC-327,TJC-328	APN: 41 - All System Output Pins Inactive
INTEST	No		
	Yes	TJC-327	APN: 43 - RUNBIST - All System Output Pins Inactive
11.6.1(i)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin
11.6.1(j)	Yes	TJC-310	APN: 22 - PRELOAD
	Yes	TJC-310	APN: 27 - Boundary Register - Enable All System Output Drivers
	Yes	TJC-322	APN: 28 - Boundary Register - Disable All System Output Drivers
INTEST	No		

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
11.6.1(k)	Yes	TJC-310	APN: 42 - CLAMP - All System Output Pins Active
	Yes	TJC-310	APN: 44 - RUNBIST - All System Output Pins Active
	Yes	TJC-327,TJC3-28	APN: 41 - All System Output Pins Inactive
11.6.1(l)	No		
11.6.1(m)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin for data cell
11.6.1(n)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin for control cell
11.6.1(o)	No		
11.7.1(a.i)	Yes	TJC-300	APN: 24 - Boundary Register I/O Mapping - System Input Pin
11.7.1(a.ii)	Yes	TJC-302	APN: 25 - Boundary Register I/O Mapping - System Output Pin.
11.7.1(a.iii)	Yes	TJC-300	APN: 24 - Boundary Register I/O Mapping - System Input Pin
11.8.1(a)	No		
Device Identification Register			
12.1.1(a)	No		
11.1.1(b)	No		
12.1.1(c)	Yes	TJC-209	APN: 14 - Data Register Capture and Shift
12.1.1(d)	No		
12.1.1(e)	Yes	TJC-209	APN: 14 - Data Register Capture and Shift
12.1.1(f)	No		
12.2.1(a.i)	No		

Rule	Verified	Message(s)	APN/1149.1 Semantic Check/ Structure Check
12.2.1(a.ii)	No		
12.2.1(b)	No		
12.3.1(a)	No		
12.3.1(b)	No		
12.4.1(a)	No		
Conformance and Documentation			
13.1.1(a)	No		
13.1.1(b)	No		
13.2.1(a)	No		
13.3.1(a)	No		
13.3.1(b)	Yes	Various - BSDL messages	Various

Verify 1149.1 Boundary Test Modes

IEEE 1149.1 BSV creates test modes when:

- If invoked on a "base" test mode (i.e. any test mode which does not define an instruction in the SCAN statement of the test mode definition file) and iomapping, iomappinginpin or iomappingoutpin checking is requested, then BSV builds the `TB_EXTEST_CAP_UPDT` test mode.

  Refer to "SCAN" in the *Encounter Test: Guide 2: Testmodes* for additional information.

- If an instruction accesses a large data register or fails verification, BSV will (by default) automatically create an additional test mode. Test modes created by BSV are used for checking and/or analysis purposes.

You can request test modes by accepting the default of `registeridentification=fail` or specifying `registeridentification=yes` (to build a test mode and perform structure verification for each register). Using the GUI, select either *If Failure* (the default) or *Always* for *Build Test Mode(s) for Register Identification.*

To suppress creation of test modes, other than the modes created for checking purposes, specify `registeridentification=no` or select *Never* for the GUI option.

For additional information, refer to the descriptions for these keywords in "verify_11491_boundary" in the *Encounter Test: Reference: Commands*.

Verify 1149.1 Boundary Results File

One of the files generated by Boundary Scan Verification produces is a readable results file. The results file contains the verifications patterns and detailed results of the boundary scan checks completed in a single run of BSV. The results file contains the following information:

- File header and legend for reading the file
- Verification Group names (AGNs) and verification procedure names (APNs) for checks
- Verification patterns containing stimulus, expected responses and simulation results for each check
- TAP state transitions for verification patterns
- Pass/Fail indicator and expect/miscompare count for each check
- Miscompare indicators for failing patterns and point of miscompare (e.g. a particular Primary Output or bit position in a test data register).
- Summary of verification results as seen in the BSV log

BSV creates one results file each time it is invoked. The file is named as:

IEEE11491ver. *testmode.experiment*

Each group of checks, and each check and each sequence include descriptive text in addition to a unique identifying number. BSV uses an absolute group number (AGN) to identify verification groups, an absolute procedure number (APN) to identify checks, and an absolute sequence number (ASN) to identify sequences. The AGN and APN numbers and descriptive text is also written to the BSV log and TBD to allow correlation of the data seen in the results file with information seen in the log file and TBD.

There are two aspects to understanding the information included in the results file. One is how to read the information which is shown and secondly, how to know what the checks are and how they are accomplished. This section includes a description of how to read the information shown in the results file. The checks are described in "1149.1 Boundary Scan Verification Checks" on page 87.

BSV uses a few different formats for reporting the verification patterns. All checks include a list of primary input and primary output pins participating in the check. Checks that involve the boundary register may also include a list of boundary register latches participating in the check either explicitly via a stim latch/measure latch or implicitly via serial load/unload of the boundary register.

Example: Failing BSV check which uses PI/PO stim/measure/expect data

```
**************************************************************************
GroupName: Instruction Register (AGN:3) TestModeName: IEEE11491
**************************************************************************
ProcedureName: Instruction Register Scan (APN:5)
SequenceName: Scan InstructionRegister. No expect on Capture IR. Expect 5 bit
Signature. (ASN:40)
TAPstates:
TLR->RTI->SelectDR->SelectIR->CaptureIR->ShiftIR(10)->Exit1IR->PauseIR
Stim Vector Data:
    16->01010101010101010101010101010101 TCK
    19->00111100000000000000000000011001 TMS
    17->NNNNNNNNN1100001111NNNNNNNNNNNNN TDI

Expect Vector Data:
    18->ZZZZZZZZZUUUUUUUUUU1100001111ZZZ TDO

Result Vector Data:
    18->ZZZZZZZZZ11000000011000011111ZZZ Miscompare=v21 TDO
SequenceName: Return to home state (ASN:41)
TAPstates: PauseIR->Exit2IR->UpdateIR->SelectDR->SelectIR->TLR(2)
Stim Vector Data:
    16->0101010101010 TCK
    19->1111111111111 TMS
    17->NNNNNNNNNNNNN TDI

Expect Vector Data:
    18->ZZZZZZZZZZZZZ TDO
Result Vector Data:
    18->ZZZZZZZZZZZZZ TDO
Instruction Register Scan (APN:5) NumExpects = 36,
NumMiscompares = 4 (failed)
```

This example has two sequences named ASN:40 and ASN:41. The sequences are included in a procedure (check) called APN:5 which is included in a group of checks called AGN:3. The sequences shows which PI and PO pins are participating in the check. Each pin in a sequence is shown with its hier model pin index, the values that were applied to or observed at the pin and the pin name. Each 'column' of stim, expect, or result data represents one vector or pattern. Reading from left to right shows the values for a pin over patterns. This example shows a miscompare (failing check). The first miscompare occurred on the TDO pin in vector 21 of the first sequence (ASN:40). The miscompare is a result of the TDO expect of a logic 1 in vector 21 and a TDO result of 0 in vector 21. Additional miscompares exist as indicated by a miscompare count of four. The other miscompares can be located by finding additional occurrences of '`miscompare=vnn`' and or by reviewing the expect versus result data for a particular failing pin. Each sequence includes a list of TAP states the sequence goes through. If a sequence remains in a state for more than one cycle, then a count is shown for that state. In this example, sequence ASN:40 remained in the `ShiftIR` state for 10 clock cycles as denoted by `ShiftIR`(10).

Example: Passing BSV boundary register check which uses PI/PO stim/measure/expect data

```
**************************************************************************
ProcedureName: Preload (APN:22)
SequenceName: Load Instruction Register with PRELOAD opcode (ASN:114)
TAPstates:TLR->RTI->SelectDR->SelectIR->CaptureIR->ShiftIR(4)->Exit1IR->
    UpdateIR->RTI
Stim Vector Data:
    16->0101010101010101010101010 TCK
    19->00111100000000001111001 TMS
    17->NNNNNNNNNN00001100NNNNN TDI
Expect Vector Data:
    18->ZZZZZZZZZZUUUUUUUUZZZZZ TDO
Result Vector Data:
    18->ZZZZZZZZZZ11000000ZZZZZ TDO
SequenceName: For Boundary register load input and internal cells with the safe
state, control cells with the enable value and output cells with the active state
(ASN:115)
TAPstates:
RTI->SelectDR->CaptureDR->ShiftDR(6)->Exit1DR->UpdateDR->RTI
Stim Vector Data:
    16->0101010101010101010101010 TCK
    19->11000000000000001111001 TMS
    17->NNNNNN110000000011NNNNN TDI
Expect Vector Data:
    18->ZZZZZZUUUUUUUUUUUUZZZZZ TDO
Result Vector Data:
    18->ZZZZZZ1111111111XXZZZZZ TDO
Boundary register stim, expect and result data as seen at TDI/TDO Data extracted
from: TDI VecNum = 7 -> TDI VecNum = 18, TDO VecNum = 7 -> TDO VecNum = 18)
```

stim val	exp val	act val	error	BSDL cell#	control cell	cell function	port name	I/O Cellname
1	U	1		0		control	*	
0	U	1		1		input	OE	BP3350_A
0	U	1		2		input	SCLK	BP3350_A

```
stim  exp    act           BSDL   control    cell        port       I/O
val   val    val    error  cell#  cell       function    name       Cellname

0     U      1             3                 input       RST        BP3350_A

0     U      1             4                 input       SDS_RD     BP3350_A

1     U      1             5      0          output3     AGC_HLD    BP3350_A
```

```
SequenceName: Load instruction register with EXTEST opcode.
Expect all SOPs to switch to active state on falling edge of TCK
pulse entering UpdateIR. (ASN:116)
TAPstates: RTI->SelectDR->SelectIR->CaptureIR->ShiftIR(4)->Exit1IR->UpdateIR->RTI
Stim Vector Data:
     16->010101010101010101010 TCK
     17->NNNNNNNN00000000NNNNN TDI
     19->111100000000001111001 TMS
Expect Vector Data:
     10->UUUUUUUUUUUUUUUUUU111 AGC_HLD
     18->ZZZZZZZZUUUUUUUUZZZZZ TDO Result
Vector Data:
     10->XXXXXXXXXXXXXXXXXX111 AGC_HLD
     18->ZZZZZZZZ11000000ZZZZZ TDO
SequenceName:Return to the home state (ASN:117)
TAPstates: RTI->SelectDR->SelectIR->TLR(2)
Stim Vector Data:
     16->010101010 TCK
     19->111111111 TMS
     17->NNNNNNNNN TDI
Expect Vector Data:
     18->ZZZZZZZZZ TDO
Result Vector Data:
     18->ZZZZZZZZZ TDO
Preload (APN:22) NumExpects = 51, NumMiscompares = 0 (passed)
SAMPLE/PRELOAD (AGN:10) NumExpects = 636, NumMiscompares = 0 (passed)
```

This example shows how BSV provides two views of the boundary register data for a serial load/unload of the register. The second sequence (ASN:115) is being used to serially load the boundary register as preconditioning for the next sequence. The load sequence is first shown in the traditional horizontal view. Beneath it, the boundary register load data is shown in a table format that maps the load data to the boundary register functionality. The table header shows what vectors the stim, expect and result data was taken from. The stim val column shows the values loaded into the boundary register via TDI (re: TDI Stim Vector Data). The exp val column shows the values that are expected to be unloaded from the boundary register via TDO (re: TDO Expect Vector Data). The act val column shows the values which were actually unloaded from the boundary register via TDO (re: TDO Result Vector Data). The error column will have 'miscompare' if a miscompare occurred at TDO for the particular bit position. The next four columns show the boundary register functionality as defined by the BSDL. This information does not change from check to check. The last column labeled *I/O CellName* contains the logic model DEF name of the hierarchical cell which contains the driver/receiver logic for the port. The next sequence (ASN:116) loads the `EXTEST` instruction into the instruction register and traverses to the UpdateIR state where it expects the system output pins (SOPs) to switch to their `PRELOAD` state as determined by the previous boundary register load. In this example, the `AGC_HLD` pin switched to logic one as expected.

