ATPG Flow with Modus DFT Software Solution

Course Version 22.1

Lab Manual

Revision 1.0

cādence®

Printed in the United States of America.

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

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence customers in accordance with, a written agreement between Cadence and the customer.

Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Table of Contents

ATPG Flow with Modus DFT Software Solution

Module 1: About This Course

No Labs for This Module

Module 2: Introduction to Modus DFT Software Solution

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

Objective: To bring up and execute Modus tool and learn about the ATPG directory structure.

This lab is intended to introduce the invocation of Modus DFT Software Solution and the ATPG directory structure. Modus offers a Tcl based command line interface which is tightly integrated with the GUI.

It is assumed that the users have Modus and Xcelium™ available in the environment:

```
setenv CDS_LIC_FILE yourLicenseFilePath
setenv MODUS_PATH {MODUS_Install_Path}
setenv IUS_PATH {IUS_Install_Path}
set path = ( $path $IUS_PATH/tools/bin $MODUS_PATH/tools/bin )
```

You can verify the availability of the tools in environment at the command prompt with:

- `which modus`
- `which xrun`

Bring up and Execute Modus

To invoke the Modus command line:

```
user_prompt:/> modus
```

Modus GUI can be invoked directly from the shell.

```
user_prompt:/> modus -gui
```

Note: By default, Modus will create a modus.log log file and a modus.cmd file containing the summary of commands executed. Users can change the name of log file and command file with –log option.

Alternatively, the GUI can be invoked from Modus Tcl command line:

```
modus:/> gui_open
```

Quick help on the commands can be obtained with

```
modus:/> help -command <command_name>
```

To obtain the detailed information on a command option

```
modus:/> help -command <command -option> -detail
```

To bring up the new Cadence Help systems (cdnshelp), just type in the following command at the Modus prompt:

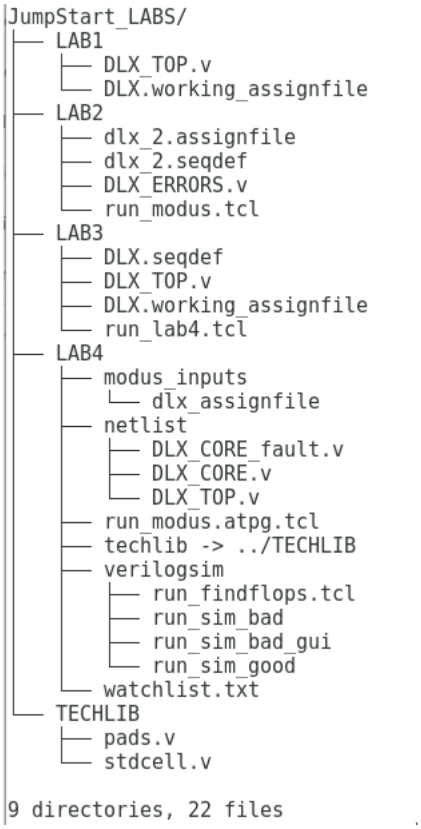
```
modus:/> cdnshelp
```

To get a better understanding of a particular message

```
modus:/> msgHelp <INFO/WARNING/ERROR message>
```

Directory Structure

The following is the directory structure:



```
JumpStart_LABS/
├── LAB1
│   ├── DLX_TOP.v
│   └── DLX.working_assignfile
├── LAB2
│   ├── dlx_2.assignfile
│   ├── dlx_2.seqdef
│   ├── DLX_ERRORS.v
│   └── run_modus.tcl
├── LAB3
│   ├── DLX.seqdef
│   ├── DLX_TOP.v
│   ├── DLX.working_assignfile
│   └── run_lab4.tcl
├── LAB4
│   ├── modus_inputs
│   │   └── dlx_assignfile
│   ├── netlist
│   │   ├── DLX_CORE_fault.v
│   │   ├── DLX_CORE.v
│   │   └── DLX_TOP.v
│   ├── run_modus.atpg.tcl
│   ├── techlib -> ../TECHLIB
│   ├── verilogsim
│   │   ├── run_findflops.tcl
│   │   ├── run_sim_bad
│   │   ├── run_sim_bad_gui
│   │   └── run_sim_good
│   └── watchlist.txt
└── TECHLIB
    ├── pads.v
    └── stdcell.v

9 directories, 22 files
```

Tool Versions

The labs have been designed with the following software versions. Some of the messages and log lines explained in the instructions may not appear with other versions.

- Modus 22.10
- XCELIUM xm-Verilog simulator XCELIUM22.01.001

Unzip the Database and Run the Lab

To Unzip and untar Modus_22_1.tar.gz

```
tar -zxvf Modus_22_1.tar.gz
```

To change the directory:

```
cd JumpStart_LABS
```

End of Lab

Module 3: The ATPG Flow

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

Objective: To learn the Modus ATPG Flow and get familiar with Modus Tcl and GUI Interface.

In this lab, you will go through the basic Modus ATPG Flow and learn about the Modus GUI and Tcl interface.

Invoke the Modus

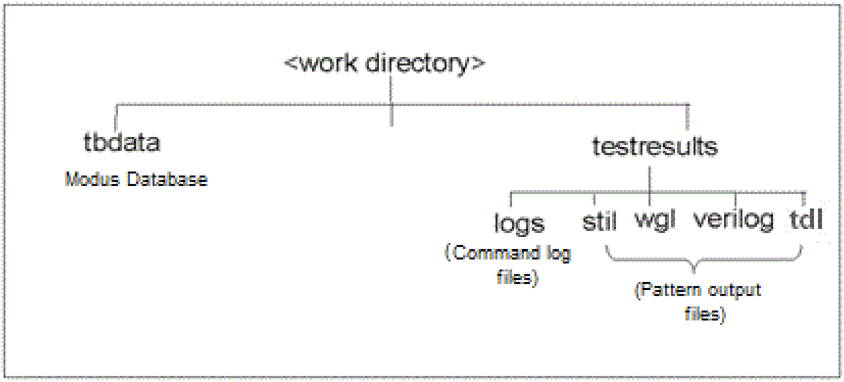
```
user_prompt:/> cd JumpStart_LABS/LAB1
```

```
user_prompt:/> modus
```

Note: Once you invoke Modus you should see Modus Tcl prompt.

Setup Modus Workdir

Modus stores its database in a work directory and the following figure shows the directory structure created by Modus for a work directory.



By default, if a workdir is not explicitly specified, Modus will use the current directory as workdir and create tbdata and testresults directories as shown above in Figure. To explicitly set the workdir use the command:

```
set_db workdir .
```

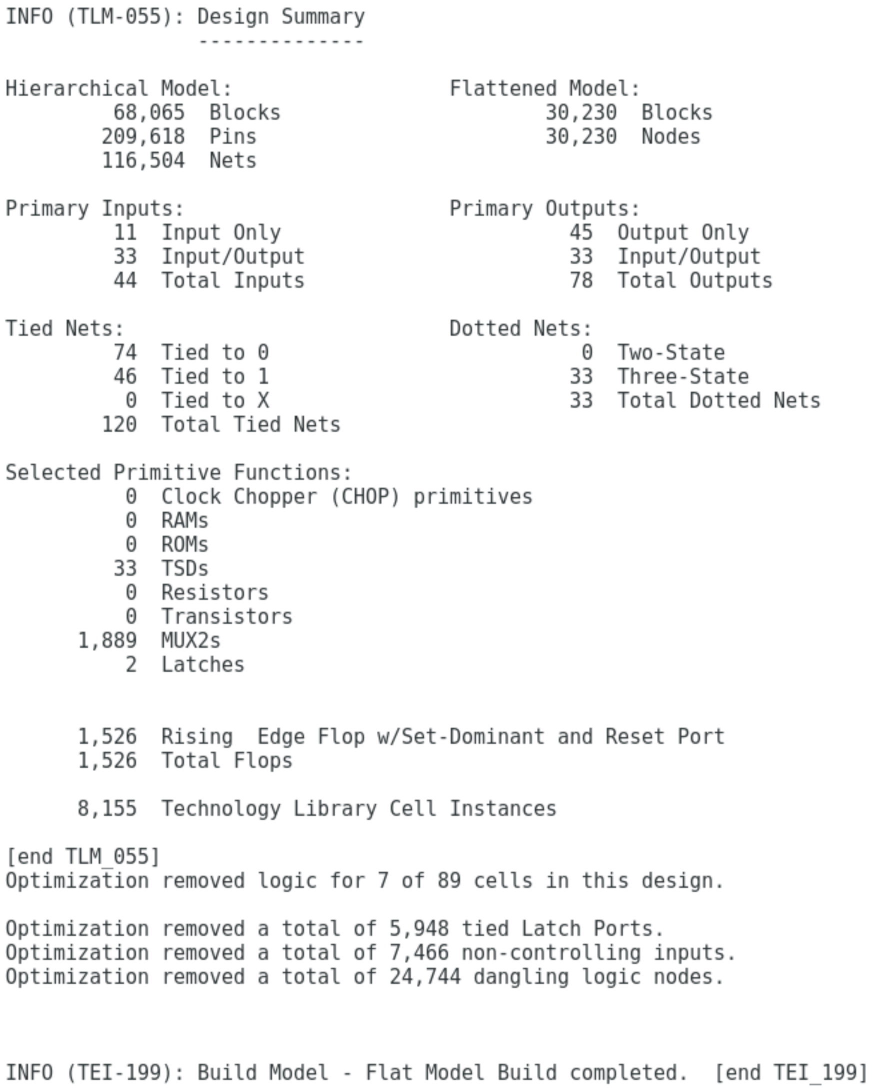
Building the Modus Test Model

In this step we will demonstrate how Modus can read your netlist and libraries and compile them into a Modus test model. This test model will be used for all future steps.

On your Modus command line run the following command.

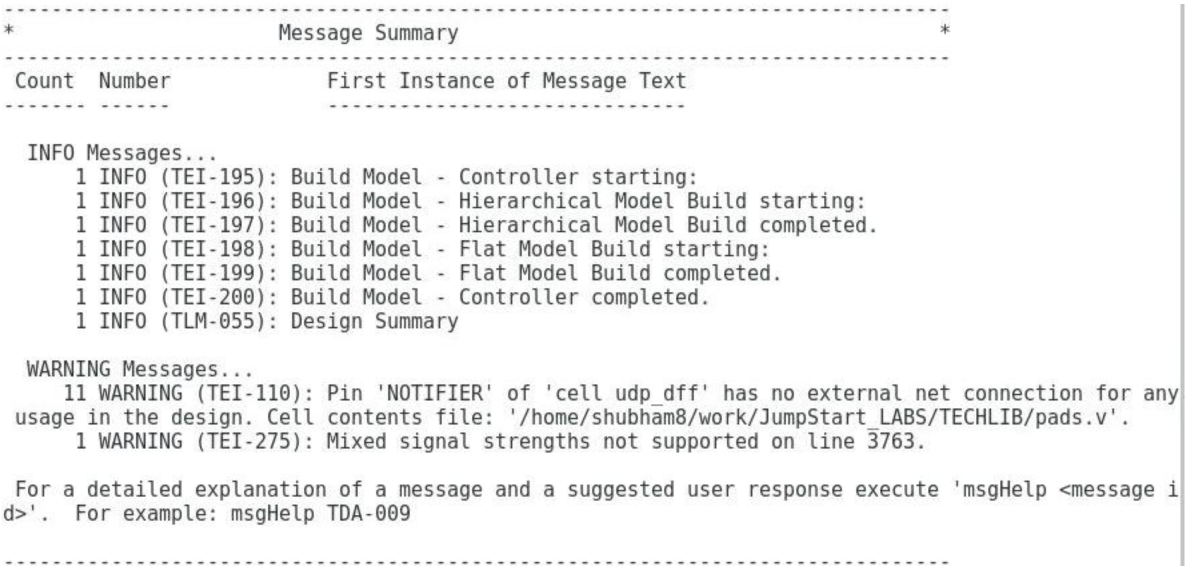
```
       build_model –cell DLX_TOP –designsource ./DLX_TOP.v      \
         –techlib ../TECHLIB/*.v -allowmissingmodules yes        \
         -blackboxoutputs z
```

The command will read in your libraries and netlist to compile a Modus test model. At the end of the build_model processing you should find the fault model statistics summary which provides the number of various objects identified in the design.



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

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

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

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

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


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

      8,155  Technology Library Cell Instances

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

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



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

You should also see a summary of any warnings/errors that tool encounters while processing the data.



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

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

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

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

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

Users are required to review and analyze any Errors/Warnings flagged by the tool.

What are the TEI-110 messages?

Review the Errors and Warnings Inside Modus GUI

Modus offers extensive GUI debugging capabilities. Let’s take a first look on how to review the Errors/Warnings inside GUI. From your Modus command line, invoke the GUI using the following command.

You must explicitly setup another “set_db workdir. “This is necessary at this point after you build_model because, this is a new tbdata database. Setting the command before you run build_model will not retain this value. You will get the following error message:

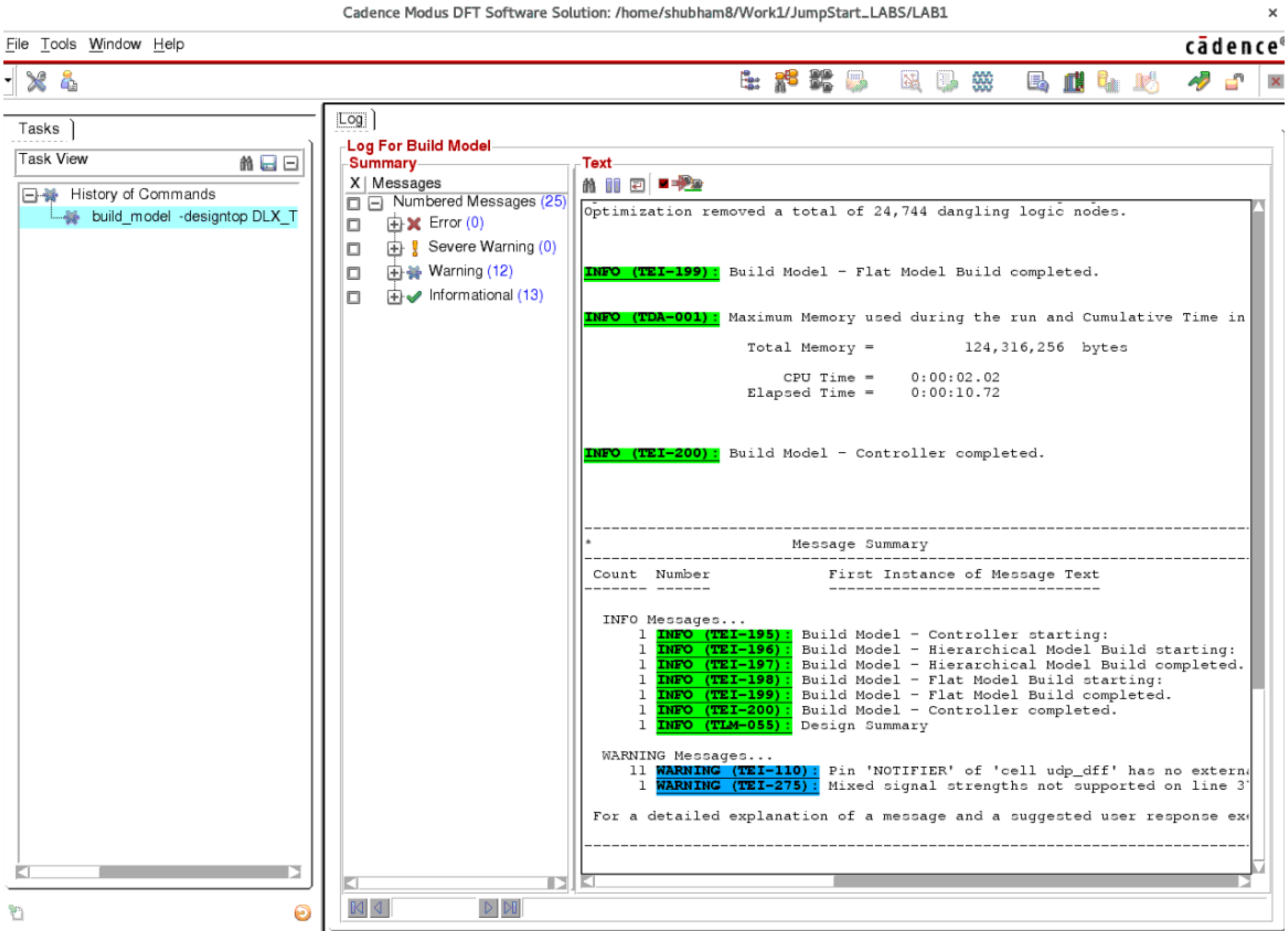
```
ERROR (TUI-809): gui_open invocation requires an explicit invocation of
   'set_db workdir ‘to the current working directory.  [end TUI_809]
```

This is not the case if there is already a tbdata in the directory.