Example: Failing BSV boundary register iomapping check that uses PI/PO stim/measure/expect data and stim latch/measure latch data.

```
********************************************************************************
GroupName: Boundary Register I/O Mapping System Input Pins (AGN:14)
TestModeName: TB_EXTEST_CAP_UPDT
********************************************************************************p
rocedureName: Boundary Register IO Mapping System Input Pin (APN:24)
TargetPin: OE, TargetPinValue: 1
SequenceName: Transfer from CaptureDR to ShiftDR and Precondition Boundary register
(ASN:8)
TAPstates: CaptureDR->ShiftDR
Stim Vector Data:

    16->010 TCK
    19->000 TMS
    17->NNN TDI

Stim Latch Data (vector=3):


 representative                  stim  BSDL   control  cell        port  I/O
 latch block                     val   cell#  cell#    function    name  CellNam

 TBB_BSC_MERGED_0.Shift.LATCH1   0     0               control     *     BP3350_A

 TBB_BSC_OE_1.Shift.LATCH1       0     1               input       OE    BP3350_A

 TBB_BSC_SCLK_2.Shift.LATCH1     1     2               clock             BP3350_A

 TBB_BSC_RST_3.Shift.LATCH1      1     3               input             BP3350_A

 TBB_BSC_SDS_RD_4.Shift.LATCH1   1     4               input             BP3350_A

 TBB_BSC_AGC_HLD_5.Shift.LATCH1  1     5      0        output3           BP3350_A


Expect Vector Data:

    18->UUU TDO

Result Vector Data:

    18->XXZ TDO

SequenceName: LoadSuffixSequence: Transfer from ShiftDR to CaptureDR
TAPstates: ShiftDR->Exit1DR->UpdateDR->CaptureDR

SequenceName: Precondition input pins, transfer from CaptureDR to ShiftDR and
observe boundary register. (ASN:58)
TAPstates: CaptureDR->ShiftDR
Stim Vector Data:
    11->11 OE
    12->00 RST
    14->00 SCLK
    15->00 SDS_RD
    16->010 TCK
    17->NNN TDI
    19->000 TMS
Expect Vector Data:
    18->UUU TDO
Result Vector Data:
    18->ZZZ TDO
```

Expect and Result Latch Data (vector=3):

representative latch block	exp val	act val	error	BSDL cell#	con- trol cell	cell function	port name	I/O Cellname
TBB_BSC_MERGED_0.Shift. LATCH1	U	0		0		control	*	BP3350_A
TBB_BSC_OE_1.Shift. LATCH1	1	1		1		input	OE	BP3350_A
TBB_BSC_SCLK_2.Shift. LATCH1	0	0		2		clock	SCLK	BP3350_A
TBB_BSC_RST_3.Shift. LATCH1	0	0		3		input	RST	BP3350_A
TBB_BSC_SDS_RD_4.Shift. LATCH1	0	0		4		input	SDS_R D	BP3350_A
TBB_BSC_AGC_HLD_5.Shift. LATCH1	U	X		5	0	output3	AGC_H LD	BP3350_A

```
SequenceName: LoadSuffixSequence: Transfer from ShiftDR to CaptureDR
TAPstates: ShiftDR->Exit1DR->UpdateDR->CaptureDR

Boundary Register IO Mapping System Input Pin (APN:24) TargetPin: OE,
TargetPinValue: 1 NumExpects = 4, NumMiscompares = 0 (passed)
```

This example shows the results of an iomapping input pin verification. The APN description (APN:24) shows the target pin name and target pin value. All iomapping checks occur in the `EXTEST` mode whose name is shown in the AGN description (`TB_EXTEST_CAP_UPDT`). The first sequence begins in `CaptureDR` since `CaptureDR` is the home state for the `EXTEST` mode. BSV uses stim latch and measure latch events to perform the iomapping checks as well as other checks in the `EXTEST` mode. The stim latch and measure latch events are shown in a table to show stim, expect, and result values in each of the boundary register bit positions. These tables also show the correspondence of the boundary register bits to the boundary register functionality as defined by the BSDL. This information does not change from check to check. When a sequence contains a measure latch, the table will include the expect and actual values (exp val and act val) for each bit position in the register and an indication of whether the latch which corresponds to that bit position got the expected value. Miscomparing bit positions will have 'Miscompare' in the error column.

BSV also provides a description of its mapping of the BSDL boundary register functionality to the Encounter Test logic model. This information is shown at the top of the results file in a table called 'Boundary Register - BSDL to Logic Model Mapping'. The example is split for the purpose of viewing within this document. In the actual output, the columns starting with *I/O Cell Name* continue to the right of *BSDL Port Name.*

```
Boundary Register - BSDL to Logic Model Mapping
------------------------------------------------------------------
```

```
BSDL   BSDL        BSDL     Boundary    Representative                  BSDL
Cell   Cell        Cell     Cell        Latch                           Port
Number Type        Function Name        BlockName                       Name

0      BSR_ENAB_NT control  BSR_ENAB_NT TBB_BSC_MERGED_0.Shift.LATCH1   *

1      BSR_IN_NT   input    BSR_IN_NT   TBB_BSC_OE_1.Shift.LATCH1       OE

2      BSR_CLKIN   clock    BSR_CLKIN   TBB_BSC_SCLK_2.Shift.LATCH1     SCLK

3      BSR_IN_NT   input    BSR_IN_NT   TBB_BSC_RST_3.Shift.LATCH1      RST

4      BSR_IN_NT   input    BSR_IN_NT   TBB_BSC_SDS_RD_4.Shift.LATCH1   SDS_RD

5      BSR_OUT_NT  output3  BSR_OUT_NT  TBB_BSC_AGC_HLD_5.Shift.LATCH1  AGC_HLD
```

```
I/O Cell Name   |TSI bit StimLatch TSI bit MeasLatch  hier block   flat node
*               |6                 1                  1141         691
BP3350_A        |5                 2                  1152         697
BP3350_A        |4                 3                  1168         705
BP3350_A        |3                 4                  1160         701
BP3350_A        |2                 5                  1176         709
BP3350_A        |1                 6                  1126         683
```

The information in the columns is derived as follows:

- *BSDL Cell Number* from 1149.1 BSV input BSDL
- *BSDL Cell Type* from 1149.1 BSV input BSDL
- *BSDL Cell Function* from 1149.1 BSV input BSDL

- *Boundary Cell Name* from the logic model
- *Representative Latch BlockName* from the logic model
- *BSDL Port Name* from 1149.1 BSV input BSDL
- *I/O Cell Name* from the logic model
- *TSI bit StimLatch* from the logic model as defined by Create a Test Mode
- *TSI bit MeasLatch* from the logic model as defined by Create a Test Mode
- *hier block* from the logic model
- *flat node* from the logic model

A complete view of all the latches in the boundary register, including update latches, can be viewed via Report Model Statistics (control / observe register information) for the `EXTEST` mode (e.g. `TB_EXTEST_CAP_UPDT`).

Comparing Unit and Zero Delay Simulations for Boundary Scan Verification

When using `verify_boundary_scan` to generate patterns and simulate them to verify compliance, there might be situations where the type of simulator being used affects the final simulation results. The default for verify 1149 boundary scan is to use a unit delay simulation. Some designs might require zero delay simulation because of unbalanced clock lines. The `verify_1149_boundary` command has the option `delaymode=zero` to invoke a zero delay simulation.

For unit-delay simulation to work, you have to account for the flip-flops, latches, and RAMs in the model, taking clock distribution unit-delay skews into consideration. If the number of gate delays are not well balanced (a difference of 7 or more gates), you might have to modify the model to make unit delay simulations work. For example, consider a case in which a single clock feeds two flops A and B, where flop A feeds flop B. If flop B has 10 extra buffers on the clock line, flop A will evaluated and its value will be propagated to flop B. In this case, flop B would capture the input data to flop A. Under normal situation, flop B would have expected to capture the original value in flop A instead of its input value. This is an example of how an unbalanced clock path can cause incorrect simulation results. This situation can be resolved by inserting more gates on the clock line to flop A to make the number of delays more equivalent. In addition to balancing the clock distribution unit delays, you can also insert gates (buffers) into the data inputs or outputs of all latch flip-flops or RAM models. This delays the propagation of new data signals feeding to opposite phase latches until their clocks can turn off.

Zero delay simulation predicts the clock always wins the clock/data race. That is, for clock gating or data hold time races, zero-delay simulation makes calculations assuming that the clock goes off before new data arrives.

Note: When running zero delay simulation in NC-sim, you might need to add the keyword `ncelab -seq_udp_delay 10ps` to add some unit of delay for latch and flip-flop calculations.

3

Verify Core Isolation

Verify Core Isolation is the Encounter Test application that analyzes the design and verifies that valid isolation exists for each embedded device. The specification of a device's isolation requirements is provided in a file called the Macro Isolation Control (MIC) file. See “Isolation Requirements” on page 213 for more information. If valid isolation does not exist, Verify Core Isolation identifies the device and operation in error. Verify Core Isolation saves all valid isolation details for use by subsequent applications.

Encounter Test supports the testing of embedded devices known as *macros* or *cores* by deriving a correspondence between each device pin and a product input or output (I/O) or scannable latch. Device testing can then be accomplished by applying test values to the product I/O and latches and observing macro response values at the product I/O and latches.

This concept of controlling embedded device pins through product I/O or scannable latches is known as *isolation*. It has the following elements:

- Correspondence

  Correspondence is the one-to-one relationship between an embedded device pin and one of the following correspondence points:

  - A tester-accessible product I/O (primary input, primary output)
  - A scannable latch

  The logic path between a correspondence point and a device pin is called a correspondence path.

- Preconditioning

  Preconditioning is the set of values placed on product I/O or in scannable latches that enable the correspondence paths.

Verify Core Isolation supports the concept of “isolation by operation,” which means that multiple isolation sets may exist, one for each testing-related operation (protocol) of the device.

Performing Verify Core Isolation

To perform *Verify Core Isolation* using the graphical interface, refer to “Verify Core Isolation” in the *Encounter Test: Reference: GUI*.

To perform *Verify Core Isolation* using command lines, refer to “verify_core_isolation” in the *Encounter Test: Reference: Commands.*

A license for either Encounter Test Architect, Encounter True-Time, or Daignostics Engine is required to run Verify Core Isolation. Refer to “Encounter Test and Diagnostics Product License Configuration” in *What’s New for Encounter® Test and Diagnostics* for details on the licensing structure.

Restrictions

Please note the following restrictions as you run Verify Core Isolation:

- Verify Core Isolation supports only operations requiring normal loads. Verify Core Isolation does not support operations requiring skewed loads.
- Verify Core Isolation does not derive correspondence through scan elements.
- Preconditioning is derived as a single state only, i.e., no sequential preconditioning.
- When core scan chains are “visible” to Encounter Test (a core “white” or “gray” box model), then the core scan chains may be merged with chip chains. Conversely, if core chains are not visible (a core “black” box model), then core scan chains must isolate to chip I/O, either directly or through combinational logic only.
- For devices to be eligible for grouping, they must all use the same MIC file. Refer to “Grouping” on page 214 for more information.

Prerequisite Tasks

1. Open or import a new project. Refer to “Performing Build Model” in the *Encounter Test: Guide 1: Models* for details.
2. Create or select a test mode, using `SCAN TYPE=LSSD` or `GSD`. Refer to “Performing Build Test Mode” in the *Encounter Test: Guide 2: Testmodes* for more information on creating or selecting a test mode.

Input Files

The suffix*.gz*, indicating file compression, is attached to the files listed below if they were created with the storage versus performance environment variable set to `TB_MODE=COMPRESS`*.* Refer to "Using Commands to Manage Files and Disk Space" in the *Encounter Test: Reference: Commands* for additional information.

Verify Core Isolation uses files that result from importing the design and from defining the test mode. It also uses a Macro Isolation Control file from the technology library and, in some cases, a user control file or a linehold file that you may define.

Logic Model Files

The following files resulting from *Import New Circuit* are used as input to Verify Core Isolation:

- `hierModel`

  This file contains the basic logical block, pin and net connections. It supports an arbitrary number of levels, as required, to describe the logic hierarchy. This model contains all levels of logic down to and including the primitive gates.

- `hierAttributes`

  The hierarchical model attribute file contains all the block, pin, net, and macro attributes. The macro attributes are used in conjunction with the MIC file to provide a complete description of the isolation requirements.

- `flatModel`

  This file contains a flattened, primitive gate-level logic model which is the logical equivalent of the `hierModel`.

- `globalData`

  This file keeps track of all the Encounter Test-generated data and files. It stores audit information and registers pattern statistics.

Test Mode Files

The following files resulting from *Create Test Mode(s)* are used as input to Verify Core Isolation:

- `modeInfo.`*`modename`*

  This file contains model tables specific to the test mode.

- `activeNodeMap.`*`modename`*

  This file identifies which pins in the hierarchical model are active in the test mode.

- `TSItsvInterface.`*`testmodename`*

  This file contains the identification of test structures for use by *Verify Core Isolation.*

Macro Isolation Control (MIC) file

The macro isolation control (MIC) file defines the isolation requirements for a device type. It is usually provided by the IP provider. The MIC file resides in a directory specified in the `MICPATH`. The `MACRO_MIC_NAME` property in the design source identifies the MIC used for the macro. If there is no `MACRO_MIC_NAME`, and `MACRO=YES`, Verify Core Isolation verifies isolation for all pins simultaneously.

For syntax and an example of the MIC file, refer to "Macro Isolation Control (MIC) Reference" *on page 223.* The MIC file path is an optional input to Verify Core Isolation.

User Control File

The optional user control file is a text file that assists Verify Core Isolation in verifying or modifying isolation. It contains a set of statements used to override Verify Core Isolation automatic processing. It is parsed and semantically checked by Verify Core Isolation. Refer to "Verify Core Isolation User Control Files" on page 217 for descriptions of the statements that are allowed in a user control file.

Using Verify Core Isolation

To invoke Verify Core Isolation using the Graphical User Interface, do the following:

1. Click *Verification - Verify - Core Isolation* on the main menu. The Verify Core Isolation window is displayed.
2. Select the desired options on the Verify Core Isolation window. The default is to run Verify Core Isolation to verify isolation for all devices, all test algorithms (for example, manufacturing test, diagnostics), and all test operations, as specified in the MIC file. Refer to "Verify Core Isolation" in the *Encounter Test: Reference: GUI* for detailed information on the Verify Core Isolation windows.
3. Select *Run* to begin Verify Core Isolation processing.

   When Verify Core Isolation processing is complete, Encounter Test issues a beep.

Analyzing Verify Core Isolation Messages

To analyze Verify Core Isolation messages using the Graphical User Interface, do the following:

1. Select the *Messages* Tab and click the *View Messages* icon; or *Window - Messages - Verify Core Isolation.*
2. Select a message to analyze from the *Verify Core Isolation Message Summary* list.
3. Select *View.*

   Note: To ensure that Encounter Test produces valid test data, analyze and resolve all severe errors before proceeding to other applications. Severe errors in Verify Core Isolation may cause the Create Core Tests application to fail to produce test data.
4. Select a message to evaluate from the *Verify Core Isolation Specific Message List.*
5. Select *Analyze.*

   Encounter Test displays the portion of the design causing the message. Additional information about blocks, pins, and nets is displayed in the Information Window. The design state is set to the appropriate analysis state. Help messages state how to analyze problems.

   To obtain help about a specific message, select the message and then select the *Message Help* button.

Refer to "Debugging Verify Core Isolation (TMI) Messages" on page 201 and "Message Analysis Windows" in the *Encounter Test: Reference: GUI* for additional information.

Output Files

The following files are produced:

- Binary Isolation File - `MacroIsolationbin.`*`testmode`*

  This file contains verified isolation data in binary format for all devices for a test mode. The file is registered on the `globalData` file. The binary isolation file resides in the directory that contains the imported design.

  This file is built the first time a design is run. All the MIC file isolation data used by this design is parsed, checked, and stored in this file.
- ASCII Isolation File - `MacroIsolation.`*`testmode`*`[.`*`qualifier`*`]`

This file contains the verified isolation data in ASCII format. Verify Core Isolation creates this file if you choose to *Resolved Macro Isolation in a Separate File* option in the *Verify Core Isolation Reporting* field. The ASCII isolation file can be used to send device isolation information to IC manufacturers. The optional name *qualifier* can be used to create several ASCII isolation files for a test mode.

- Message File - `TMImessageFile.`*`testmode`*

  This file contains message data (needed for Verify Core Isolation interactive analysis). The file is registered on the `globalData` file. The message file resides in the directory that contains the imported design.

- `TMIanalysis.testmode`

  This file supports interactive analysis.

The Encounter Test Log contains statistics on all embedded devices in all test modes, regardless of whether they are selected for processing. The statistics are generated when specifying `reportstatics=yes` in command line mode, or by selecting *Macro Statistics* on the Verify Core Isolation Reporting Options window.

An example of a macro statistics report:

```
Macro: macro usage block name 1
  MSV: S P I V    MTG: S P T    Mode: testmode name 1
.
.
.
  MSV: S P I V    MTG: S P T    Mode: testmode name n

Macro: macro usage block name 2
  Macro has not been selected for processing in any test mode.
```

Also included are these status indicators:

Indicator	Status
S (!S)	The macro selection criteria have been satisfied (not satisfied) for this test mode.
P (!P)	The macro has been processed (not processed) by Verify Core Isolation or Create Core Tests.
I (!I)	The macro was successfully (not successfully) isolated by Verify Core Isolation.
V (!V)	The macro isolation was verified (not verified) by Verify Core Isolation.

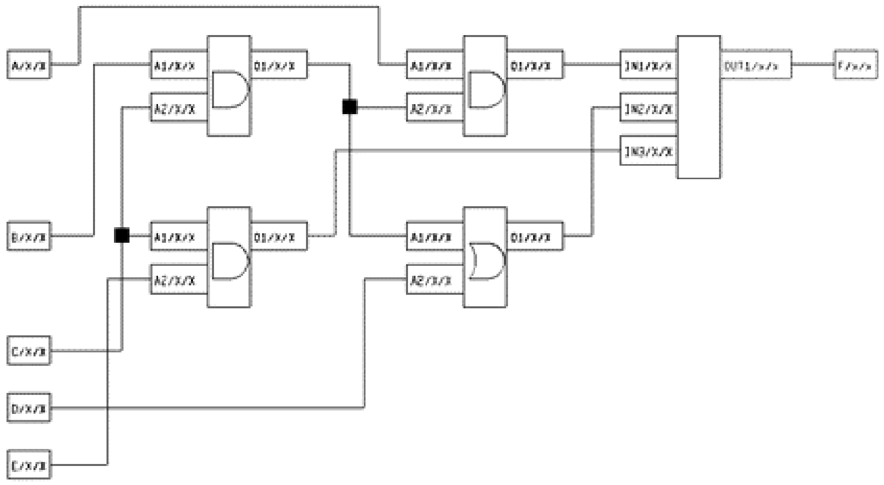
Indicator	Status
T (!T)	The macro passed Create Core Tests processing successfully (not successfully).

Debugging Verify Core Isolation (TMI) Messages

This section describes analysis techniques for Verify Core Isolation to identify causes of failures to isolate.

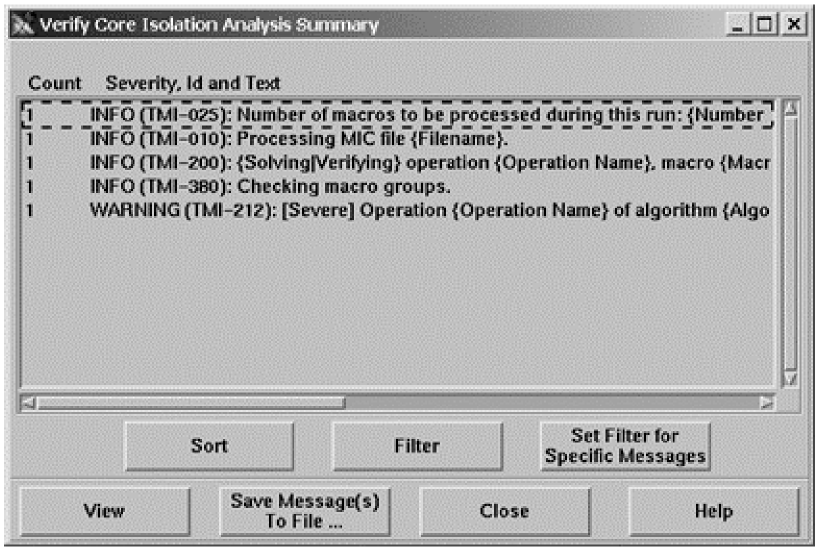
The design in Figure 3-1 is used for a sample analysis exercise.The rightmost block is a core that requires correspondence for all three input pins. The five pins at the left are the package pins that control the core input pin.

Figure 3-1 Sample Isolation Analysis Design



When Verify Core Isolation is unable to identify a solution for the operation identified in the message, information related to that failure is saved for analysis. Messages TMI-212 and TMI-213 are issued to indicate an unsolvable operation. Begin message analysis by clicking *Window - Messages - Verify Core Isolation* on the Encounter Test main menu. The message summary is displayed as shown in Figure 3-2.

Figure 3-2 Verify Core Isolation Message Summary

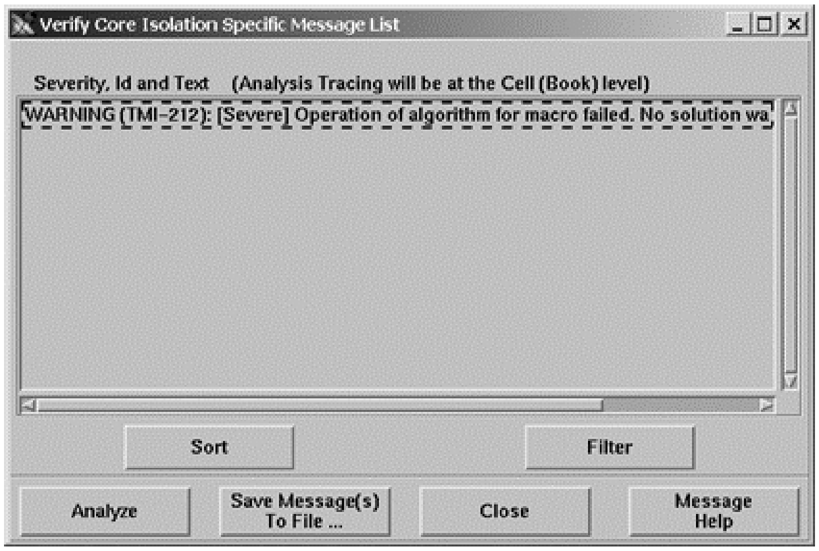

Verify Core Isolation Analysis Summary
Count
Severity, Id and Text
Sort
Filter
Set Filter for Specific Messages
View
Save Message(s) To File ...
Close
Help

```
1       INFO (TMI-025): Number of macros to be processed during this run: {Number
1       INFO (TMI-010): Processing MIC file {Filename}.
1       INFO (TMI-200): {Solving|Verifying} operation {Operation Name}, macro {Macr
1       INFO (TMI-380): Checking macro groups.
1       WARNING (TMI-212): [Severe] Operation {Operation Name} of algorithm {Algo
```

Select message TMI-212 as shown in the preceding figure then select *View* to display the Specific Message List in Figure 3-3 on page 202.

Note: An alternative method of displaying the list is to double-click the TMI-212 message entry.

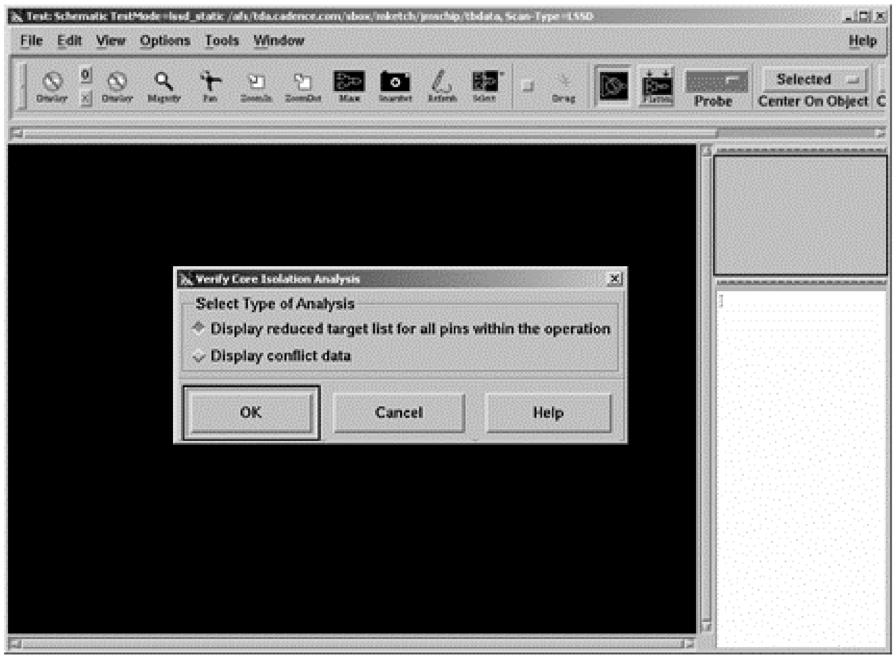
Figure 3-3 Verify Core Isolation Specific Message List


Verify Core Isolation Specific Message List
Severity, Id and Text (Analysis Tracing will be at the Cell (Book) level)
Sort
Filter
Analyze
Save Message(s) To File ...
Close
Message Help

```
WARNING (TMI-212): [Severe] Operation of algorithm for macro failed. No solution wa
```

Select message TMI-212 (or TMI-213 if it exists) and then select *Analyze* (or double-click the TMI-212 entry) to display the Schematic view and *Verify Core Isolation Analysis* parameters as shown in Figure 3-4 on page 203. Select the type of analysis information to identify the cause(s) of the failure to isolate. In some cases, analysis of a TMI-212 message another pop-up may be displayed referencing a message that preceded TMI-212 that identifies a specific condition that precludes the operation from resolving.