To open the Modus GUI, at the Modus prompt use the command:

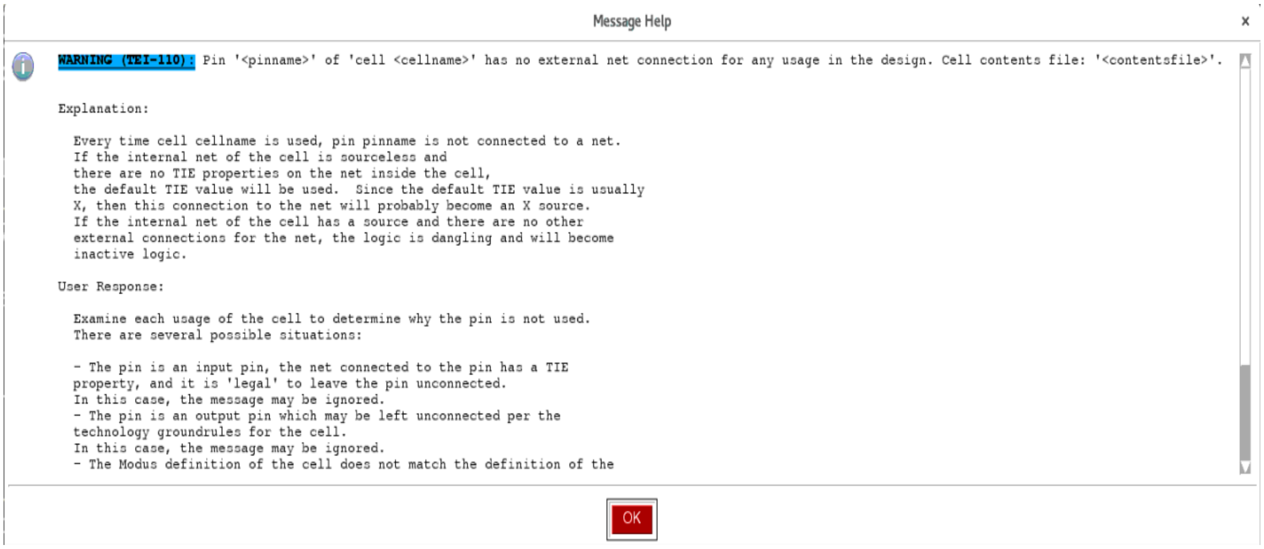
```
gui_open
```

You should now see the GUI window with a TASKs pane on the left. Click on the build_model task that was run. If you click on one of the Warnings in the Log file output **(Log Tab)**, what information is shown in the Log.



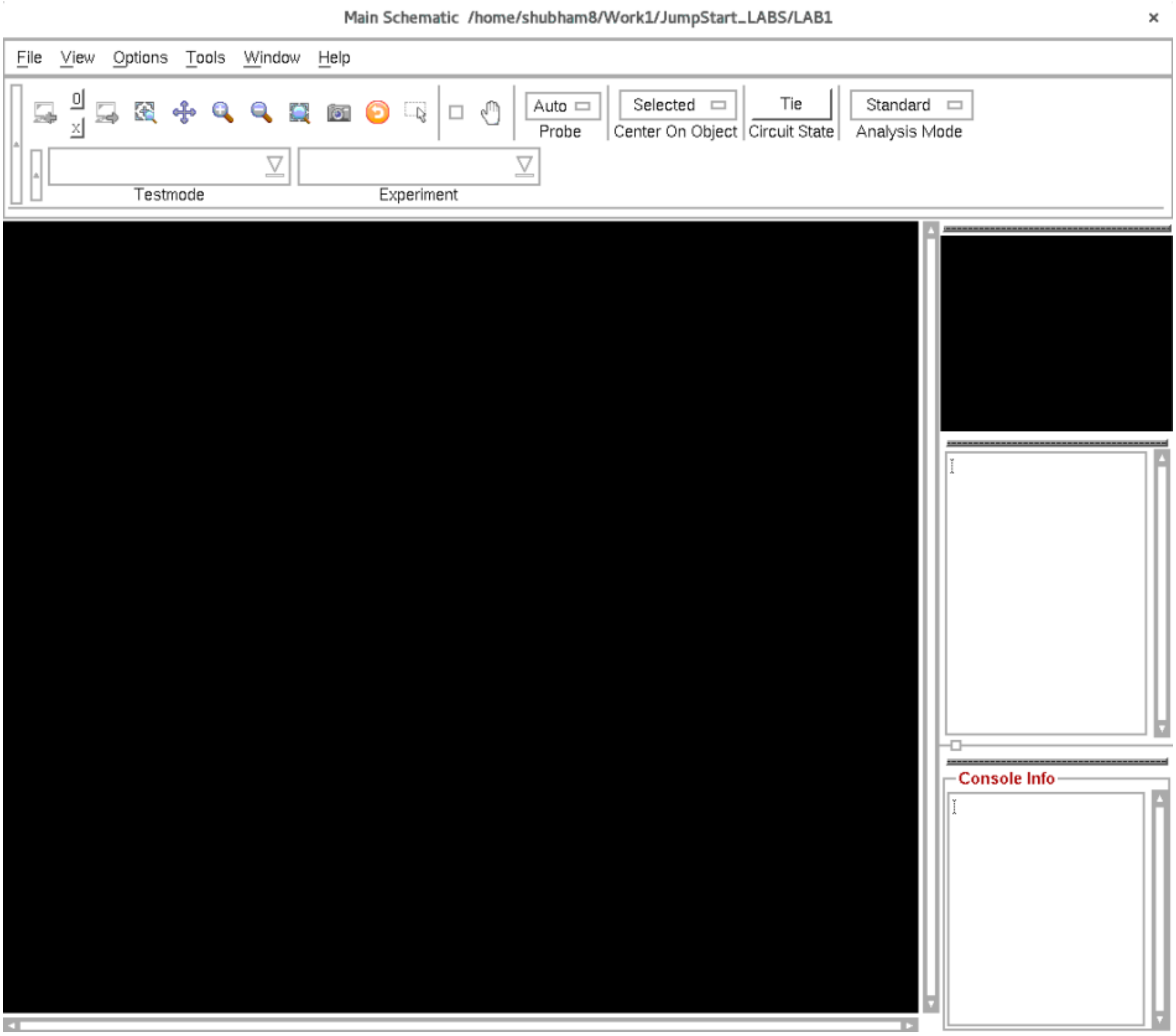
```
Optimization removed a total of 24,744 dangling logic nodes.


INFO (TEI-199): Build Model - Flat Model Build completed.


INFO (TDA-001): Maximum Memory used during the run and Cumulative Time in

                     Total Memory =           124,316,256  bytes

                         CPU Time =     0:00:02.02
                     Elapsed Time =     0:00:10.72



INFO (TEI-200): Build Model - Controller completed.




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

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

  WARNING Messages...
     11 WARNING (TEI-110): Pin 'NOTIFIER' of 'cell udp_dff' has no externa
      1 WARNING (TEI-275): Mixed signal strengths not supported on line 3

 For a detailed explanation of a message and a suggested user response exe

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

Now search for a Verilog file reference in the Warnings in your Log, such as the TEI-110 warnings. When you mouse over it, a hyperlink is visible. Clicking on the TEI-110 hyperlinked warning opens with a window furnishing more details that helps user to debug the warning.



```
WARNING (TEI-110): Pin '<pinname>' of 'cell <cellname>' has no external net connection for any usage in the design. Cell contents file: '<contentsfile>'.


Explanation:

  Every time cell cellname is used, pin pinname is not connected to a net.
  If the internal net of the cell is sourceless and
  there are no TIE properties on the net inside the cell,
  the default TIE value will be used.  Since the default TIE value is usually
  X, then this connection to the net will probably become an X source.
  If the internal net of the cell has a source and there are no other
  external connections for the net, the logic is dangling and will become
  inactive logic.

User Response:

  Examine each usage of the cell to determine why the pin is not used.
  There are several possible situations:

  - The pin is an input pin, the net connected to the pin has a TIE
  property, and it is 'legal' to leave the pin unconnected.
  In this case, the message may be ignored.
  - The pin is an output pin which may be left unconnected per the
  technology groundrules for the cell.
  In this case, the message may be ignored.
  - The Modus definition of the cell does not match the definition of the
```

Let’s try out the Hyperlinking. In the Log Tab, select the “Toggle Highlight” button. This will also pop open the schematic window. You can close this window. It is helpful during design rule checking.



Build the Test Mode

Build Test Mode identifies the test structures (scan chains, test pins, compression structures, and so on) required to generate patterns or to perform diagnostics. It is called a test mode because it defines a certain test configuration (mode) for the design. The command to do this is called build_testmode.

We need to tell the tool what the different test pins are and their associated functions. For this circuit we have the following test pins.

- **16 Scan Ins:** DLX_CHIPTOP_DATA[15:0]
- **16 Scan Outs:** DLX_CHIPTOP_DATA[31:16]

- **Scan Enables:**
  - DLX_CHIPTOP_SE (Active high asserted)
  - DLX_CHIPTOP_RESET2 (Active low asserted)
    - You can treat a reset as a scan enable to test more faults on the reset line.
    - This means during shift it will be set to its de-asserted value but can change during capture if the tool desires.
- Test Enables: (Tied to 1)
  - DLX_CHIPTOP_TEST_ENABLE
  - DLX_CHIPTOP_TCK
- Test Enables: (Tied to 0)
  - DLX_CHIPTOP_TDI
  - DLX_CHIPTOP_TMS
  - DLX_CHIPTOP_TRST
- Clocks used for Shift and Capture:
  - DLX_CHIPTOP_TEST_CLOCK (Off state is 0)
  - DLX_CHIPTOP_SYS_CLK (Off state is 0)
- Clocks used only for Capture:
  - DLX_CHIPTOP_RESET (Off state is 1)

You now need to create an assign file for Modus that contains the above pin information. Here are a few examples to get you started:

```
assign pin=DLX_CHIPTOP_DATA[15] test_function=SI;
assign pin=DLX_CHIPTOP_SE test_function=+SE;
assign pin=DLX_CHIPTOP_TCK test_function=+TI;
```

Attempt to create this file first before looking at the working one. Then compare your assign file to the file DLX.working_assignfile.

Did the polarity on the test_functions match?

Is your syntax correct?

Did the pin hold the correct test functionality as intended?

Now, run the build_testmode command to build a test view of your circuit based on the pin assignments set in the assignfile.

```
build_testmode -testmode FULLSCAN -assignfile <your_assignfile>
```

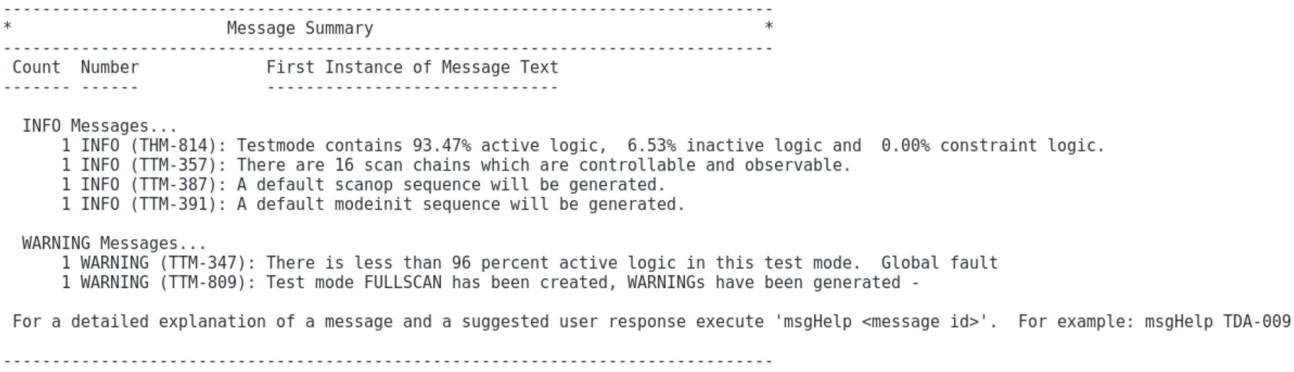
Once the command is executed, look for any Severe Warning messages or syntax errors.

What is the active logic number and what does it mean?

Any info on scan chains in this log file?

Do we have valid scan chains?

If you inserted assign file “DLX.working_assignfile” to build testmode, summary at the end of the building the testmode is:



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

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

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

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

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

Report Test Structures

Before we run full design-rule checking, we can see if our testmode is set up properly and whether we have continues chains from SI to SO, by using the command:

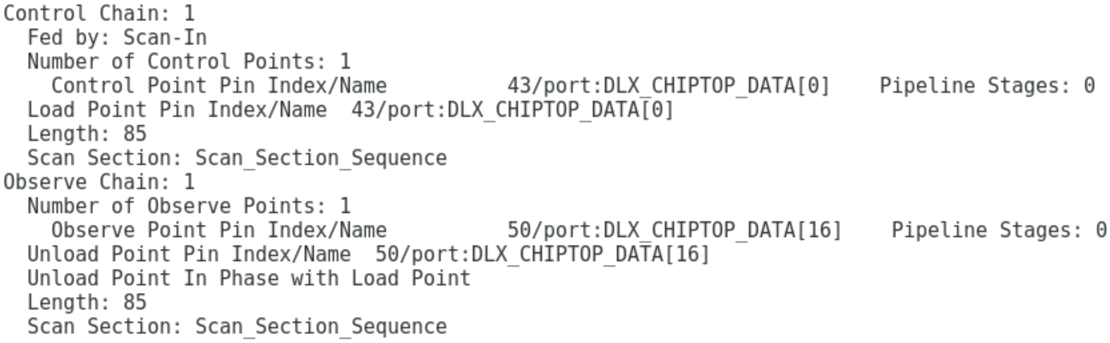
```
report_test_structures -testmode FULLSCAN
```

This tells the tool to trace the scan chains from SO backwards and SI forwards.

How many Controllable chains do we have?

How many Observable chains do we have?

A good chain in the report file would look like this:



```
 Control Chain: 1
   Fed by: Scan-In
   Number of Control Points: 1
     Control Point Pin Index/Name          43/port:DLX_CHIPTOP_DATA[0]    Pipeline Stages: 0
   Load Point Pin Index/Name  43/port:DLX_CHIPTOP_DATA[0]
   Length: 85
   Scan Section: Scan_Section_Sequence
 Observe Chain: 1
   Number of Observe Points: 1
     Observe Point Pin Index/Name          50/port:DLX_CHIPTOP_DATA[16]    Pipeline Stages: 0
   Unload Point Pin Index/Name  50/port:DLX_CHIPTOP_DATA[16]
   Unload Point In Phase with Load Point
   Length: 85
   Scan Section: Scan_Section_Sequence
```

Note: The Bit Lengths of the Controllable and Observable trace are of identical length. This means the tool traced backwards and forwards the same equal length from SI to SO and SO to SI.

A bad chain might look like this:

```
Control Chain: 2

    Fed by: Scan-In

    Number of Control Points: 1

      Control Point Pin Index/Name
   44/port:DLX_CHIPTOP_DATA[10]    Pipeline Stages: 0

    Load Point Pin Index/Name  44/port:DLX_CHIPTOP_DATA[10]

    Length: 84

    Scan Section: Scan_Section_Sequence


<<<<line spacing between them>>>>


Observe Chain: 2

    Number of Observe Points: 1

      Observe Point Pin Index/Name
   51/port:DLX_CHIPTOP_DATA[17]    Pipeline Stages: 0

    Unload Point Pin Index/Name  51/port:DLX_CHIPTOP_DATA[17]

    Unload Point In Phase with Load Point

    Length: 85

    Scan Section: Scan_Section_Sequence
```

Note: The Bit Lengths are different. When the log file prints out the chain report, they will also not be listed together. This is because the tool does not know they are supposed to belong to the same scan chain. This will be more evident in LAB2 when we have broken chains.

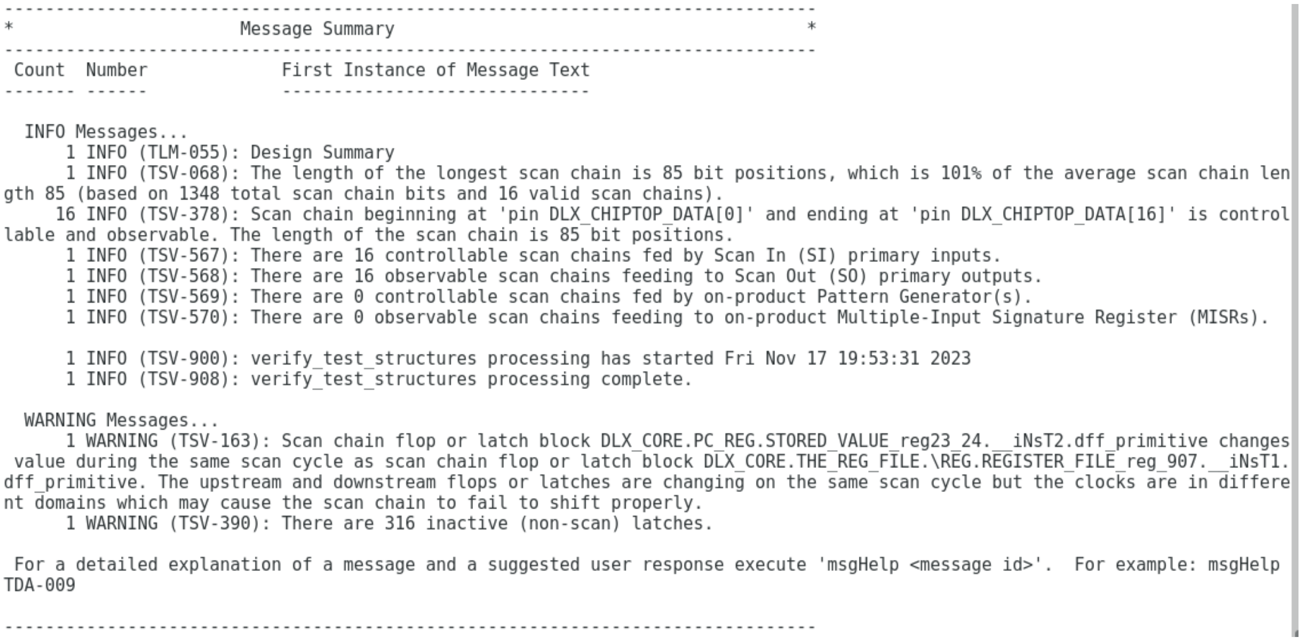
Note: When you get into compression structures the format is a bit different. You will have multiple Control Reg points feeding to multiple Observer Reg points.

Verify Test Structures

This step will run the full design rule checking for your circuit. This checks everything from clock race conditions, tri-state burnout to corruptible registers in capture mode. The types of checks you run is totally user controllable. To run the full design rule for your circuit, use the command:

```
verify_test_structures -testmode FULLSCAN
```

Once the command is executed look for Informational message **TSV-378** which identifies how many complete scan chains were found. The summary of verify_test_structures is:



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

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

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

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

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

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

Note: Look for any Severe Warning or Error messages in the Message Summary section of the log file. If these occur, you need to verify that you understand them and handle each one appropriately.

Do you have any TSV-390 messages in the verify_test_structures summary? If so, how can you get a listing of all the floating/non-scan flops?

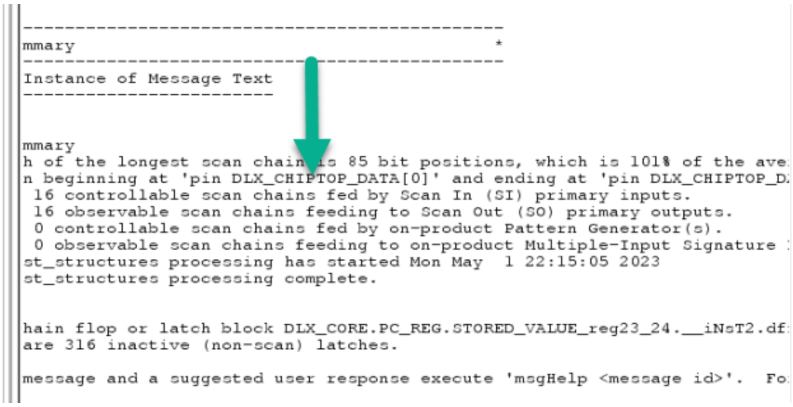
To get the list of floating/inactive latches, use the command:

```
report_test_structures -testmode FULLSCAN -reportreginactive
```

This command will report all the inactive latches.

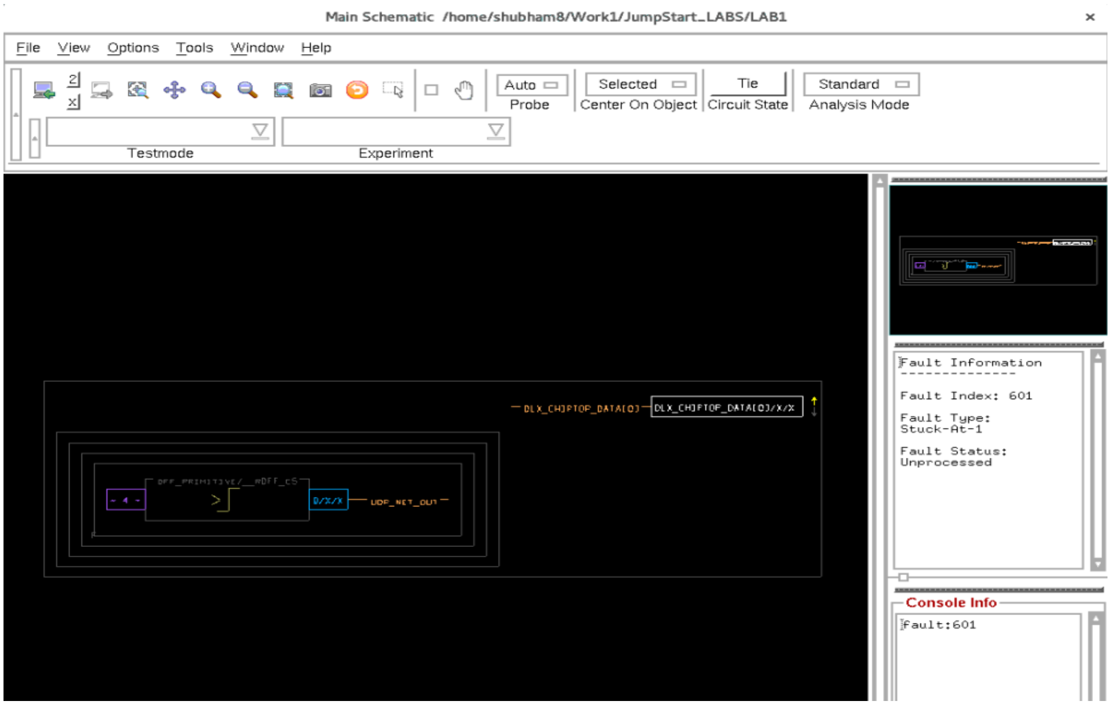
In your GUI window, if you still have the hyperlinking turned on then you can explore its functionality more here.

1. Refresh the Task pane by clicking the yellow arrow at the bottom.
2. Click on verify_test_structures step in your Tasks pane.
3. Search in the Text pane for any of the TSV-378 messages. These list the chains that Modus finds. If you scroll to the right of any of these messages you will see a reference to the SI and SO pins that make up that particular chain.
4. Mouse over the SI or SO signal name. You will notice a hyperlink show up.



```
-----------------------------------------------
mmary                                          *
-----------------------------------------------
Instance of Message Text
------------------------


mmary
h of the longest scan chain is 85 bit positions, which is 101% of the ave
n beginning at 'pin DLX_CHIPTOP_DATA[0]' and ending at 'pin DLX_CHIPTOP_D
 16 controllable scan chains fed by Scan In (SI) primary inputs.
 16 observable scan chains feeding to Scan Out (SO) primary outputs.
 0 controllable scan chains fed by on-product Pattern Generator(s).
 0 observable scan chains feeding to on-product Multiple-Input Signature
st_structures processing has started Mon May  1 22:15:05 2023
st_structures processing complete.


hain flop or latch block DLX_CORE.PC_REG.STORED_VALUE_reg23_24.__iNsT2.df
are 316 inactive (non-scan) latches.

message and a suggested user response execute 'msgHelp <message id>'.  Fo
```

5. Click on the hyperlink. This will pop up the schematic viewer and highlight that pin in the circuit.



```
Fault Information
----------------

Fault Index: 601

Fault Type:
Stuck-At-1

Fault Status:
Unprocessed
```

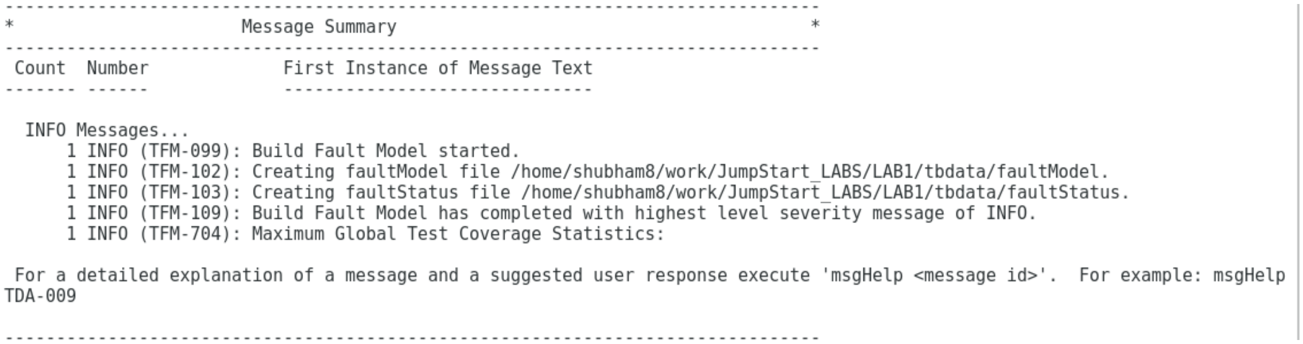
```
fault:601
```

Build the Fault Model

Next, we need to build the global fault model. Run the following command to place the faults on the appropriate nodes.

```
build_faultmodel
```

Summary at the end of build_fault model in the log file is shown below:



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

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

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

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

Note: If you find TFM-109 severe warning info message in the build_faultmodel log file, then you can-not move forward. The expected highest severity is INFO. If the highest severity is reported, you must investigate the associated messages to determine if they are acceptable for the design. It is mandatory to check this info message in the log, then only you can move forward.

Running **build_faultmodel** will only include static fault modeling by default. To include dynamic faults also, user can turn on dynamic fault by using the **-includedynamic on**.

By default, the faults are only marked at the cell boundary as per accepted industry practice. You can also tell the tool to apply faults inside the cell boundary. For this lab you should use the default, i.e., apply the faults on the cell boundary. Observe the build_faultmodel log file and look for the below points.

How many Static Faults are there in the entire design?

How many Static faults are active in the FULLSCAN mode?

Exiting the Modus

You have now completed the netlist processing and are ready to generate patterns. Go ahead and close the GUI by clicking the **X** in the upper right-hand side of the GUI or by selecting **File -> Exit GUI and Console**, or use the command:

```
exit
```

End of Lab

Lab 3-2 Generating ATPG Vector

Objective: To perform ATPG vector generation, and writing out the Verilog and WGL patterns.

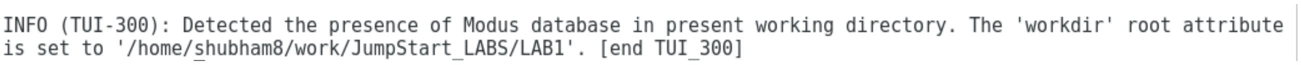
In this lab, you will generate the ATPG vectors and write out the Verilog and WGL patterns.

Invoke Modus

In the same directory **JumpStart_LABS/LAB1** invoke Modus by using the command:

```
User Prompt:/> modus
```

Once the Modus is invoked, you can see that the session imports the database which has been created in LAB1, you can continue where you left off in LAB1. You will see the following message:



```
INFO (TUI-300): Detected the presence of Modus database in present working directory. The 'workdir' root attribute
is set to '/home/shubham8/work/JumpStart_LABS/LAB1'. [end TUI_300]
```

Generating ATPG Vectors

ATPG tool will generate two types of test vectors.

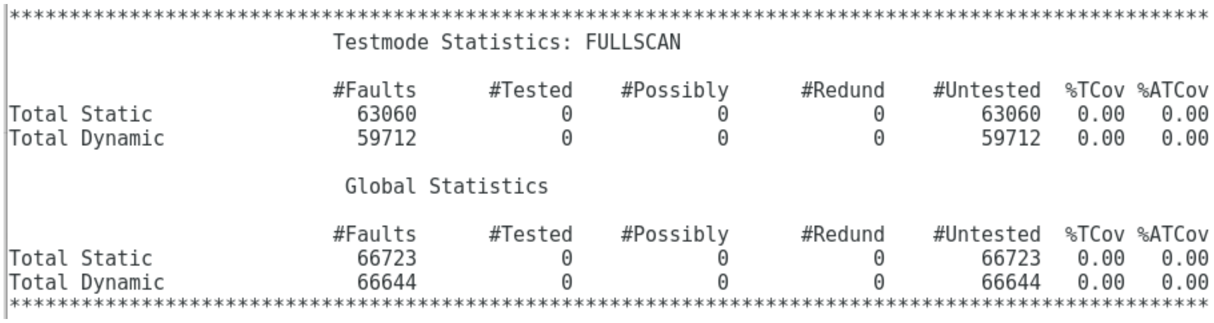
1. Scan chain tests targeting the faults along the scan path.

2. Logic tests to test the faults in the functional logic.

To generate the test vectors use the command:

```
create_logic_tests -testmode FULLSCAN -experiment logic
```

Once the above command is run, take a look at the log file and make sure there are no Severe Warnings. Look at the initial coverage section you should see that the tool started with a 0 coverage.