Figure 3-4 Schematic Display with Analysis Options



Verify Core Isolation Analysis

Select Type of Analysis

[✓ Display reduced target list for all pins within the operation]

Display conflict data

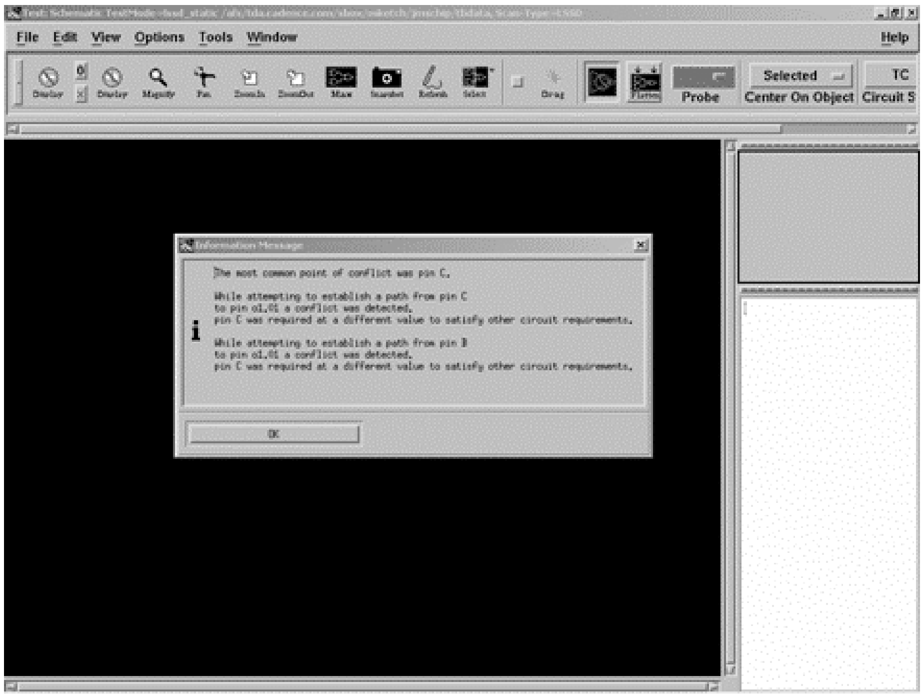
Select one of the following under *Select Type of Analysis*:

- *Display reduced target list for all pins within the operation* displays the *Verify Core Isolation Reduced Target List* showing the core pins requiring correspondence. Refer to "Verify Core Isolation Reduced Target List" on page 205.
- *Display conflict data* identifies the most common source of conflict determined by Verify Core Isolation as it attempted to solve the operation. A number of specific conflicts and the associated conditions that were attempted to be satisfied may also be identified.

  In some cases, no conflict data will be saved due to the inability to identify a source of conflict.

It is recommended to utilize both analysis functions to identify the root cases(s) of the failure. Figure 3-5 on page 204 shows results for conflict data.

Figure 3-5 Conflict Data Results

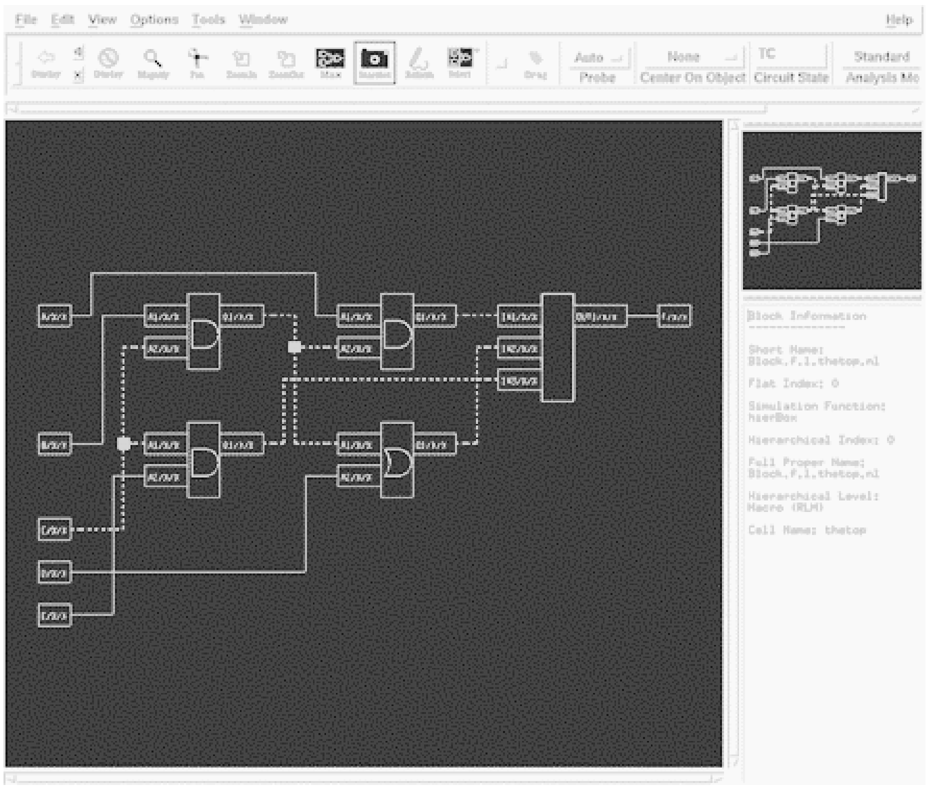


```
The most common point of conflict was pin C.

While attempting to establish a path from pin C
to pin o1.01 a conflict was detected.
pin C was required at a different value to satisfy other circuit requirements.

While attempting to establish a path from pin B
to pin o1.01 a conflict was detected.
pin C was required at a different value to satisfy other circuit requirements.
```

Note that the first line contains the most common point of conflict, pin `C`. from the example in Figure 3-1 on page 201. Verify Core Isolation identifies the most common point of conflict by examining the saved conflict information and determining which point in the logic was identified as the source of conflict in the most cases. Following the most common point of conflict are individual conflicts. For the first, pin `C` needs a value to satisfy some of the design requirements while it was also being used as part of a path to a core pin. Since it is not possible to be both a correspondence point and have a stable value, a conflict occurred. For the second conflict, pin `C` was again involved but in a slightly different way. In this case the conflict occurred when trying to establish a path from pin `B` to pin `o1.01`. It appears that pin `C` was required to be at one value to satisfy an earlier design requirement and now is needed to be at a different value. Pin `C` cannot simultaneously be at two different values.

At this point in the analysis, it appears that pin `C` is a point of interest. To gain an understanding of the role of pin `C` in the failure the schematic in Figure 3-6 on page 205 was produced by tracing back from the core. The effects of pin `C` were highlighted using the probe forward facility. Refer to Probe/Unprobe Nets in the *Encounter Test: Reference: GUI* for additional information.

Figure 3-6 Trace and Probe Results



Block Information

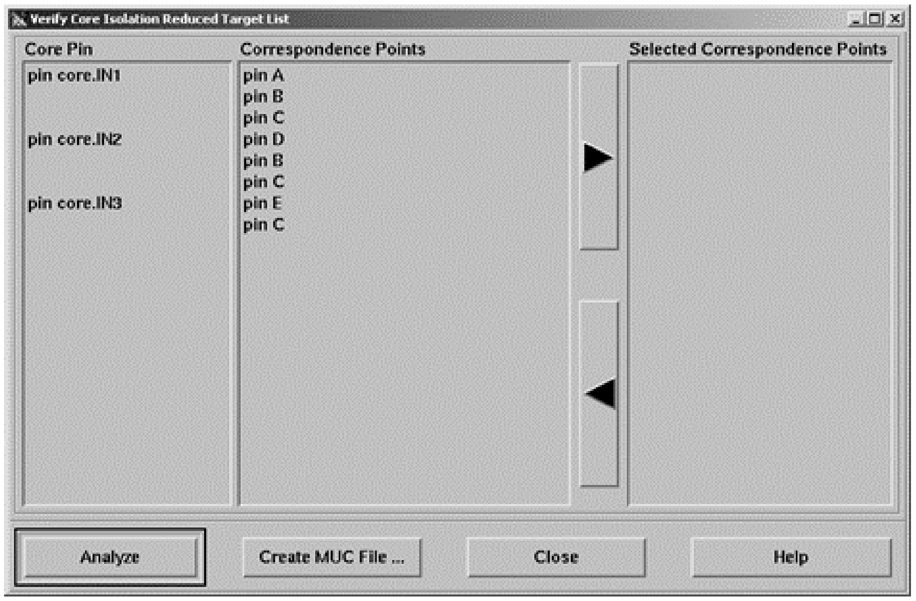
Short Name: Block.f.l.thetop.nl
Flat Index: 0
Simulation Function: hierBox
Hierarchical Index: 0
Full Proper Name: Block.f.l.thetop.nl
Hierarchical Level: Macro (RLM)
Cell Name: thetop

The schematic indicates that pin C may affect the paths to all three of the core pins of the example. Since all three of the core pins require correspondence, this may be a problem. Use the other analysis tool to better understanding the relationship of pin C to failure to successfully isolate the correspondence pins. By reselecting the specific TMI-212 message the Verify Core Isolation Analysis is re-displayed. Select *Display reduced target list for all pins within the operation*. Refer to "Verify Core Isolation Reduced Target List" on page 205 for details.

Verify Core Isolation Reduced Target List

Figure 3-6 shows a reduced target list.

Figure 3-7 Verify Core Isolation Reduced Target List


Verify Core Isolation Reduced Target List
Core Pin
Correspondence Points
Selected Correspondence Points
Analyze
Create MUC File ...
Close
Help

Core Pin	Correspondence Points
pin core.IN1	pin A
	pin B
	pin C
pin core.IN2	pin D
	pin B
	pin C
pin core.IN3	pin E
	pin C

The list columns contain the following elements:

- *Core Pin*s requiring correspondence (Macro Pin)
- *Possible Correspondence Points* (generally primary inputs or latches)
- *Selected Correspondence Points* for analysis

The *Analyze* button invokes interactive analysis.

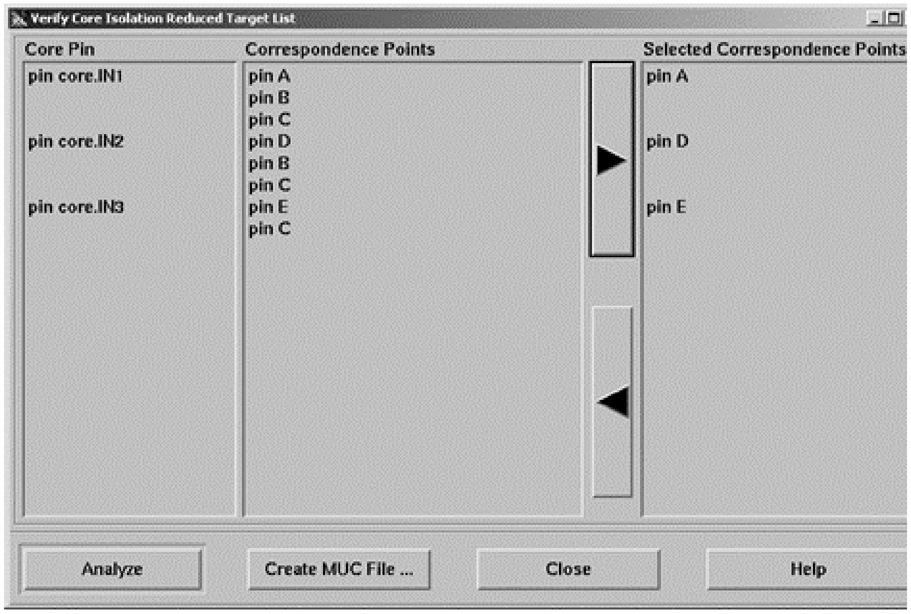
Create MUC File displays a file selection dialog to create a User Control file based on the selected pins in the *Selected Correspondence Points* column. Refer to "User Control File" in the *Verification and Analysis Reference* for details on file content. Refer to "File and Directory Selection from View Schematic and Message Windows" in the *Encounter Test: Reference: GUI* for details on saving a file.

The example in Figure 3-7 on page 206 displays the following information:

- `pin core,IN1` of the core has Pin `A`, pin `B`, and pin `C` as possible correspondence points
- `pin core.IN2` of the core has pin `D`, pin `B`, and pin `C` as possible correspondence points
- `pin.core.IN3` has pin `E` and pin `C` as possible correspondence points

Using knowledge of the design, the *Correspondence Points* (pin `A`, pin `D`, and pin `E` respectively) that were intended to feed each of the Macro Pins are selected and then moved the *Selected Correspondence Points* column as shown in Figure 3-8.

Figure 3-8 Reduced Target List with Selected Correspondence Points

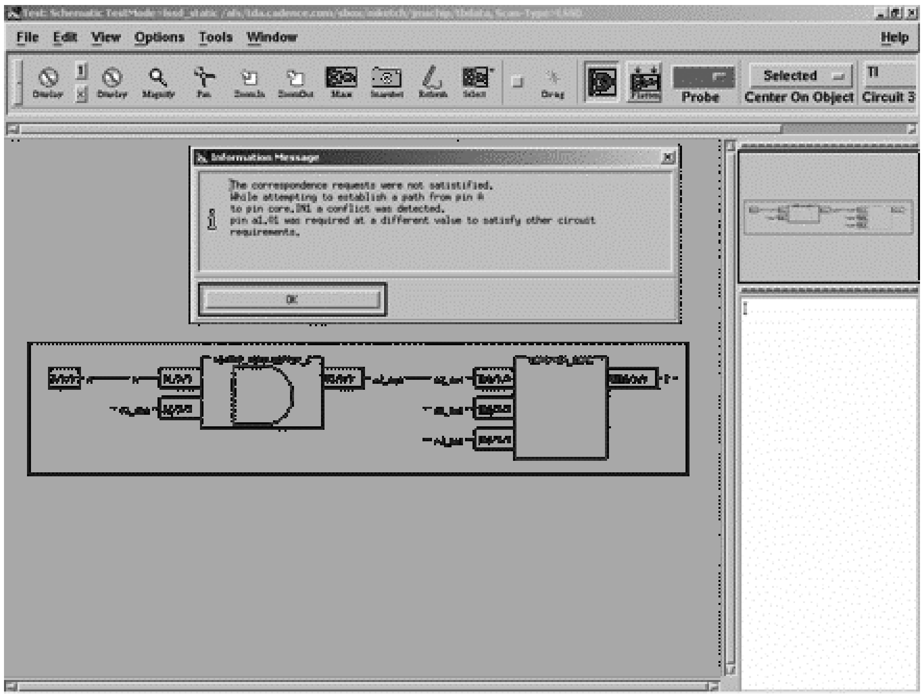


Verify Core Isolation Reduced Target List

Core Pin	Correspondence Points	Selected Correspondence Points
pin core.IN1	pin A pin B pin C	pin A
pin core.IN2	pin D pin B pin C	pin D
pin core.IN3	pin E pin C	pin E

Click *Analyze* to invoke interactive analysis and attempt to solve the selected set. The results (success or failure) of the attempt are displayed in Figure 3-9 on page 208.

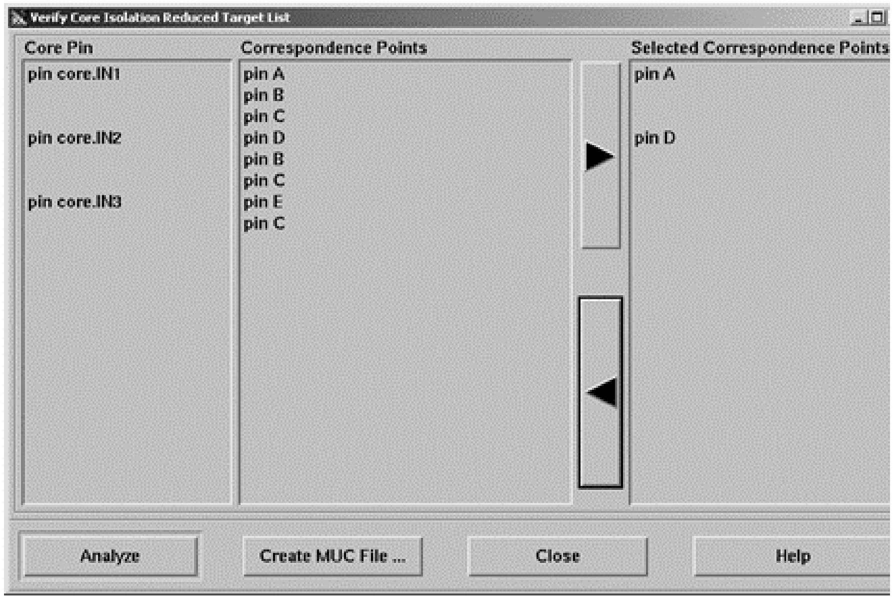
Figure 3-9 Initial Analysis Results



```
The correspondence requests were not satistified.
While attempting to establish a path from pin A
to pin core.IN1 a conflict was detected.
pin a1.01 was required at a different value to satisfy other circuit
requirements.
```

The *Information Message* states the analysis tool was unable to satisfy paths for the requested correspondence points. Reduce the failure by removing the correspondence point for pin `E`. Select pin `E` and click the left arrow to remove pin `E` from the *Selected Correspondence Points*. Figure 3-10 on page 209 shows the result.

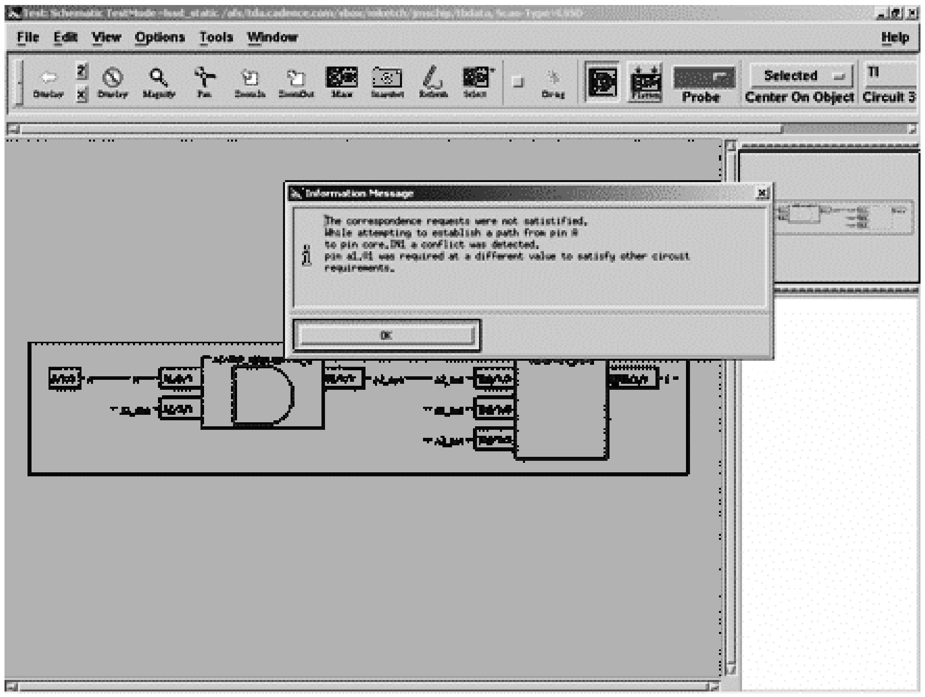
Figure 3-10 Reduced Correspondence Points



Core Pin	Correspondence Points	Selected Correspondence Points
pin core.IN1	pin A	pin A
	pin B	
	pin C	
pin core.IN2	pin D	pin D
	pin B	
	pin C	
pin core.IN3	pin E	
	pin C	

Click *Analyze* to invoke interactive analysis and attempt to solve the selected set. The results (success or failure) of the attempt are displayed in Figure 3-11 on page 210.

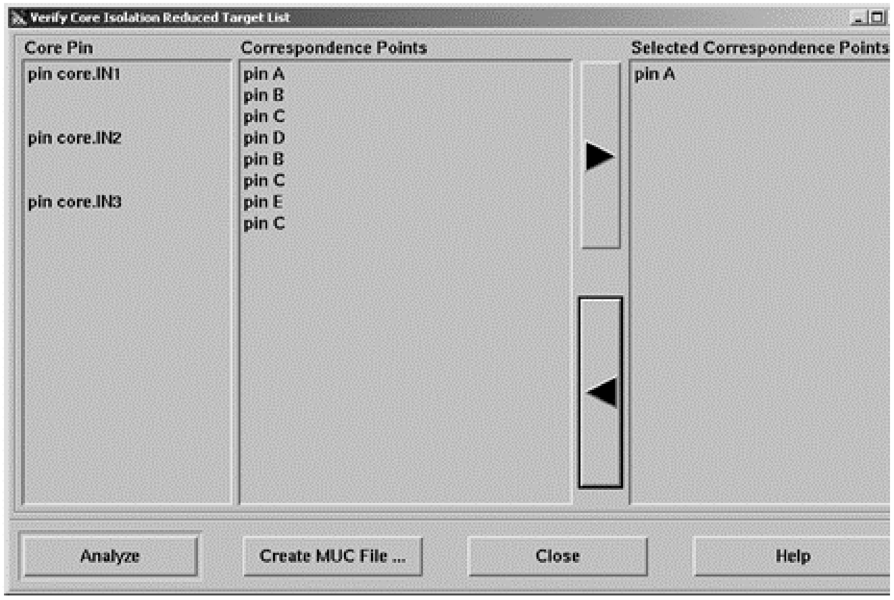
Figure 3-11 Analysis Results with Two Correspondence Points



```
The correspondence requests were not satisfied.
While attempting to establish a path from pin A
to pin core.IN1 a conflict was detected.
pin a1.01 was required at a different value to satisfy other circuit
requirements.
```

The *Information Message* states a problem still exists. Remove pin `B` from the Selected Correspondence Points to further reduce the number of correspondence paths (correspondence point / Macro pin pairs) to resolve. The result is shown in Figure 3-12 on page 211.

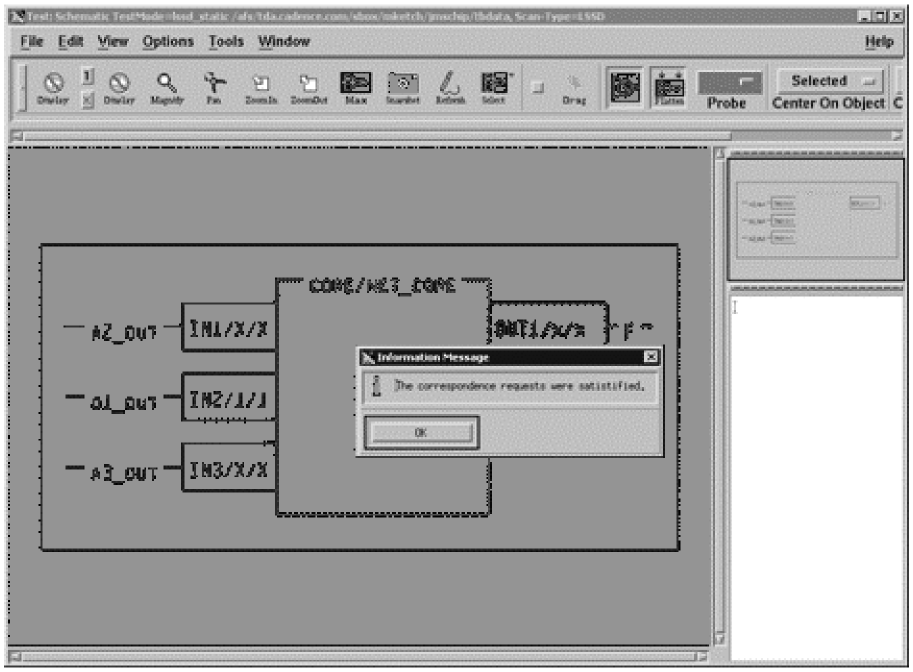
Figure 3-12 One Selected Correspondence Point


Verify Core Isolation Reduced Target List

Core Pin	Correspondence Points	Selected Correspondence Points
pin core.IN1	pin A	pin A
	pin B	
	pin C	
pin core.IN2	pin D	
	pin B	
	pin C	
pin core.IN3	pin E	
	pin C	

Click *Analyze* to invoke interactive analysis and attempt to solve the selected set. The latest results (success or failure) of the attempt are displayed in Figure 3-13 on page 211.