```
*******************************************************************************************************
                           Testmode Statistics: FULLSCAN

                           #Faults      #Tested    #Possibly      #Redund    #Untested  %TCov %ATCov
Total Static                 63060            0            0            0        63060   0.00   0.00
Total Dynamic                59712            0            0            0        59712   0.00   0.00

                            Global Statistics

                           #Faults      #Tested    #Possibly      #Redund    #Untested  %TCov %ATCov
Total Static                 66723            0            0            0        66723   0.00   0.00
Total Dynamic                66644            0            0            0        66644   0.00   0.00
*******************************************************************************************************
```

You can take a quick look at some other option with create_logic_tests –help for informational purpose. Some options provide instructions to the tool on how to handle specific circuit scenarios. Examples:

a. **-contentionremove**: This option controls the way ATPG handles the 3 state contentions. If you select hard, the tool will throw out any vectors that result hard contention but will keep patterns with soft contention.

b. Similarly, pay attention to options such as **maxpatterns**, **maxcputime**, and **maxelapsedtime**.

For an ATPG run it is important to know the final coverage and the number of vectors generated. Look for the information in the log file. Find the answers to following questions.

What is your resultant Fault Coverage for that Testmode?

How about your Global Coverage?

Why are they different?

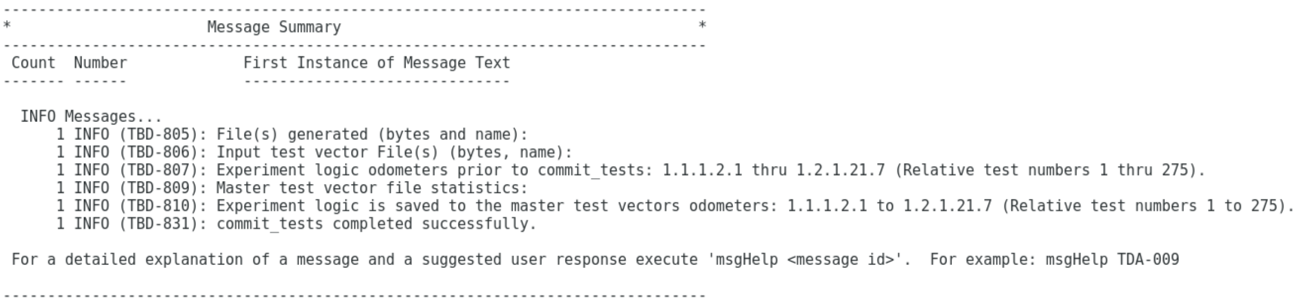
How many Test Sequences were generated?

Committing ATPG Vectors

This process will apply these vectors permanently against the Global Fault list and save these vectors. if you feel the vector set is good and meets the required coverage targets and pattern count, you can commit these tests. Once the tests are committed no other tests, if further generated, will try to go after the faults tested with these set of vectors. To commit test vectors use the command:

```
commit_tests –testmode FULLSCAN –inexperiment logic
```

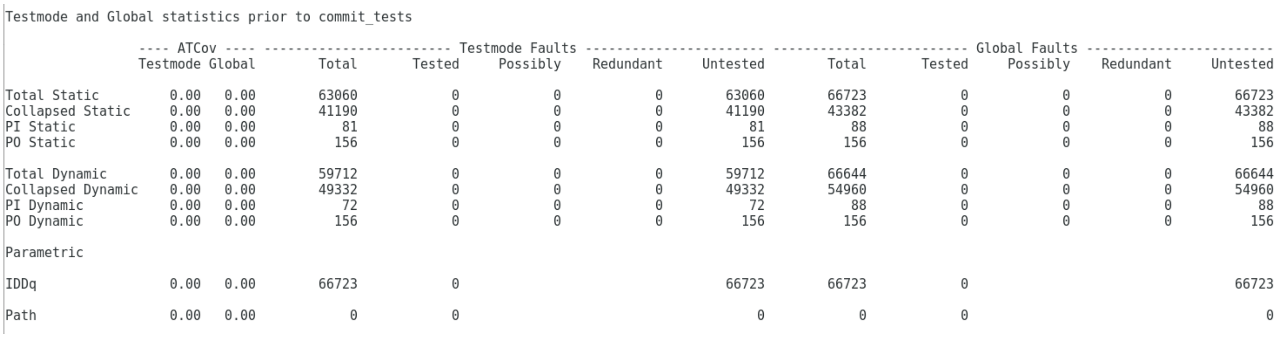
After the command has been run, the commit_tests summary is:



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

  INFO Messages...
      1 INFO (TBD-805): File(s) generated (bytes and name):
      1 INFO (TBD-806): Input test vector File(s) (bytes, name):
      1 INFO (TBD-807): Experiment logic odometers prior to commit_tests: 1.1.1.2.1 thru 1.2.1.21.7 (Relative test numbers 1 thru 275).
      1 INFO (TBD-809): Master test vector file statistics:
      1 INFO (TBD-810): Experiment logic is saved to the master test vectors odometers: 1.1.1.2.1 to 1.2.1.21.7 (Relative test numbers 1 to 275).
      1 INFO (TBD-831): commit_tests completed successfully.

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

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

Note: Make sure you get a TBD-831 message at the end.

Look at the initial coverage section in the log file, you should see that the tool started with a 0 coverage.



```
Testmode and Global statistics prior to commit_tests

                 ---- ATCov ---- ------------------------ Testmode Faults ----------------------- ------------------------- Global Faults ------------------------
                 Testmode Global        Total       Tested     Possibly    Redundant     Untested        Total       Tested     Possibly    Redundant     Untested

Total Static        0.00   0.00        63060            0            0            0        63060        66723            0            0            0        66723
Collapsed Static    0.00   0.00        41190            0            0            0        41190        43382            0            0            0        43382
PI Static           0.00   0.00           81            0            0            0           81           88            0            0            0           88
PO Static           0.00   0.00          156            0            0            0          156          156            0            0            0          156

Total Dynamic       0.00   0.00        59712            0            0            0        59712        66644            0            0            0        66644
Collapsed Dynamic   0.00   0.00        49332            0            0            0        49332        54960            0            0            0        54960
PI Dynamic          0.00   0.00           72            0            0            0           72           88            0            0            0           88
PO Dynamic          0.00   0.00          156            0            0            0          156          156            0            0            0          156

Parametric

IDDq                0.00   0.00        66723            0                                   66723        66723            0                                   66723

Path                0.00   0.00            0            0                                       0            0            0                                       0
```

Writing Vectors

This steps writes out the vectors that can be used for manufacturing tests and for CAD simulation and verification. Typically, the vector formats that are used for manufacturing tests are WGL, STIL, TDL. The format used for cad simulation and verification is Verilog. Users can write out the patterns which are committed or the patterns which are not committed. While writing the patterns specify the experiment name using the option **–inexperiment** to write the uncommitted vectors. In this lab case you have committed the vectors, hence you will not use –inexperiment option.

1. Write out WGL Patterns.

   Use the following command and write out the patterns in WGL format.

   ```
   write_vectors -testmode FULLSCAN -language wgl
   ```

   Pay attention to the top section which lists all of the timing parameters used to generate the WGL. We will see how to control these in the next section.

   What is your Total Cycle Count?

   The results will be in **JumpStart_LABS/LAB1/testresults/wgl**. Open the WGL file **WGL.FULLSCAN.scan.ex1.ts1.wgl** and take a look. You will see comments at the top of the file that describe all of the timing involved in writing out the vectors.

2. Write out Verilog Patterns.

   To write out the Verilog vectors use the command:

   ```
   write_vectors -testmode FULLSCAN -language verilog -testrange 1:100
   ```

   Note: This step is for generating patterns for simulation and verification. Use write_vectors with –language Verilog to write out the patterns in Verilog format. Use the option testrange to write out a specific set of patterns.

write_vectors offers two scan formats of patterns with Verilog. A serial format that loads the patterns serially on the design using the scan inputs is representative of the actual scan load of manufacturing tests. A parallel load is also available, which directly forces the scan values on the flops. This saves significant simulation time and is achieved by adding **-scanformat** parallel to the command. The above command writes out the Verilog vectors in serial scan format, which is the default.

Exiting the Modus Software

To close the Modus, use the command at the Modus prompt:

```
exit
```

End of Lab

Module 4: Debug Scan Chains with GUI and Tcl Interface

Lab 4-1 Debugging Broken Scan Chains with Modus GUI

Objective: To debug broken scan chains violations found in the verify_test_strucutres report using Modus circuit display.

In this lab, you will debug the design rule violations found in the verify_test_strucutres report using Modus circuit display.

Invoke Modus Software

1. Change the work directory to **JumpStart_LABS/LAB2** using the command:

```
user_prompt:/> cd JumpStart_LABS/LAB2
```

2. Invoke Modus tool by using the command:

```
user_prompt:/> modus
```

3. Source the **run_modus.tcl file**, present in the LAB2 directory by using the command:

```
source run_modus.tcl
```

Note: This is a command line file that executes build_model, build_testmode, report_test_structures and verify_test_structures. The same flow you did in LAB1. Wait for the script to finish. This will create your test model and view. The command **verify_test_structures** is also run towards the end of the script. You can review the log files of each command separately in the ./testresults/logs directory.

You should now review the outcome of verify_test_structures. By looking at the TSV-384 and several TSV-40X (for exampleTSV-401 TSV-402) messages, answer the following questions:

Are there any broken scan chains?

How many broken scan chains are there?

Open Modus GUI

Let’s debug the broken scan chains in the Interactive GUI. Again, the GUI won’t open until the execute another “set_db workdir .”.

1. To set the work directory use the command:

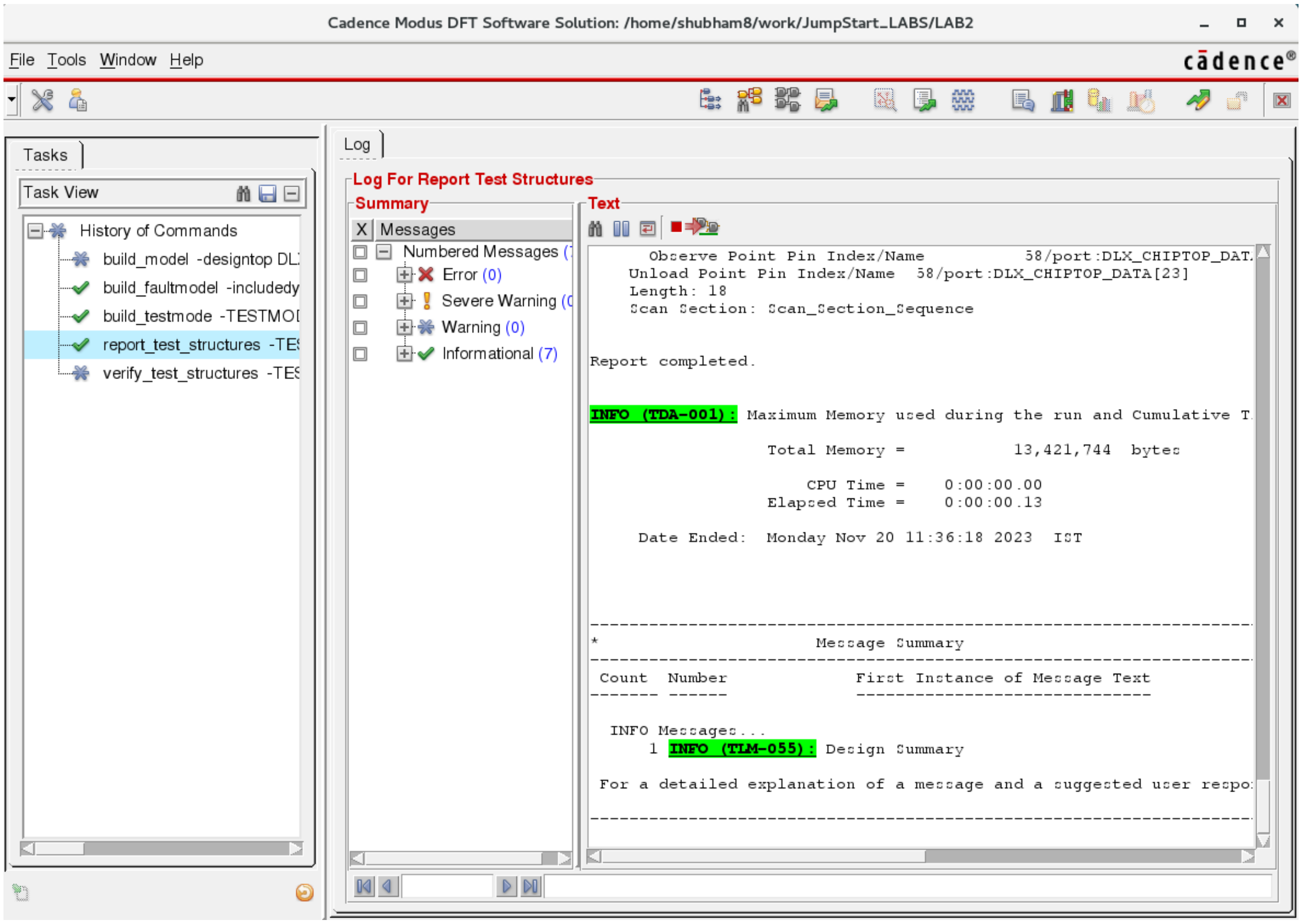
```
set_db workdir .
```

2. Open the Modus GUI by using the command:

```
gui_open
```

The GUI window should come up and the Tasks pane should show the commands that were run.

3. At the task view, click on the **report_test_structures** run and look at the log file and scan chain report.

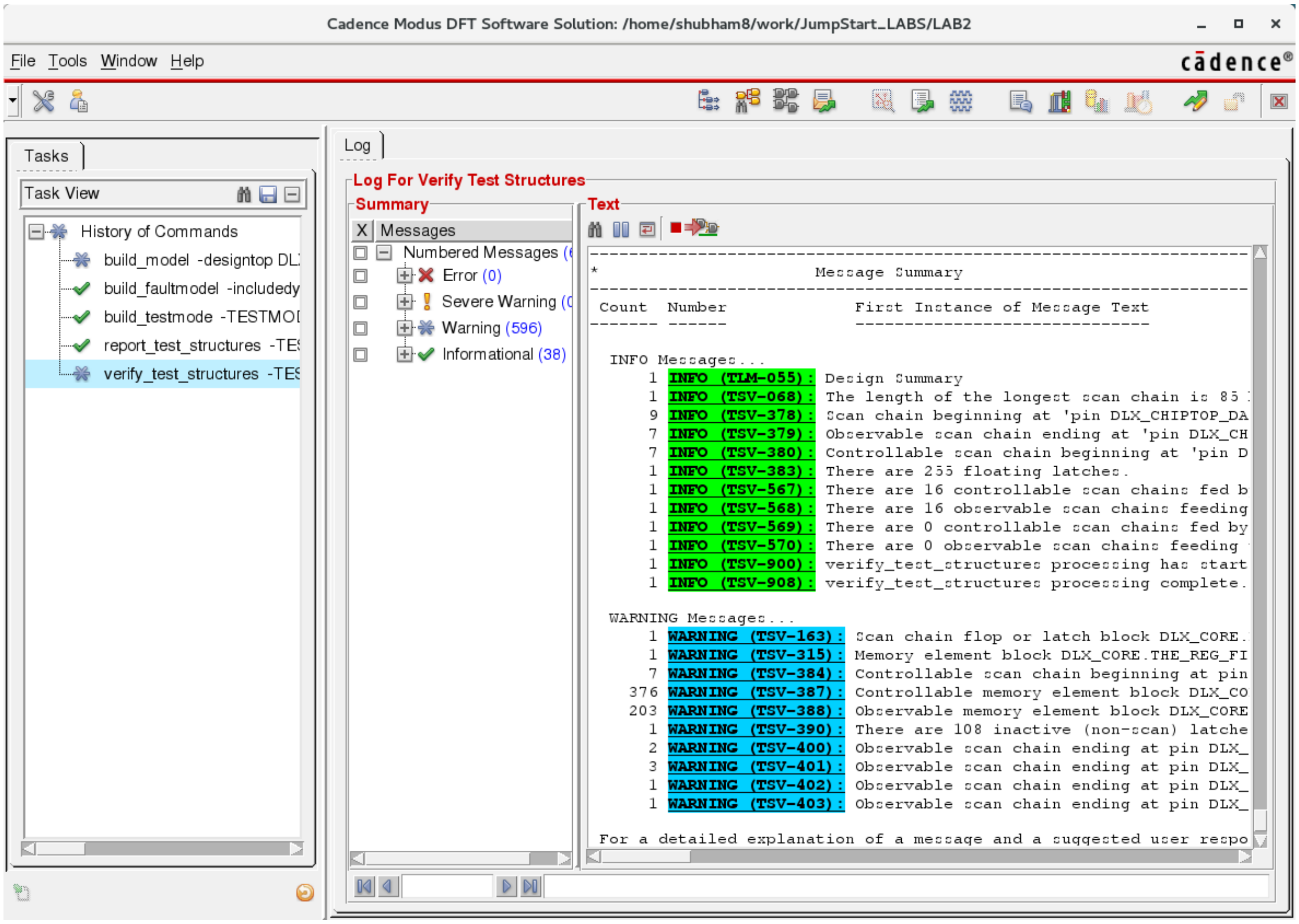

Cadence Modus DFT Software Solution: /home/shubham8/work/JumpStart_LABS/LAB2
File
Tools
Window
Help
cādence®
Tasks
Task View
Log
Log For Report Test Structures
Summary
Text
X
Messages

```
History of Commands
    build_model  -designtop DL
    build_faultmodel  -includedy
    build_testmode  -TESTMOD
    report_test_structures  -TES
    verify_test_structures  -TES
```

```
□ ⊟ Numbered Messages (
□     ⊞ Error (0)
□     ⊞ Severe Warning (
□     ⊞ Warning (0)
□     ⊞ Informational (7)
```

```
      Observe Point Pin Index/Name            58/port:DLX_CHIPTOP_DAT
    Unload Point Pin Index/Name   58/port:DLX_CHIPTOP_DATA[23]
    Length: 18
    Scan Section: Scan_Section_Sequence


Report completed.


INFO (TDA-001): Maximum Memory used during the run and Cumulative T

                  Total Memory =           13,421,744  bytes

                      CPU Time =    0:00:00.00
                  Elapsed Time =    0:00:00.13

     Date Ended:  Monday Nov 20 11:36:18 2023  IST




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

  INFO Messages...
      1 INFO (TLM-055): Design Summary

 For a detailed explanation of a message and a suggested user respo

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

4. Click on the **verify_test_structures** run and review the Error and Warning Messages.



Cadence Modus DFT Software Solution: /home/shubham8/work/JumpStart_LABS/LAB2

History of Commands
- build_model -designtop DL
- build_faultmodel -includedy
- build_testmode -TESTMOD
- report_test_structures -TE
- verify_test_structures -TE

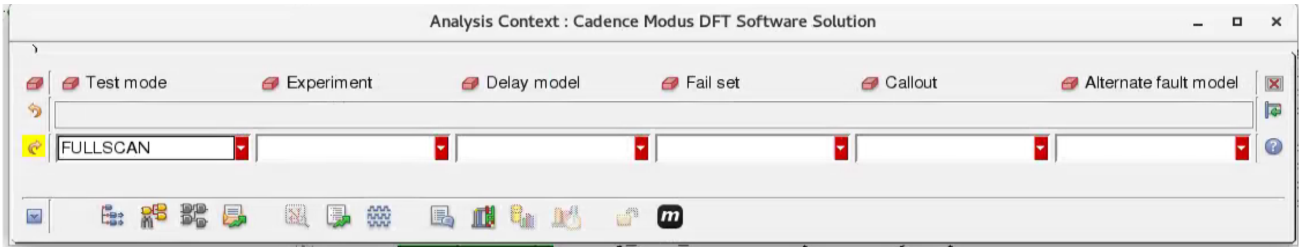
Messages
- Numbered Messages (
- Error (0)
- Severe Warning (
- Warning (596)
- Informational (38)

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

  INFO Messages...
      1 INFO (TLM-055): Design Summary
      1 INFO (TSV-068): The length of the longest scan chain is 85
      9 INFO (TSV-378): Scan chain beginning at 'pin DLX_CHIPTOP_DA
      7 INFO (TSV-379): Observable scan chain ending at 'pin DLX_CH
      7 INFO (TSV-380): Controllable scan chain beginning at 'pin D
      1 INFO (TSV-383): There are 255 floating latches.
      1 INFO (TSV-567): There are 16 controllable scan chains fed b
      1 INFO (TSV-568): There are 16 observable scan chains feeding
      1 INFO (TSV-569): There are 0 controllable scan chains fed by
      1 INFO (TSV-570): There are 0 observable scan chains feeding
      1 INFO (TSV-900): verify_test_structures processing has start
      1 INFO (TSV-908): verify_test_structures processing complete.

  WARNING Messages...
      1 WARNING (TSV-163): Scan chain flop or latch block DLX_CORE.
      1 WARNING (TSV-315): Memory element block DLX_CORE.THE_REG_FI
      7 WARNING (TSV-384): Controllable scan chain beginning at pin
    376 WARNING (TSV-387): Controllable memory element block DLX_CO
    203 WARNING (TSV-388): Observable memory element block DLX_CORE
      1 WARNING (TSV-390): There are 108 inactive (non-scan) latche
      2 WARNING (TSV-400): Observable scan chain ending at pin DLX_
      3 WARNING (TSV-401): Observable scan chain ending at pin DLX_
      1 WARNING (TSV-402): Observable scan chain ending at pin DLX_
      1 WARNING (TSV-403): Observable scan chain ending at pin DLX_

 For a detailed explanation of a message and a suggested user respo
```

Setting Analysis Context

Before going to debug the broken scan chain, we need to set the analysis context first and then look for the respective broken scan chains.

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

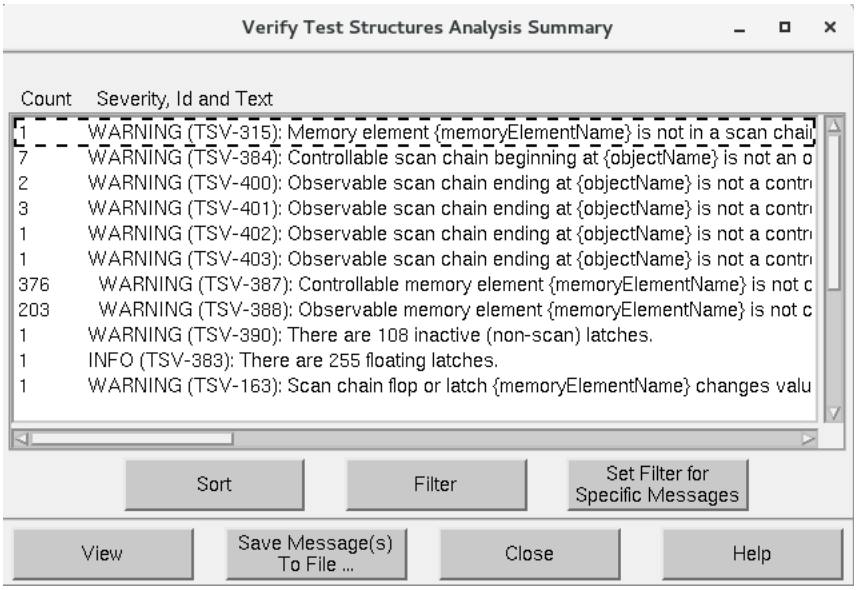


Analysis Context : Cadence Modus DFT Software Solution

Test mode: FULLSCAN

2. Choose the analysis mode you want to analyze, in our case we only have 1 testmode named **FULLSCAN**. Select the testmode from the drop down menu and click on . This should set the analysis context.

3. Once the context is set click the **view messages icon** on the Analysis Context window. This will open the new messages window which contains the summary of TSV messages.


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

Count	Severity, Id and Text
1	WARNING (TSV-315): Memory element {memoryElementName} is not in a scan chai
7	WARNING (TSV-384): Controllable scan chain beginning at {objectName} is not an o
2	WARNING (TSV-400): Observable scan chain ending at {objectName} is not a contr
3	WARNING (TSV-401): Observable scan chain ending at {objectName} is not a contr
1	WARNING (TSV-402): Observable scan chain ending at {objectName} is not a contr
1	WARNING (TSV-403): Observable scan chain ending at {objectName} is not a contr
376	WARNING (TSV-387): Controllable memory element {memoryElementName} is not c
203	WARNING (TSV-388): Observable memory element {memoryElementName} is not c
1	WARNING (TSV-390): There are 108 inactive (non-scan) latches.
1	INFO (TSV-383): There are 255 floating latches.
1	WARNING (TSV-163): Scan chain flop or latch {memoryElementName} changes valu

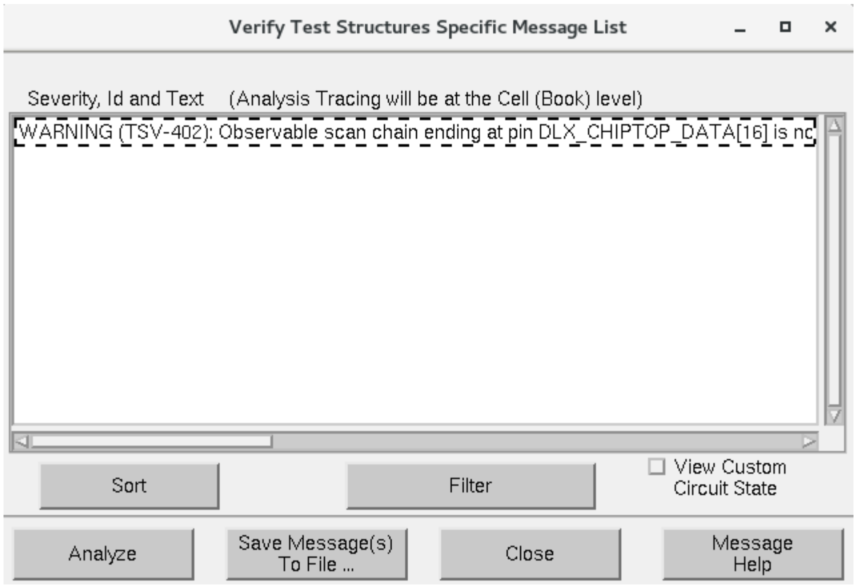
4. Look for **TSV-384** and several **TSV-40X (for exampleTSV-401 TSV-402)** messages in the Verify Test Structures Analysis Summary window, these messages indicate the broken chains (number of observable scan chains that are not controllable and the controllable scan chains which are not observable).

   Note: Generally, it is easier to debug the broken observable chains in comparison to broken controllable chains as there is usually less choices in logic while tracing backward in comparison to tracing forward.

Going forward in this lab we will debug the Observable scan chains which are not controllable (TSV-40X).

5. Select the TSV-402 messages and click **View**. This should bring up another window with 1 individual instances of message.

```
(TSV-402): Observable scan chain ending at pin DLX_CHIPTOP_DATA[16] is not a
   controllable scan chain. The last observation flop in the scan chain is
   block DLX_CORE.PC_REG.STORED_VALUE_reg_0.__iNsT2.dff_primitive. Tracing
   stopped because data input is at constant value.
```

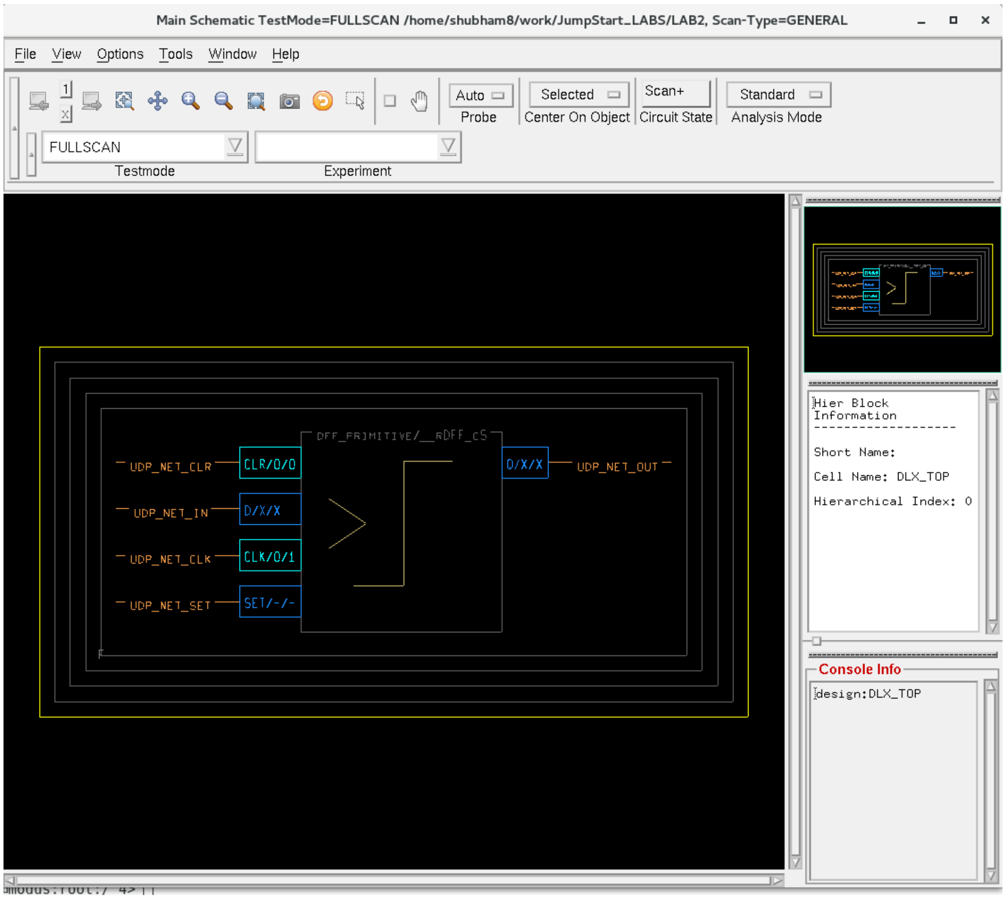


Verify Test Structures Specific Message List
Sort
Filter
View Custom Circuit State
Analyze
Save Message(s) To File ...
Close
Message Help

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