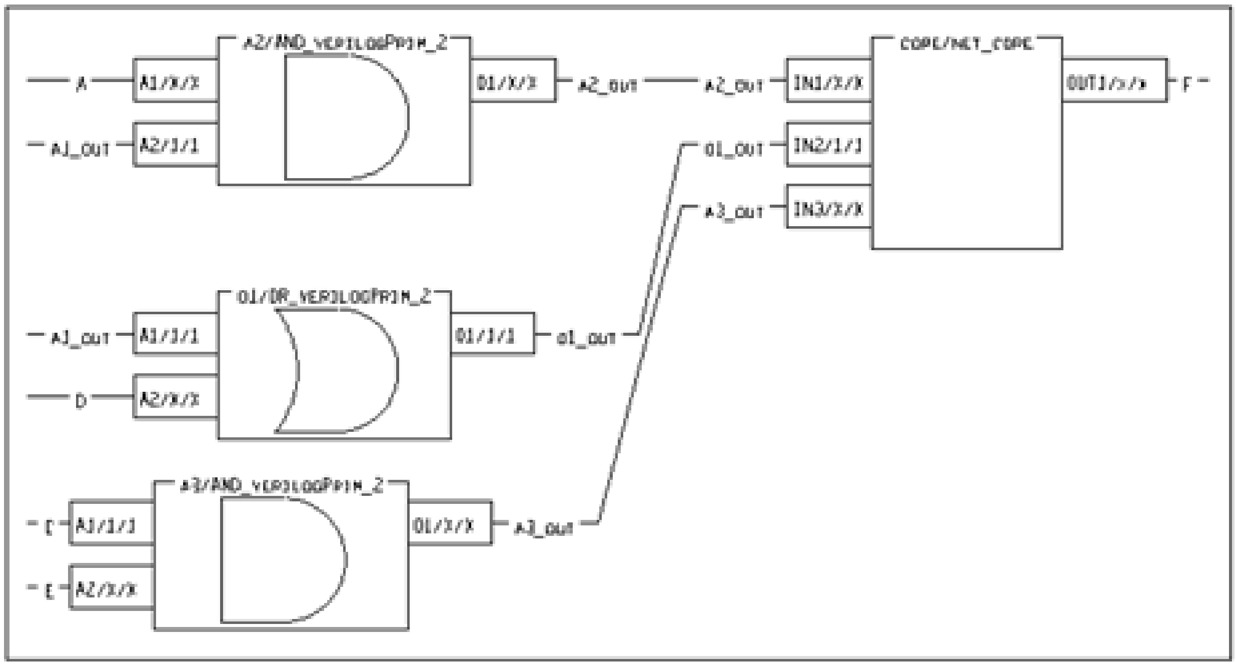
Figure 3-13 Satisfied Correspondence Requests


Information Message

The correspondence requests were satisfied.

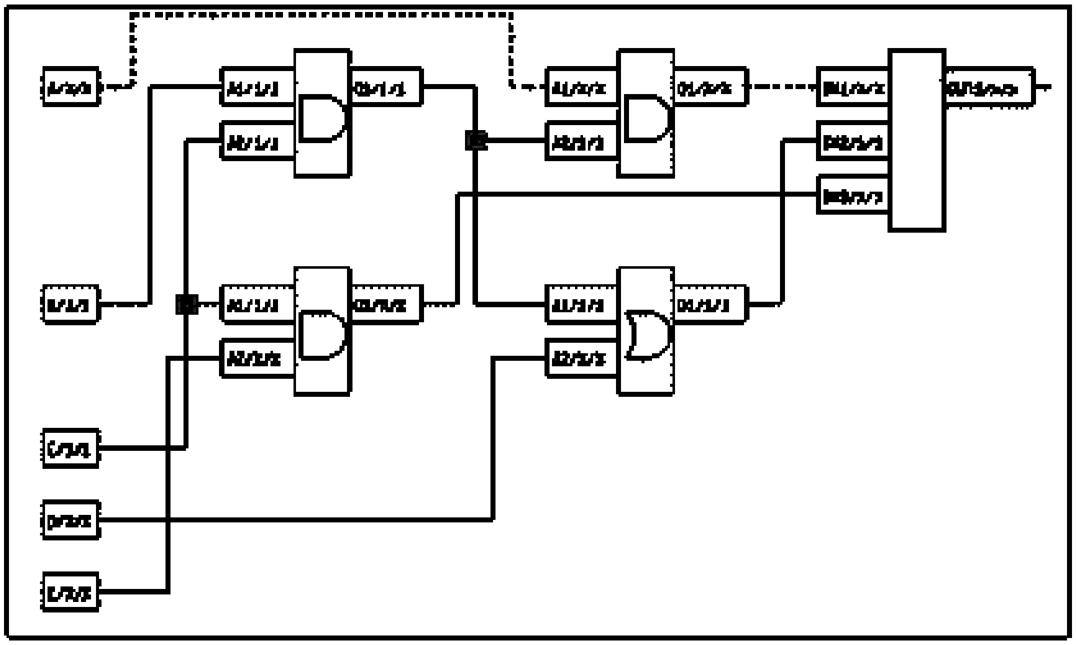
The *Information Message* states the correspondence requests were satisfied. Note that the schematic view has displayed the core in question with the design values used to satisfy the request. Tracing back from the core inputs produces the schematic display in Figure 3-14 on page 212.

Figure 3-14 Result from Core Input Trace



A forward probe from the successful correspondence point, pin `A`, produces the schematic display in Figure 3-15 on page 213.

Figure 3-15 Probe Results from Pin A



Examining the display shows that in order to establish a path from pin `A` to the `IN1` pin of the core the `AND` block required a value of `1` on its lower pin. By tracing back we see that to get that pin to a `1` required both pin `B` and pin `C` to be at a `1` but setting those two pins to value forced the `IN2` pin of the core to `1`. Given that correspondence is required on all the pins of the core, a logic change must be made. In this case changing the `OR` to an `AND` allows all the correspondence requirements to be met.

Isolation Requirements

Verify Core Isolation supports the concept of "isolation by operation," which simply means that multiple isolation sets may exist, one for each testing-related operation (protocol) of the device. Embedded device isolation requirements are identified in the Macro Isolation Control (MIC) file. The MIC file is usually provided by the device provider, although it may be provided by the user for custom devices. The MIC file relates the following elements:

- Device Test Algorithms

  The MIC file identifies one or more test schemes (algorithms) that are required to be applied to the device. For example, specific algorithms may be required only for diagnostic testing or burn-in testing. Since each algorithm may have specific operational requirements, operations are associated with each algorithm.

- Device Operations

  The MIC file also specifies the set of valid operations for the device and identifies the device pin isolation requirements for each operation.

- Device Pin Isolation Requirements

  Finally, the MIC file specifies how each device pin must be isolated. For example, it may indicate that a particular pin must correspond to a primary I/O and not a scan cell; or it may indicate a particular pin needs a fixed value. A pin may have different isolation requirements, depending on how it is used by each operation.

Refer to "Macro Isolation Control (MIC) Reference" *on page 223* for additional information on the MIC file and MIC statement syntax.

Grouping

Verify Core Isolation employs another concept, known as device grouping, in order to reduce embedded device test application time. Basically, grouping applies tests for multiple devices concurrently. However, grouping is only possible when the isolation of each device in the group is "compatible. This requires that:

- All devices in the group use the same MIC file.
- Input values can be simultaneously applied to all device input pin correspondence points for all devices in the group without conflict. Note that this does not necessarily mean that each device input pin must have a unique correspondence point. Broadcasting (fanout) of identical input values from a single shared correspondence point is allowed as long as the same value is always required at the device pins which correspond to that point.
- Output values can be simultaneously, but uniquely, measured at all device output pin correspondence points for all devices in the group. Note that this requires unique output correspondence points since it must be possible to attribute a failing value to a specific device.

Verify Core Isolation automatically generates grouped isolation when `GROUP=YES` is specified (or defaulted) on an `ALGORITHM` statement of the device's MIC file. To prevent grouping, `GROUP=NO` must be specified.

4

Verify Sequences

Use Verify Sequences to verify on product clock sequences. Verify Sequences reads selected sequence definitions (you must have already imported them to the TBDseq file) and simulates them, checking the simulation results against the pseudo PI events found in the sequence definition. If any miscompares are found, you may want to use *Test Pattern Analysis.* Refer to “Test Pattern Analysis” in the *Encounter Test: Guide 6: Test Vectors* for details.

To perform *Verify On Product Clock Sequences* using the graphical interface, refer to “Verify Sequences” in the *Encounter Test: Reference: GUI*.

To perform *Verify On Product Clock Sequences* using command lines, refer to “verify_sequences” in the *Encounter Test: Reference: Commands*.

Also refer to On-Product Clock Generation (OPCG) Test Modes in *Encounter Test: Guide 2: Testmodes*.

Prerequisite Tasks

1. Import a project design into the Encounter Test model format. Refer to “Performing Build Model” in the *Encounter Test: Guide 1: Models* for more information.
2. Create a Test Mode.

   Refer to “Performing Build Test Mode” in the *Encounter Test: Guide 2: Testmodes*.
3. Write the sequence definition in TBDpatt format. Refer to “TBDpatt Language Definition” in the *Encounter Test: Reference: Test Pattern Formats* for details.
4. Import the sequence definition data to translate the sequence definition data into the TBDseq. *testmode* file. If you were able to write the sequence definition (TBDpatt format) before creating the test mode, you can put it in the sequence definition file input to *Create a Test Mode* and skip this step. Refer to “Performing Import Sequence Definition Data” in *Encounter Test: Guide 6: Test Vectors* and “TBDpatt and TBDseqPatt Format” in the *Encounter Test: Reference: Test Pattern Formats*.

Input Files

The suffix `.gz`*,* indicating file compression, is attached to the files listed below if they were created with the storage versus performance environment variable set to `TB_MODE=COMPRESS`*.* Refer to "Using Commands to Manage Files and Disk Space" in the *Encounter Test: Reference: Commands* for additional information.

- `TBDseq`.*testmode* file

  This file contains initialization sequences. Refer to the description of TBDseq testmode file in the "Coding Test Sequences" section of the *Test Generation and Simulation Reference*.

Output Files

The suffix `.gz`*,* indicating file compression, is attached to the files listed below if the storage versus performance environment variable is set to `TB_MODE=COMPRESS`*.* Refer to "Using Commands to Manage Files and Disk Space" in the *Encounter Test: Reference: Commands* for additional information.

- `TBDseq.`*`testmode`*

  Verify On Product Clock Sequences sets audit fields in the sequence definition telling the results of the verification processing.

- `TBDbin.`*`testmode.experiment`*

This file contains simulation results, a by-product of the verification process. This file is needed if you want to analyze test data to diagnose problems in case miscompares are found. Refer to "Performing Test Pattern Analysis" in *Encounter Test: Guide 6: Test Vectors* and "Viewing Vectors" in the *Encounter Test: Reference: GUI* for additional information.

- `scopeData.`*`testmode.experiment`*

  This file contains the simulation results for watch list nets. This file is also used to analyze test data.

A

Verify Core Isolation User Control Files

User control files are used to provide specific directions to Verify Core Isolation. They can be comprised of the following statements:

- “ALGORITHM Statement” on page 217
- “CORRESPONDENCE Statement” on page 218
- “GROUP Statement” on page 219
- “HOLD Statement” on page 220
- “MACRO Statement” on page 220

Each statement must end with a semicolon. Comments are delineated by `/*` and `*/` and may appear any place that *white space* may appear.

Note: The optional `ASSIGN` mode definition statement specifies the exact syntax required to identify pin and net names. See “ASSIGN” in *Encounter Test: Guide 2: Testmodes* for detail.