```
WARNING (TSV-402): Observable scan chain ending at pin DLX_CHIPTOP_DATA[16] is nc
```

6. Select the message (DLX_CHIPTOP_DATA[16]) and click Analyze. This will bring the schematic window on the top and will navigate to the last good cell it encountered in the analysis.



Main Schematic TestMode=FULLSCAN /home/shubham8/work/JumpStart_LABS/LAB2, Scan-Type=GENERAL

Testmode: FULLSCAN

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

Short Name:

Cell Name: DLX_TOP

Hierarchical Index: 0
```

```
design:DLX_TOP
```

```
modus:root:/ 4> ||
```

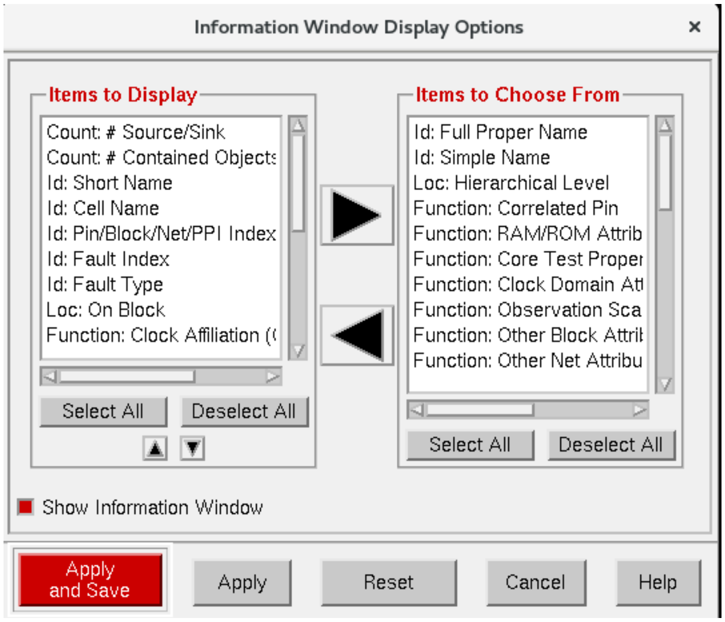
Customizing Schematic Window

It is recommended to customize some of the default settings of the Schematic window. These options change the information that you see in the different context windows.

1. **Block Information Window**

   First let's customize what we see in the **Block Information** window we see on the right-hand side.

   a. Click Pulldown menu **Options -> Information Window...** The left side of the window has the options currently displayed.



Information Window Display Options

Items to Display

- Count: # Source/Sink
- Count: # Contained Objects
- Id: Short Name
- Id: Cell Name
- Id: Pin/Block/Net/PPI Index
- Id: Fault Index
- Id: Fault Type
- Loc: On Block
- Function: Clock Affiliation (

Items to Choose From

- Id: Full Proper Name
- Id: Simple Name
- Loc: Hierarchical Level
- Function: Correlated Pin
- Function: RAM/ROM Attrib
- Function: Core Test Proper
- Function: Clock Domain Att
- Function: Observation Sca
- Function: Other Block Attrit
- Function: Other Net Attribu

☒ Show Information Window

   b. The right side has the items to add. Recommended Items to Add if not already added

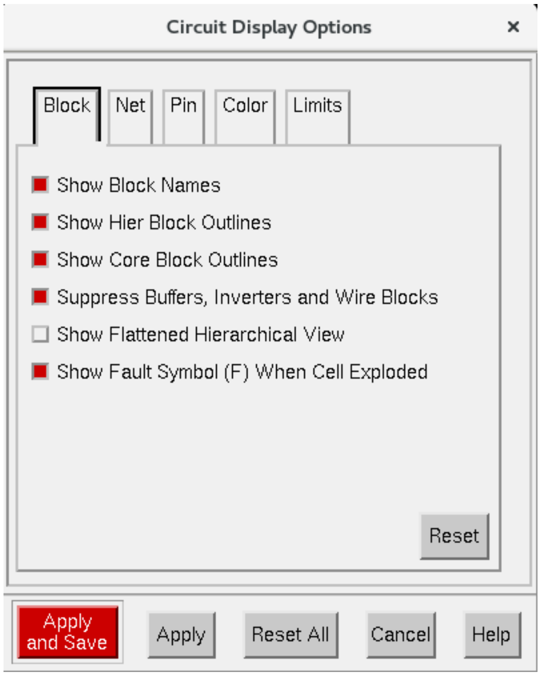
```
Function: Scan Path Data : It shows whether pins are a part of the scan chain
Function: Flop/Latch Scan Data :  It Shows the flop scan bit position
Function: Clock Affiliation :    Shows Clock information on pins
Values: Possible Value Set  :  Shows the values a pin can attain
```

   c. Click "**Apply** and **Save**" to apply.

2. **Circuit Display Information Window**

   Now, let's customize the way the nets/pins are displayed.

   a. Click pull-down menu **Options -> Circuit Display…**



☑ Show Block Names
☑ Show Hier Block Outlines
☑ Show Core Block Outlines
☑ Suppress Buffers, Inverters and Wire Blocks
☐ Show Flattened Hierarchical View
☑ Show Fault Symbol (F) When Cell Exploded

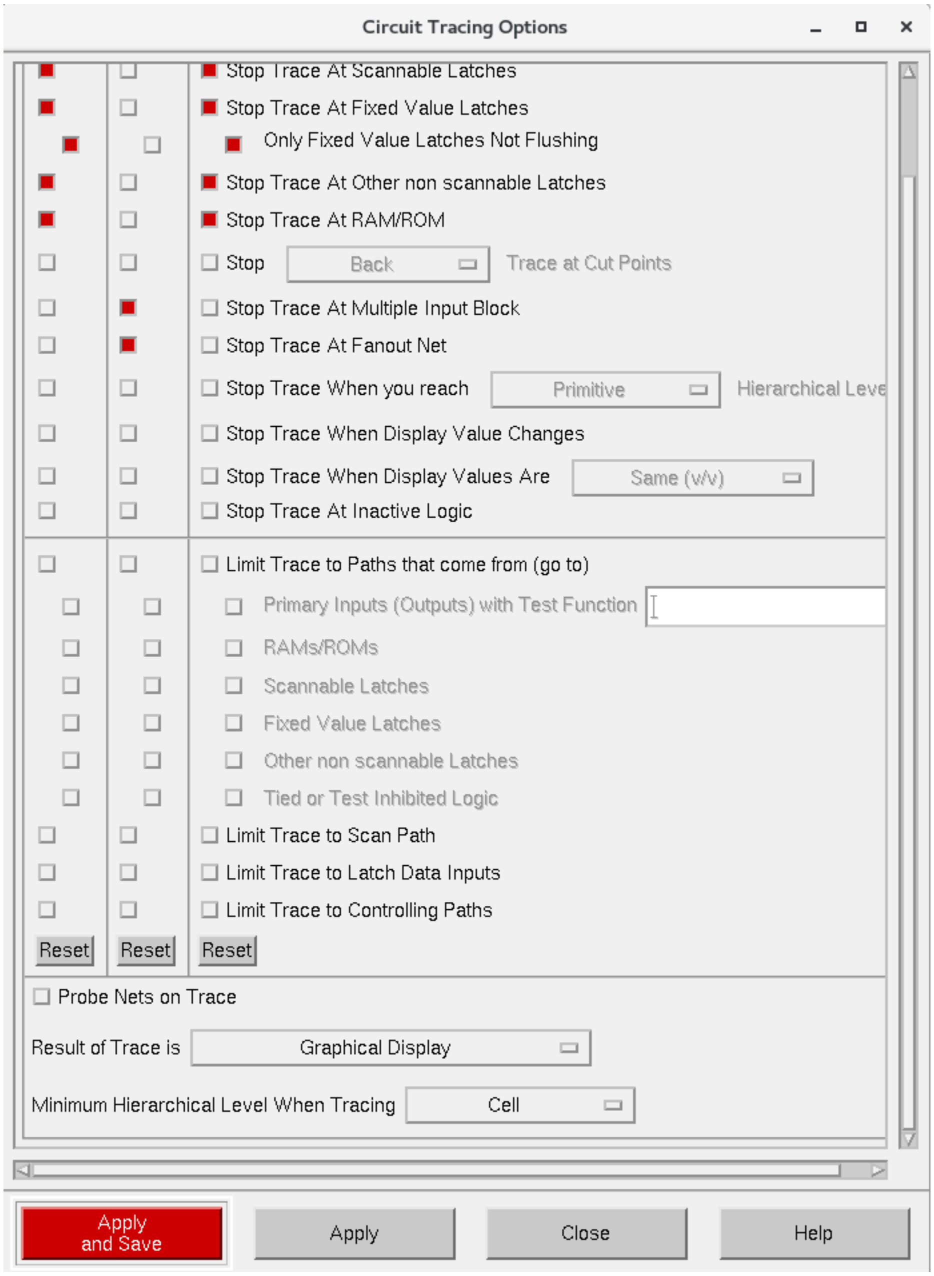
b. Click on Pin Tab and select on Show All Pins: This will show all the pins on a cell instead of collapsing them.

c. Click “**Apply** and **Save**” to apply.

3. **Circuit Tracing Window**

Now, let’s customize some of the circuit tracing options.

a. From the Pulldown menu **Options -> Circuit Tracing…** Scroll to bottom of Circuit Tracing window.



Circuit Tracing Options

■ □ ■ Stop Trace At Scannable Latches
■ □ ■ Stop Trace At Fixed Value Latches
■ □ ■ Only Fixed Value Latches Not Flushing
■ □ ■ 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 Leve
□ □ □ Stop Trace When Display Value Changes
□ □ □ Stop Trace When Display Values Are Same (v/v)
□ □ □ Stop Trace At Inactive Logic

□ □ □ 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
□ □ □ Limit Trace to Latch Data Inputs
□ □ □ Limit Trace to Controlling Paths

□ Probe Nets on Trace

Result of Trace is: Graphical Display

Minimum Hierarchical Level When Tracing: Cell

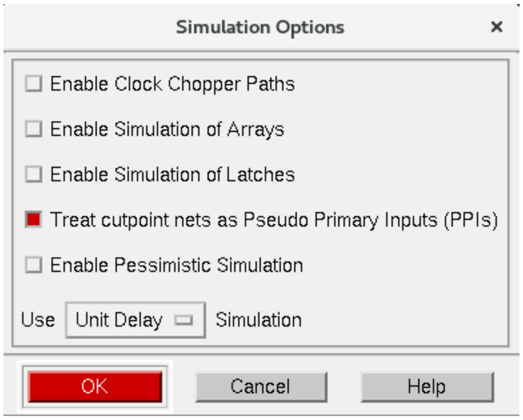
b. Set **Minimum Hierarchical Level When Tracing** to "**Cell**" if not selected.

c. Click "**Apply** and **Save**" and then Close.

4. **Simulate Options Window**

One final setting to change from the Simulate Options Window

a. From the pull-down menu **Options -> Simulate…**



Simulation Options

☐ Enable Clock Chopper Paths

☐ Enable Simulation of Arrays

☐ Enable Simulation of Latches

☒ Treat cutpoint nets as Pseudo Primary Inputs (PPIs)

☐ Enable Pessimistic Simulation

Use Unit Delay Simulation

b. **Enable Simulation of Latches**: This will allow the internal simulator to propagate values through sequential elements during debug.

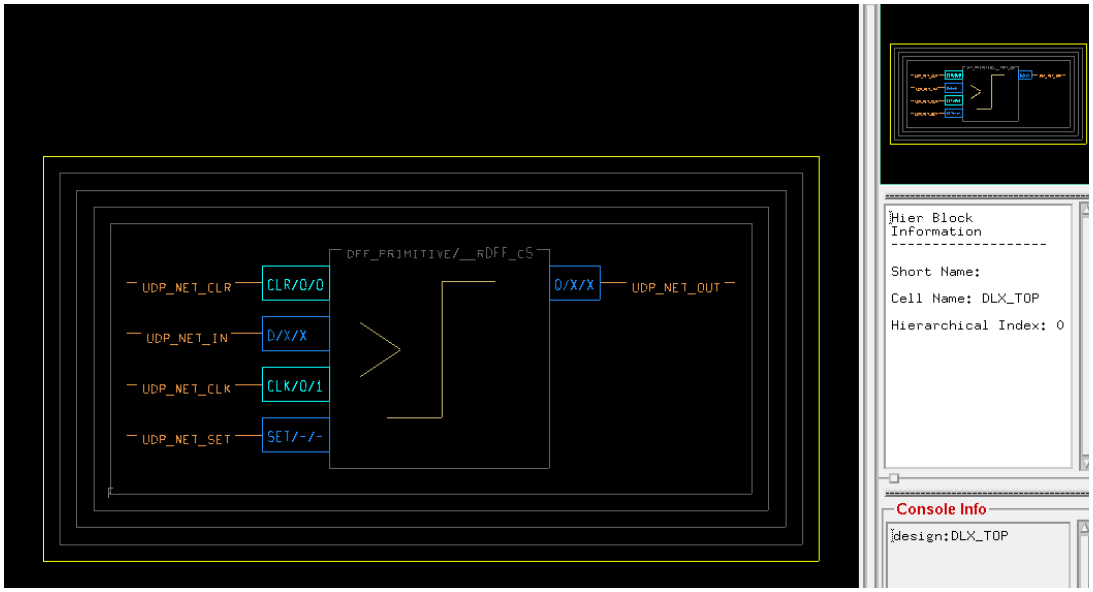
c. Select the option and click **OK**.

Interact with Schematic Display Window

In this section, we will understand some features of the Schematic Circuit Display.

1. **Schematic Hierarchies**

   You will see several white rectangles around the cell. Each of these represents a level of hierarchy. Mouse over each and see what changes in the Block Information window.



```
Hier Block
Information
--------------------
Short Name:
Cell Name: DLX_TOP
Hierarchical Index: 0
```

Console Info

```
design:DLX_TOP
```

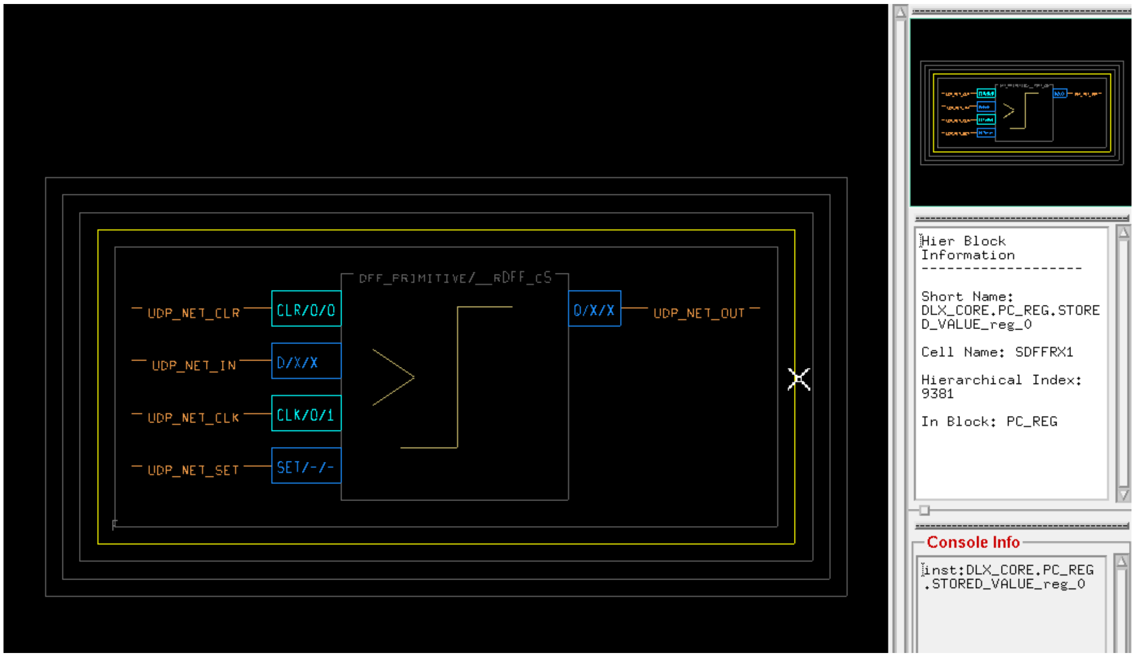
What is the Cell Name of the innermost white rectangle?

How about the next one further out?

2. **Imploding and Exploding a Hierarchy**

   To implode and explode a hierarchy, use the below steps:

   a. **Left**-click on the white rectangle representing the **SDFFRX1** cell.



Hier Block
Information

Short Name:
DLX_CORE.PC_REG.STORE
D_VALUE_reg_0

Cell Name: SDFFRX1

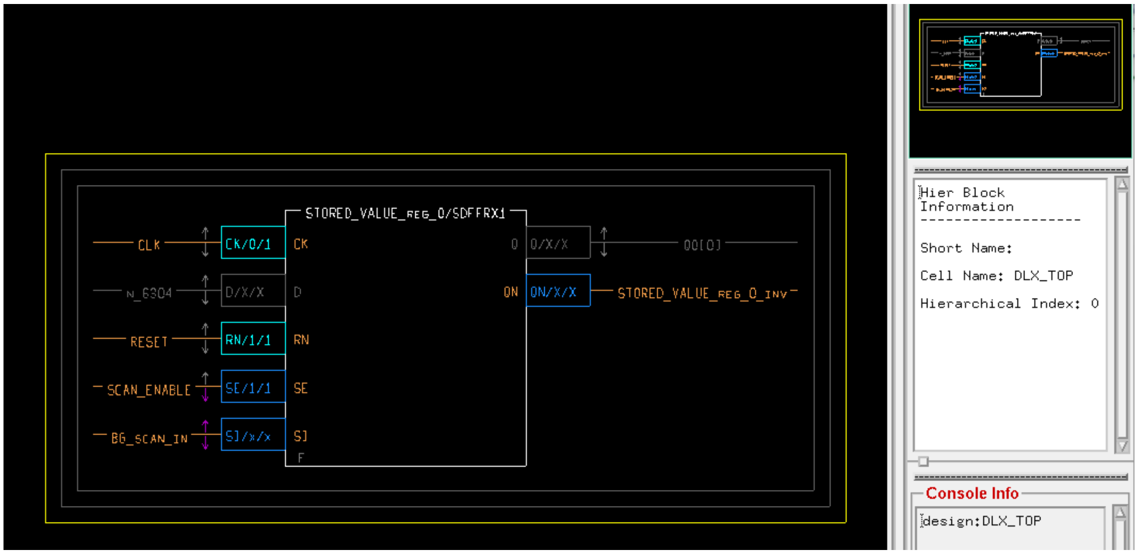
Hierarchical Index:
9381

In Block: PC_REG

Console Info

inst:DLX_CORE.PC_REG
.STORED_VALUE_reg_0

   b. **Right**-click on the rectangle and select **Implode**: This will collapse that level of hierarchy selected and you should now see the library cell SDFFRX1 in its entirety.



Hier Block
Information

Short Name:

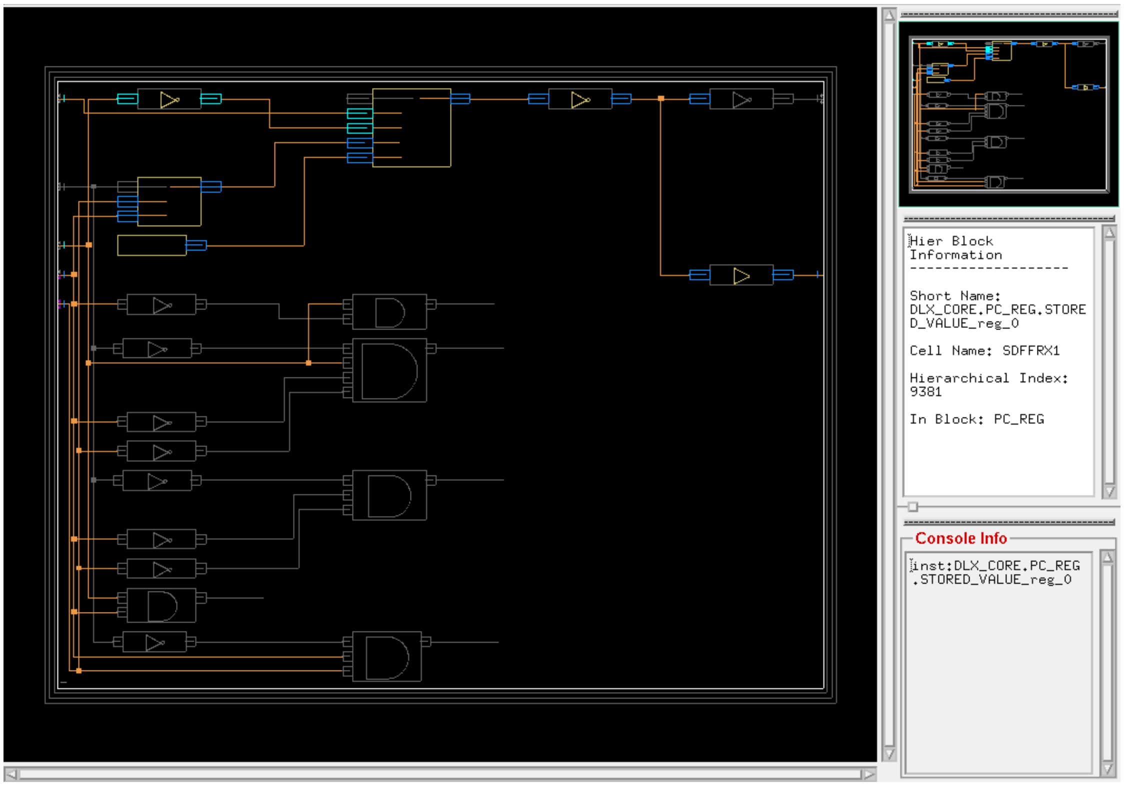
Cell Name: DLX_TOP

Hierarchical Index: 0

Console Info

design:DLX_TOP

c. Select the **SDFFRX1 cell -> right**-click **->** Click **Explode**: Now you should see the whole scan flop cell.



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

Short Name:
DLX_CORE.PC_REG.STORE
D_VALUE_reg_0

Cell Name: SDFFRX1

Hierarchical Index:
9381

In Block: PC_REG
```

Console Info

```
inst:DLX_CORE.PC_REG
.STORED_VALUE_reg_0
```

d. Getting back to previous circuit state view by the view preview/next display buttons to traverse across your displays.

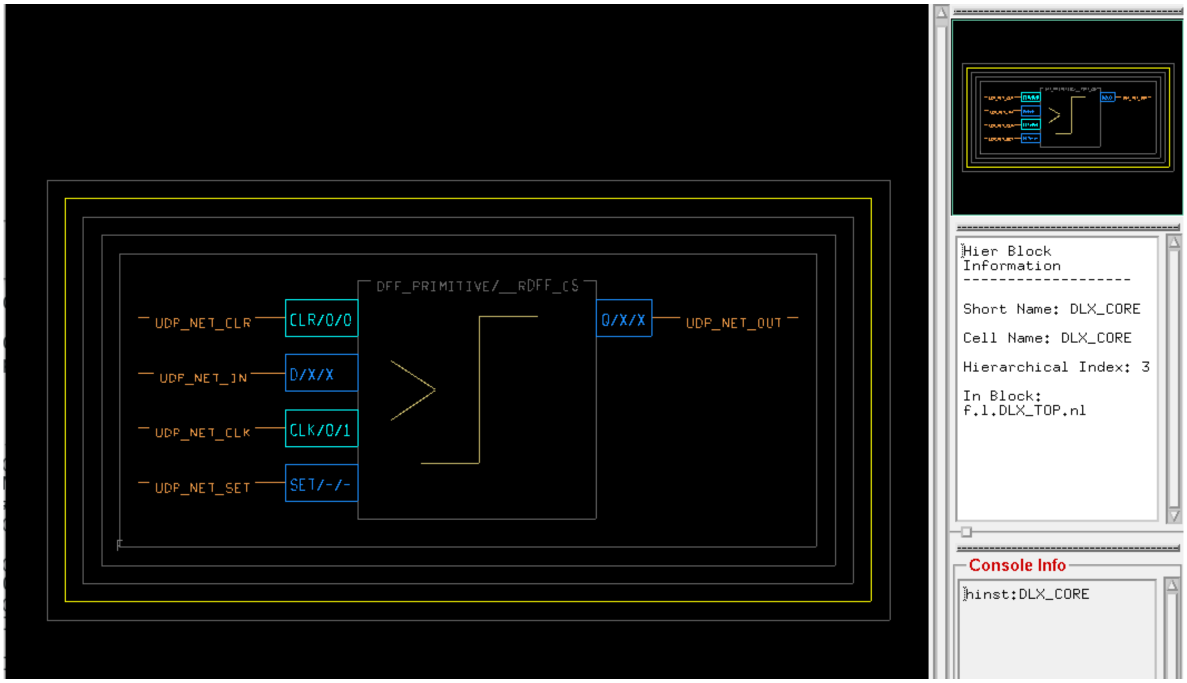
3. **Zoom in and Zoom out**

   If you have a scroll wheel on your mouse, hold the control key down and scroll with your wheel to zoom in and out. If nothing happens, click on **Options -> Controls…** and select the option there.

Debugging Broken Scan Chains

Let's debug the scan chains now. Each of the 7 broken chains has a different cause of why the scan chain is broken (which you can also see in different **TSV-40X** warnings with no. of broken scan chain).

1. As stated earlier, the schematic comes up to the last good register. The current view you are seeing is the Modus primitive of the library cell.



Hier Block Information

Short Name: DLX_CORE

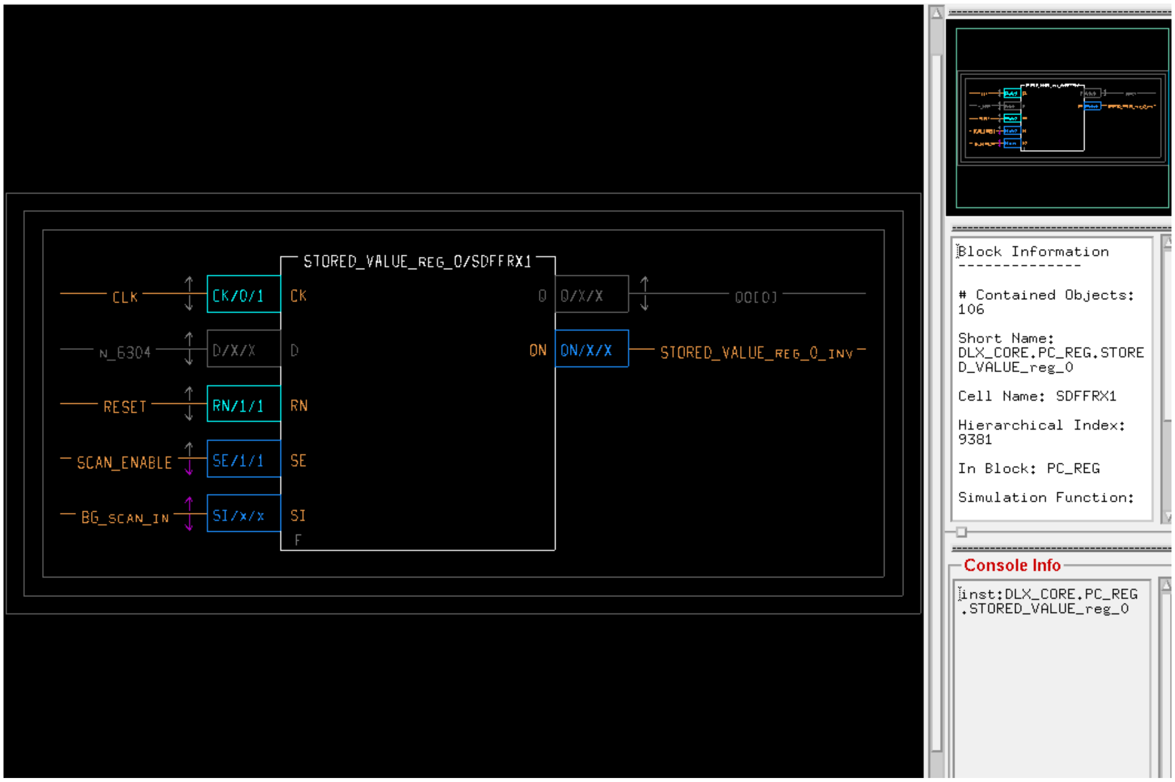
Cell Name: DLX_CORE

Hierarchical Index: 3

In Block:
f.1.DLX_TOP.nl

hinst:DLX_CORE

2. Select the **SDFFRX1 cell** hierarchy and click **Implode**.



Block Information

Contained Objects:
106

Short Name:
DLX_CORE.PC_REG.STORE
D_VALUE_reg_0

Cell Name: SDFFRX1

Hierarchical Index:
9381

In Block: PC_REG

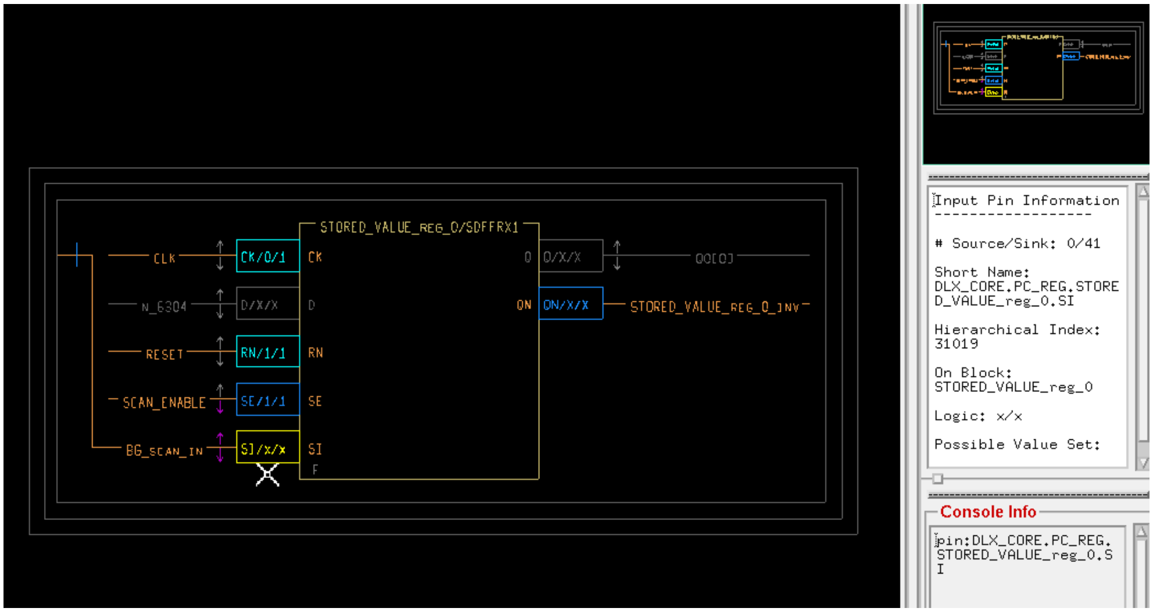
Simulation Function:

inst:DLX_CORE.PC_REG
.STORED_VALUE_reg_0

3. Pay attention to the Information Window and values in there as you mouse over different pins.

4. Look at the values specified on the pins of the cell and compare them to what was discussed in the lecture. In this case for the broken chain **DLX_CHIPTOP_DATA[16]** we can see that Scan-in pin is at lower “x” state, means this is pin is unconnected.

5. **Left**-click on a pin and type **b** to trace backward and **f** to trace forward. Here we will select the Scan-in pin and trace backward.



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

# Source/Sink: 0/41

Short Name:
DLX_CORE.PC_REG.STORE
D_VALUE_reg_0.SI

Hierarchical Index:
31019

On Block:
STORED_VALUE_reg_0

Logic: x/x

Possible Value Set:
```

```
pin:DLX_CORE.PC_REG.
STORED_VALUE_reg_0.S
I
```