ALGORITHM Statement

This statement identifies an algorithm name specified in the MIC. All user control statements following this `ALGORITHM` statement and preceding the next `ALGORITHM` statement apply only to this algorithm. A `MACRO` statement must precede the `ALGORITHM` statement. The control statements that follow it apply only to the devices specified. Otherwise, the controls apply to any device processed for the algorithm. If `HOLD` and `CORRESPONDENCE` statements precede the first algorithm statement, they apply to all algorithms for a device.

Syntax:

```
ALGORITHM algorithmname;
```

or

```
ALGORITHM algorithmname , ... , algorithmname;
```

where:

`algorithmname` is an algorithm name specified in the MIC file for the device.

Syntax Rules:

- Multiple algorithm names may be specified in one statement.
- Case sensitivity applies to the algorithm name.

Semantic Rules:

- The algorithm must be found in the MIC file for the device.

Refer to "ALGORITHM statement" on page 226 for additional information.

CORRESPONDENCE Statement

This statement specifies the desired correspondence point for a device pin.

Verify Core Isolation allows both RPIs and correlated PIs (CPIs) to be used for user specified lineholds and correspondence points. Verify Core Isolation ensures that no more than one PI from a correlated set is specified for an operation.

Syntax:

```
CORRESPONDENCE devicepin = entity;
```

or

```
CORRESPONDENCE devciepin = entity , ... ,
devicepin = entity ;
```

where:

`devicepin` is the simple pin name

`entity` is:

for a primary I/0:

```
[PIN] hier_pin_name
```

for a scan cell:

```
[PIN] hier_pin_name of scan celloutput pin
BLOCK hier_block_name of scan cell
```

Syntax Rules:

- Only a single correspondence point is allowed per device pin.
- Device pin and correspondence point names are case sensitive.

Semantic Rules:

- One of these statement combinations must precede a `CORRESPONDENCE` statement:
  - ❑ `MACRO`
  - ❑ `MACRO` and `ALGORITHM`
  - ❑ `MACRO, ALGORITHM,` and `OPERATION`
- The device pin name must be in the simple form (relative to the block boundary).
- The entity name must be in the proper form.
- The entity can be only a primary input, a primary output, or a scan cell.

GROUP Statement

This statement defines a desired group of devices for the Verify Core Isolation run. Verify Core Isolation attempts to verify compatible isolation for the devices specified in this statement. Use this statement when Verify Core Isolation does not find the desired grouping or when you wish to reduce Verify Core Isolation run time by explicitly specifying your own grouping.

Syntax:

```
GROUP algorithmname/devicename , ... , devicename;
```

where:

`devicename` is a `hier_block_name`

`algorithmname` is an algorithm name specified in the MIC file for the device.

Syntax Rules:

- The device names are case sensitive.

Semantic Rules:

- All devices in a group must use the same MIC file.
- Each device specified must have a MACRO attribute specified in the model.
- Multiple GROUP statements are allowed.
- A device cannot appear in more than one GROUP statement for an algorithm.

HOLD Statement

This statement defines lineholds for this Verify Core Isolation run. Verify Core Isolation supports only hard lineholds in the user control file; it does not support default lineholds specified in the Encounter Test linehold file.

Syntax:

```
HOLD entity = value;
```

or

```
HOLD entity = value , ... , entity = value;
```

where:

`entity` is one of the following:

`[PIN]` *hier_pin_name*
`NET` *hier_net_name*
`BLOCK` *hier_block_name*

`value` is 0, 1, Z (or H), or X

Syntax Rules:

- Multiple entity/value pairs may be specified in one statement.
- Only a single value is allowed per entity.
- Only the entity names are case sensitive.

Semantic Rules:

- The entity name must be in the proper form.
- The entity can be only a primary input or a scan cell where:
  - The primary input is the highest level input pin of the hierarchy.
  - The scan cell is the pin corresponding to the output of the scan cell primitive.
- Other than these two rules, the general semantic rules for the Encounter Test Linehold utility apply.

MACRO Statement

This statement defines the device for which the user control statements following it and preceding the next `MACRO` statement apply. If `HOLD` statements are before the first `MACRO` statement, they apply to all devices. Note that this statement is not used to select devices to be processed. That is done with the Verify Core Isolation Additional Options window or the `macro` command line keyword.

Syntax:

```
MACRO devicename;
```

or

```
MACRO devicename , ... , devicename;
```

where:

`devicename` is a `hier_block_name`.

Syntax Rules:

- Multiple device names may be specified in one statement.
- The device names are case sensitive.

Semantic Rules

- Each block specified must have a MACRO attribute specified in the Encounter Test model.

OPERATION Statement

This statement defines the operation for which the user control statements following it and preceding the next `OPERATION` statement apply. An `ALGORITHM` statement must precede the `OPERATION` statement. The control statements that follow it apply to the specified `ALGORITHM`. Otherwise, the control is applied to any device processed for the `OPERATION`.

If `HOLD` or `CORRESPONDENCE` statements are before the first `OPERATION` statement for a device or an algorithm, they apply to all operations for this algorithm.

Syntax:

```
OPERATION operationname;
```

or

```
OPERATION operationname , ... , operationname;
```

where:

`operationname` is an operation name specified in the MIC file for the device.

Syntax Rules:

- Multiple operation names may be specified in one statement.

- The operation names are case sensitive.

Semantic Rules

- The operation must exist in the MIC file for the devices and algorithms it applies to.

B

Macro Isolation Control (MIC) Reference

Introduction

The Macro Isolation Control (MIC) file is used to define test requirements for embedded macro devices.This information drives both the Verify Core Isolation and the Create Core Tests applications. These applications test embedded devices by establishing specific, one-to-one correspondence paths between the embedded macro pins and SoC primary I/O pins and/or scan elements. By using these paths, macro test values can be applied to SoC inputs and scan elements and macro responses can be sampled at SoC outputs and scan elements to verify correct macro operation.

Note: This section refers to the terms *macro* and *core* interchangeably.

The MIC is typically supplied by the macro provider in conjunction with the tests to be applied to the embedded macro. In its most basic form, the MIC identifies the following types of test information:

- One or more sets of macro pin embedding requirements
- Identification of the test(s) which must be applied

In addition to the above, the MIC can also describe more advance macro test information such as:

- Concurrent test requirements
- The type of I/O used for establishing the pin correspondence enabling states

In order to facilitate the specification of this information in a reasonable and efficient manner, a variety of “statements” with *`keyword=value`* parameters are used in a hierarchical manner. An example MIC is shown Figure B-1 on page 223.

Figure B-1 MIC File Example

```
MIC_DEF;                                              /* begins all MICS (if used) */
```

```
ALGORITHM  NAME=WALK010,                          /* 1st test pattern set */
           OPERATIONS=READ,WRITE;

ALGORITHM  NAME=CHECKER,                          /* 2nd test pattern set */
           OPERATIONS=READ_PO,WRITE_PI;

OPERATION  NAME=READ,                             /* read pins */
           PINGROUPS=ADDRESS,DATAOUT,CONTROL,
           PRECONDITION_TYPE=PI_LATCH;

OPERATION  NAME=READ_PO                           /* read pins to POs */
           PINGROUPS=ADDRESS,DATAOUT_PO,CONTROL,
           PRECONDITION_TYPE=PI_LATCH;

OPERATION  NAME=WRITE,                            /* write pins */
           PINGROUPS=ADDRESS,DATAIN,CONTROL,
           PRECONDITION_TYPE=PI_LATCH;

OPERATION  NAME=WRITE_PI,                         /* write pins from PIs */
           PINGROUPS=ADDRESS,DATAIN_PI,CONTROL,
           PRECONDITION_TYPE=PI_LATCH;

PINGROUP   NAME=ADDRESS,                          /* all address pins */
           PINS=ADDR00:ADDR07,
           CORRESP_TYPE=PIPO;

PINGROUP   NAME=DATAIN,                           /* all data input pins */
           PINS=DIN00:DIN31,
           CORRESP_TYPE=ANY;

PINGROUP   NAME=DATAIN_PI,                        /* all data input pins */
           PINS=DIN00:DIN31,
           CORRESP_TYPE=PIPO;

PINGROUP   NAME=DATAOUT,                          /* all data output pins */
           PINS=DOUT00:DOUT31,
           CORRESP_TYPE=ANY;

PINGROUP   NAME=DATAOUT_PO,                       /* all data output pins */
           PINS=DOUT00:DOUT31,
           CORRESP_TYPE=PIPO;

PINGROUP   NAME=CONTROL                           /* all control pins */
           PINS=READ,WRITE,
           CORRESP_TYPE=PIPO;
```

In the preceding example, the basic elements of the test requirements are specified in a simple and logical manner. Each MIC can start with the `MIC_DEF` statement. This statement identifies the file as a Macro Isolation Control file and supports the specification of a date/time stamp for control purposes.

The test sets to be applied are then defined using one or more `ALGORITHM` statements. Each `ALGORITHM` statement also references the pin collections required by that test set. The collections themselves are defined by one or more `OPERATION` statements. The term "OPERATION" comes from the fact that the pin collections are used to "operate" the device, i.e. cause something to happen to the device or cause the device to respond in some manner.

In the above example, there are four OPERATION statements used: WRITE and WRITE_PI, which write values into the memory using its address, data input, and control pins, and READ and READ_PO, which read values from that memory by using its address, data output, and control pins.

Specifying the pins in this way tends to match their functional usage and thus increases the chances of finding non-conflicting correspondence states. For example, it is common to connect embedded memory data input and output pins to the same bidirectional bus. If all pins were referenced by a single OPERATION, then simultaneous correspondence would be required for all pins. This would be impossible to establish due to conflicting use of the bus. By specifying them individually using multiple OPERATION statements unique, non-conflicting correspondence sets can be successfully established. In other words, each OPERATION statement specifies the pin sets requiring simultaneous correspondence such that some portion of the test set can be mapped to SoC I/O.

Each OPERATION statement identifies the pins within the collection and their test connection requirements by referencing one or more PINGROUP statements. It is the PINGROUP statement that actually defines the embedding requirements for each pin.

Note that a PINGROUP statement may be referenced by more than one OPERATION statement and that OPERATION statements may be referenced by more than one ALGORITHM statement if their test connection requirements are identical. If not, then a different collection using the same pins, but with different test connection requirements can be defined and used as necessary.

Terminology

The following terms are used in the presentation of the MIC statement syntax:

Term	**Definition**
Algorithm	A specific set of tests to be applied to an embedded macro.
Concurrent Test	Testing multiple macros (a group) at the same time using the same test patterns. This implies that at least one operation, but not necessarily all operations, can be done concurrently.

`Correspondence`	A one-to-one relationship between a macro pin and a SoC primary I/O pin or scannable element. Usually, a preconditioning state is required to enable the path.
`Isolation Set`	A matched set of correspondence paths and preconditioning state such that the application of the preconditioning state enables a one-to-one relationship between a macro pin and an SoC primary I/O pin or scannable element. More than one isolation set may exist.
`Macro Group`	A set of macros that can be tested concurrently.
`Operation`	A set of macro pin groups used to apply some portion of a test set.
`Pin Group`	Any set of macro I/O pins which are collected together, usually because they require the same correspondence characteristics.
`Preconditioning`	A design state which establishes its associated correspondence paths. The state may include values on both SoC primary I/O and scan elements. Note that preconditioning states are not specified in the MIC. Verify Core Isolation established these states based on the correspondence paths required to isolate the various pins.

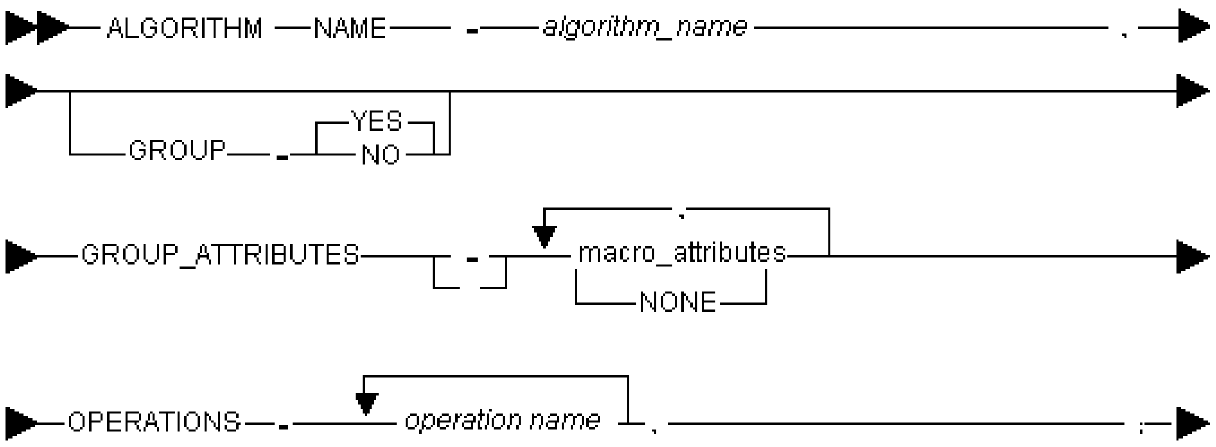
MIC Statement Syntax

The following presents the various MIC statements syntax and keyword descriptions in alphabetical order. Commentary, delimited by /* and */, can be included anywhere and “white space” is allowed.

ALGORITHM statement

The `ALGORITHM` statement is used to identify a test set to be applied to an embedded device and the conditions under which that test set is applied. Only one test set can be identified per statement, but multiple statements can be used to identify multiple test sets. At least one `ALGORITHM` statement is required.

Figure B-2 Algorithm Statement

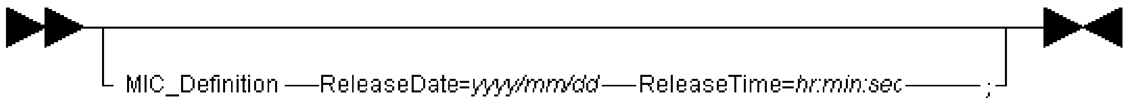


`NAME`	Specifies the name of a test set to be applied to this macro. This name correlates to the TB_TDMFILE_*`algorithm_name`* model attribute used to identify the test set source file. This keyword is required.
`GROUP`	Specifies whether macros using this test set must be grouped for concurrent testing. This keyword is optional. The default is `YES`. ■ `YES` - indicates macros must be grouped for concurrent test. ■ `NO` - indicates macros must not be grouped for concurrent test.
`GROUP_ATTRIBUTES`	For macros that use this MIC, this required keyword indicates which macro attributes must be identical for macros to be grouped together. ■ `macro_attributes` - specify a list of macro attributes. Only macros with the same attribute value are grouped.
`OPERATIONS`	Specifies the name of one or more `OPERATION` statement used by this test set. Each `OPERATION` statement specifies a set of pins used to apply some portion of the test set. This keyword is required.

MIC_DEF Statement

The MIC_DEF statement identifies the file as a Macro Isolation Control file and provides for the specification of a date/time stamp for control purposes. This statement is optional, but if used it must be the first statement in the file.

Figure B-3 MIC_DEF Statement

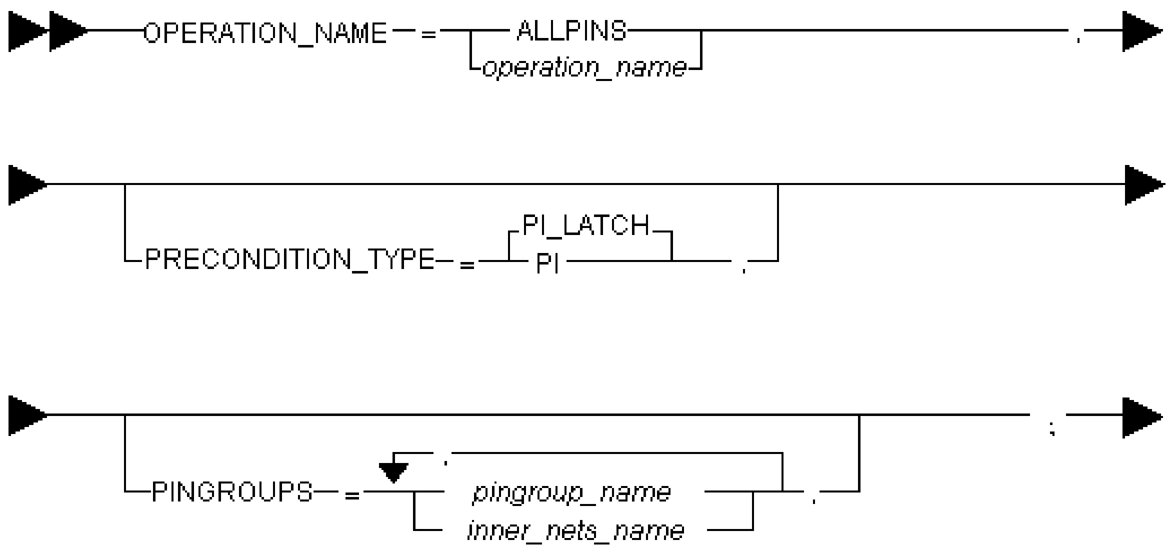


`ReleaseDate`	Indicates an applicable date in *`yyyy/mm/dd`* format. This keyword must be specified if the `MIC_DEF` statement is used. There is no default.
`ReleaseTime`	Indicates an applicable time in *`hh:mm:ss`* format. This keyword must be specified if the `MIC_DEF` statement is used. There is no default.

OPERATION Statement

The `OPERATION` statement specifies a set of macro pin groups which support the mapping of some portion of a test set. All pins specified on an `OPERATION` statement must be simultaneously isolated. Multiple `OPERATION` statements may be used, as necessary, to define multiple collections of pin groups which support the mapping of the entire test set. At least one `OPERATION` statement is required.

Figure B-4 OPERATION Statement

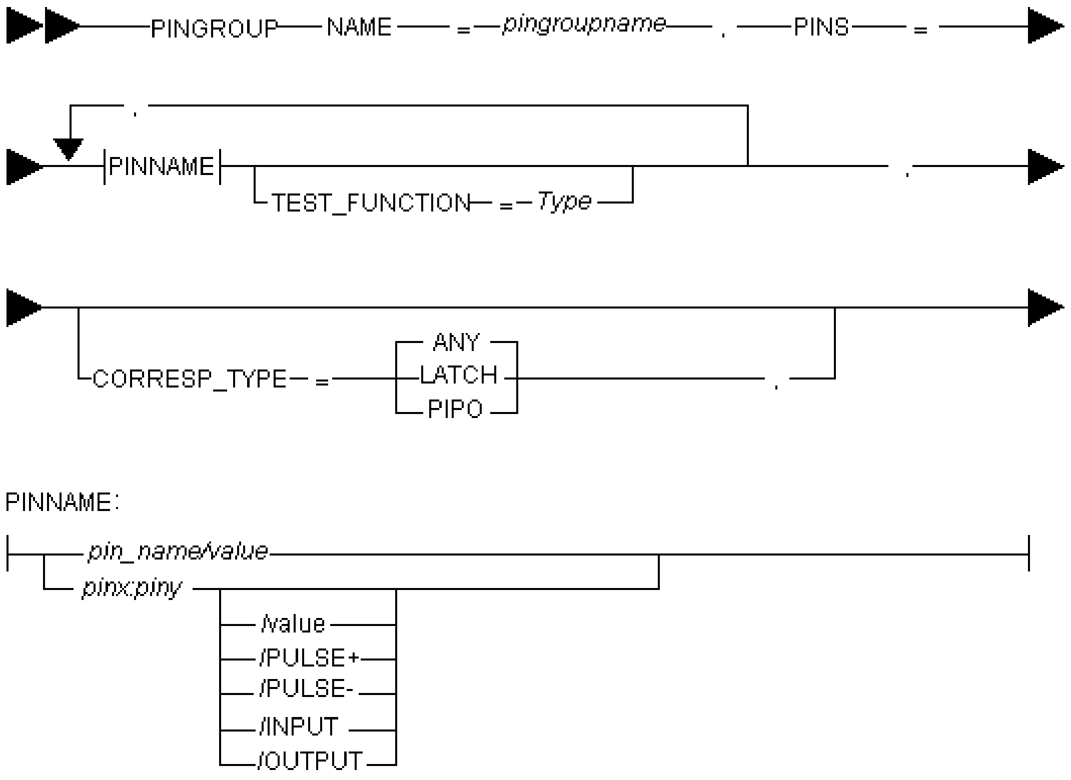


NAME	Specifies a name for this `OPERATION` statement. This name is referenced by the `ALGORITHM` statement `OPERATIONS` keyword. This keyword is required.
PRECONDITION_TYPE	Specifies the type of preconditioning I/O that must be used to simultaneously establish the correspondence paths for the pins identified for this operation. This keyword is optional. The default is `PI_LATCH`. ■ `PI` - indicates that only primary inputs may be used to establish the preconditioning state for this operation. ■ `PI_LATCH` - indicates that either, or both, primary inputs and scan elements may be used to establish the preconditioning state for this operation.
PINGROUPS	Specifies the names of one or more `PINGROUP` statements used by this operation. All pins referenced by these `PINGROUP` statements require simultaneous correspondence.

PINGROUP Statement

One or more `PINGROUP` statements are used to define the embedding requirements for each macro pin. The selection of pins which are included on each statement is typically driven by the commonality of these requirements. However, these groupings are actually arbitrary. At least one `PINGROUP` statement is required.

Figure B-5 PINGROUP Statement



`NAME`	Specifies a name for this `PINGROUP` statement. This name is referenced by an `OPERATION` statement `PINGROUPS` keyword. This keyword is required.

`PINS` — Specifies a list of the following information for each pin in the pin group:

- `pinname`

  This required field provides for the specification of one of the following pin name specifications:

  - *`pin_name`*

    A macro input or output pin name which requires correspondence to be established. This name must match the name of a pin on the macro block in the model.

  - *`pin_name:pin_name`*

    A range of macro input or output pin names. The names must end in one or more numeric characters. The range is expanded by sequentially incrementing, or decrementing, the numeric characters starting with the first pin_name and ending with the last pin_name. Each expanded name must match the name of a pin on the macro block in the model.

    For example, a range specified as addr00:addr07 expands to encompass pins addr00, addr01, addr02, addr03, addr04, addr05, addr06, and addr07. Similarly, a range of di02:di00 expands to encompass pins di02, di01, di00.

    Note: The specified lengths of the two ranges must be equal. For example, `XYZ000:XYZ10` is an invalid specification; the first range is 6 characters and the second range is 5 characters.

  - *`pin_name/value`*

    An associated pair of a macro input or output pin and a value. The value can be 0, 0a, 1, 1a, Z or Za. Values of 0, 1, or Z apply to input pin states that are required to be established on the pin, either by the TI/TIE state or by preconditioning. Values of 0a, 1a, and Za apply to output pins which are assumed to be at this constant value during test. Correspondence is not required, nor is it established, for pins with specified values.

- *`pin_name/type`*

  An associated pair of a macro input or output pin name and one of the following pin types:

  - `PULSE+`

    Indicates that the macro pin(s) specified by <pinname> require a positive pulse (0 -> 1 -> 0).

  - `PULSE-`

    Indicates that the macro pin(s) specified by <pinname> require a negative pulse (1->0->1).

  - `INPUT`

    Indicates that the macro pin(s) specified by <pinname> are bidirectional pins that are to be treated as inputs only for operations using this pingroup.

  - `OUTPUT`

    Indicates that the macro pin(s) specified by <pinname> are bidirectional pins that are to be treated as output only for operations using this pingroup.

- `TEST_FUNCTION`

  Limits the choice of correspondence pin selections to PIs which have this test function pin attribute for the associated *`pinname`* macro pin(s). This keyword is optional. If not coded, test function pin attributes are not considered in the selection process.

  Allowable values are: `A_SHIFT_CLOCK, B_SHIFT_CLOCK, E_SHIFT_CLOCK, P_SHIFT_CLOCK, A_SHIFT_SYSTEM_CLOCK, B_SHIFT_SYSTEM_CLOCK,E_SHIFT_SYSTEM_CLOCK, SYSTEM_CLOCK, SCAN_INPUT, and SCAN_OUTPUT`.

`CORRESP_TYPE`	Specifies the type of correspondence required for all pins in the pin group. This keyword is optional. The default is `ANY`.

`ANY`

Correspondence may be made to a SoC primary input, primary output, or scan element (latch or flip-flop).

`LATCH`

Correspondence must be made to a SoC scan element (latch or flip-flop).

`PIPO`

Correspondence must be made to a SoC primary input or primary output.

Advanced MIC Functions

The preceding describes basic Macro Isolation Control file function which is applicable to the majority of cases. However, the MIC supports more advanced function for special cases such as BIST testing of embedded devices.

The syntax and functional description for this advance function is documented in "Advanced Techniques for Core Test" in the *Encounter Test: Reference: Legacy Functions*.

Index