Note: Noticed that the Scan-in pin is not connected and not able to trace back further.

For the rest of this lab identify why each chain is broken.

Why is Scan Out DLX_CHIPTOP_DATA[17] chain broken?

Why is Scan Out DLX_CHIPTOP_DATA[18] chain broken?

Why is Scan Out DLX_CHIPTOP_DATA[19] chain broken?

Why is Scan Out DLX_CHIPTOP_DATA[20] chain broken?

Why is Scan Out DLX_CHIPTOP_DATA[21] chain broken?

Why is Scan Out DLX_CHIPTOP_DATA[23] chain broken?

Exiting Modus

Close the GUI once the activity is complete by using the command:

```
exit
```

End of Lab

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

Objective: To debug the broken scan chains with the help of command line interface.

This lab helps users to learn the debugging techniques using the command line interface.

Command Line Utilities to Use During Debugging

In this section, we will discuss the important commands to use while debugging the broken scan chains using command line interface.

1. **set_context**

   Context refers to a circuit testmode/experiment configuration. This requires to be set before any analysis is performed. As discussed in **Lab 4-1** context can be set using the GUI or the command line.

   ```
   set_context –testmode <testmode_name> -experiment <experiment_name>
   ```

2. **simulate_state**

   This utility does a simulation of a specific simulation state and allows the users to simulate a state before analyzing the circuit. This reports the values that are present on the pins of a circuit in a specific state such as scan, test_constraint, test_inhibit, etc.

   ```
   simulate_state <state_name>
   ```

3. **report_chain**

   Similar to report_test_structures, users can also report the information on scan chains using report_chains. In order to list all the scan chains, use the following command.

   ```
   report_chains -summary
   ```

   Note: The chains reported by report_chains either have a observe chain ID or a control chain ID or both. You can report the details of a specific chain using -observechain, for example:

   ```
   report_chains –observechain <chain_id>
   ```

4. **report_instances**

   report_instances helps you fetch the structural and simulation information of a specific instance. The values on the pins of an instance which are set by simulate value can be reported using report_instances.

   ```
   report_instances <instance_name>
   ```

5. **report_testfunctions**

   Users can report the test functions grouped with a specific type, e.g., scan clocks, capture clocks, scan and capture clocks, scan constraints, test constraints, scan inputs and outputs.

   ```
   report_testfunctions -name <type>
   ```

6. **trace_circuit**

   Similar to tracing in GUI, users can do circuit tracing on the command line, trace circuit can help you find the active fanin or fanout cone in a specific state. Use the options –forward or –backward to perform the tracing. Users can also use the –show_gui option to view the traced circuit elements in the GUI schematic viewer.

   ```
   trace_circuit -backward <pin_name>
   ```

Invoke Modus and Set Context

1. If you are not in the directory JumpStart_LABS/LAB2, then change the work directory to LAB2 by using the command:

   ```
   cd JumpStart_LABS/LAB2
   ```

2. Invoke Modus tool by using the command:

   ```
   modus
   ```

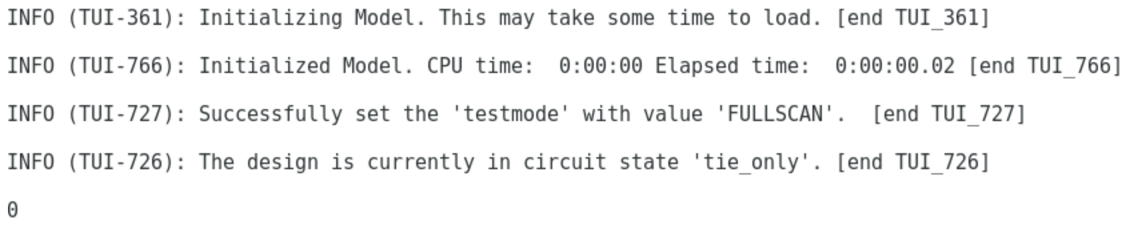
   Note: Since the database already exists it will be automatically loaded into the session. You can fetch the list of existing testmodes using the get_db utility.

3. Get the testmode by using the command:

   ```
   get_db testmodes
   ```

4. Set the analysis context by using the command:

   ```
   set_context -testmode FULLSCAN
   ```



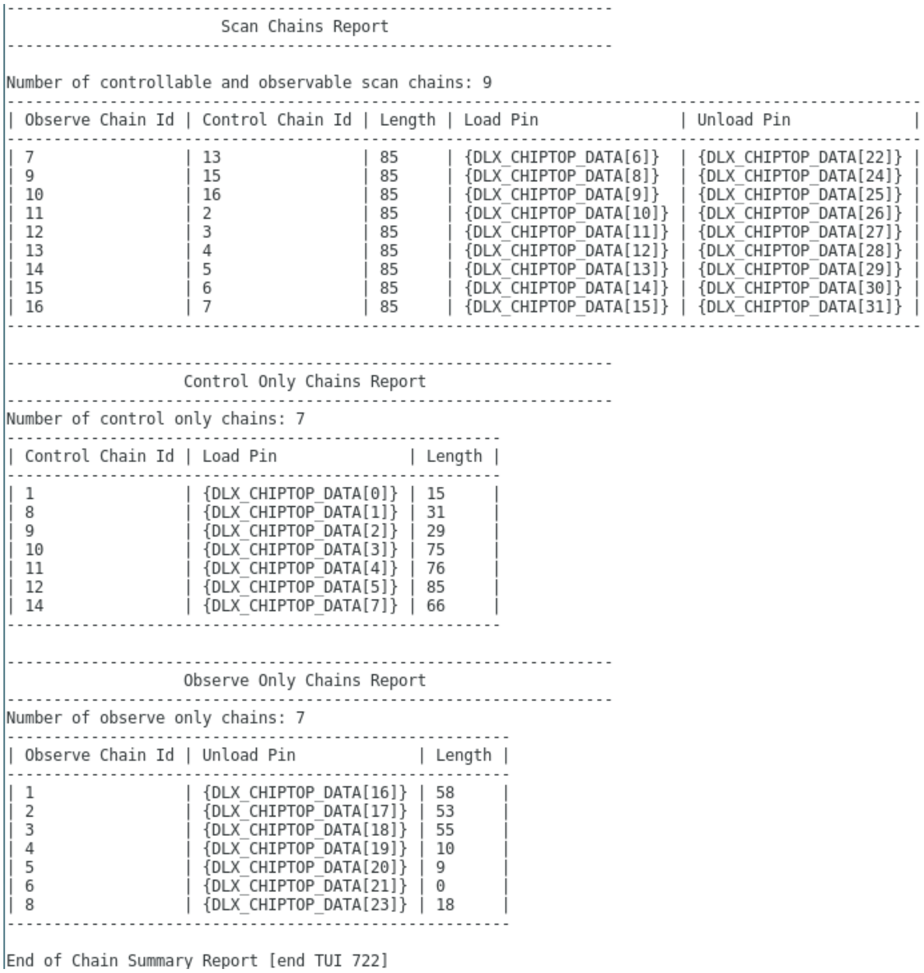
```
INFO (TUI-361): Initializing Model. This may take some time to load. [end TUI_361]

INFO (TUI-766): Initialized Model. CPU time:  0:00:00 Elapsed time:  0:00:00.02 [end TUI_766]

INFO (TUI-727): Successfully set the 'testmode' with value 'FULLSCAN'.  [end TUI_727]

INFO (TUI-726): The design is currently in circuit state 'tie_only'. [end TUI_726]

0
```

Finding a Broken Scan Chain

Let’s use the **report_chains** utility to fetch the information on the broken scan chains. This shall provide you the information on good chains and broken chains. To report the summary of the scan chains, use the command:

```
report_chains –summary
```

Output of the command will look like :



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

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

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

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

End of Chain Summary Report [end TUI 722]
```

The above is a summary of all the scan chains that Modus finds. It is broken down into three sections.

1. The first section shows the chains that are complete from SI to SO in the scan mode.

2. The next two sections are chains that are broken.

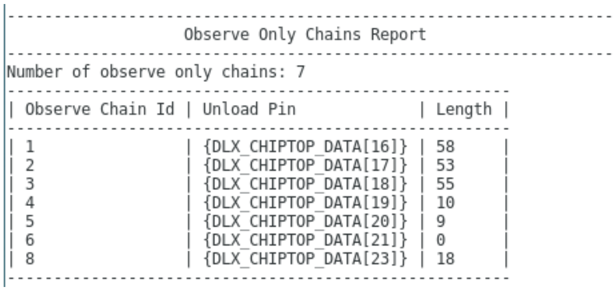
    a. The Control Only chains were traced from SI forward until Modus could not go further.

    b. The Observe Only chains were traced from SO backwards until Modus could not go further.

Note: It is easier to trace backward on broken chains then forward. So, let’s look at the first broken Observe Chain.

Debugging Observe Scan Chain 1

Start with the first one based on the SO pin **DLX_CHIPTOP_DATA[16]**. Look for the observe chain ID of this chain it is **1**.

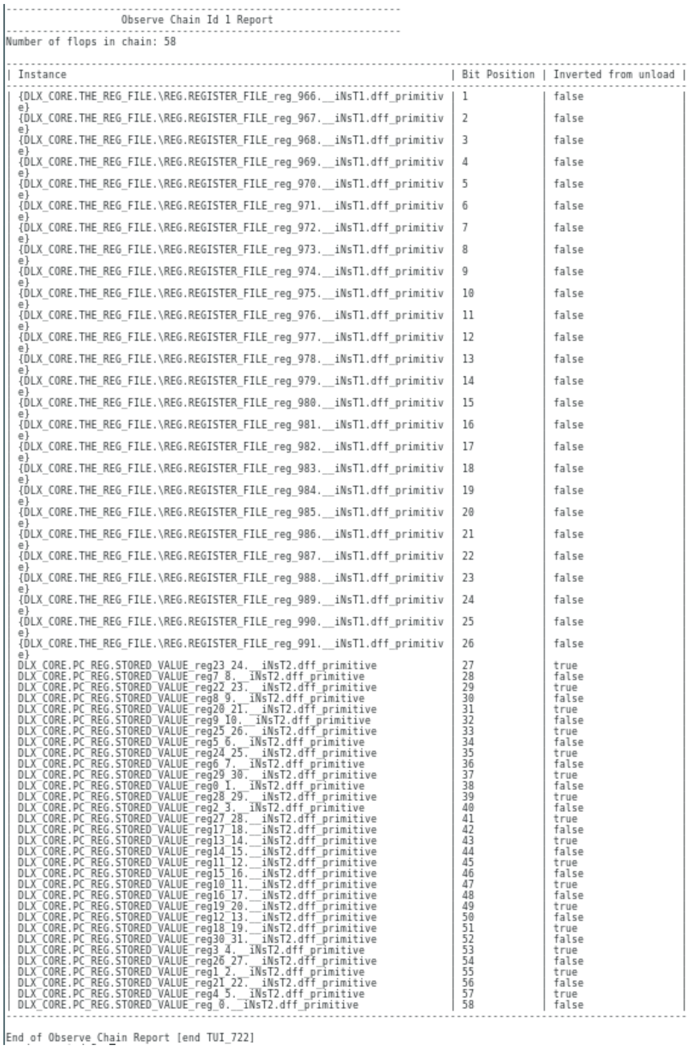


```
----------------------------------------------------------------
                  Observe Only Chains Report
----------------------------------------------------------------
Number of observe only chains: 7
-----------------------------------------------------
| Observe Chain Id | Unload Pin   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

1. Report the Observe chain with ID 1 by using the command:

   ```
   report_chains -observechain 1
   ```

The output of the observe chain summary is:



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

------------------------------------------------------------------------------------------------------------
| Instance                                                         | Bit Position | Inverted from unload |
------------------------------------------------------------------------------------------------------------
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_966.__iNsT1.dff_primitiv | 1            | false                |
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_967.__iNsT1.dff_primitiv | 2            | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_968.__iNsT1.dff_primitiv | 3            | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_969.__iNsT1.dff_primitiv | 4            | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_970.__iNsT1.dff_primitiv | 5            | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_971.__iNsT1.dff_primitiv | 6            | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_972.__iNsT1.dff_primitiv | 7            | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_973.__iNsT1.dff_primitiv | 8            | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_974.__iNsT1.dff_primitiv | 9            | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_975.__iNsT1.dff_primitiv | 10           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_976.__iNsT1.dff_primitiv | 11           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_977.__iNsT1.dff_primitiv | 12           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_978.__iNsT1.dff_primitiv | 13           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_979.__iNsT1.dff_primitiv | 14           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_980.__iNsT1.dff_primitiv | 15           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_981.__iNsT1.dff_primitiv | 16           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_982.__iNsT1.dff_primitiv | 17           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_983.__iNsT1.dff_primitiv | 18           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_984.__iNsT1.dff_primitiv | 19           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_985.__iNsT1.dff_primitiv | 20           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_986.__iNsT1.dff_primitiv | 21           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_987.__iNsT1.dff_primitiv | 22           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_988.__iNsT1.dff_primitiv | 23           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_989.__iNsT1.dff_primitiv | 24           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_990.__iNsT1.dff_primitiv | 25           | false
  e}
  {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_991.__iNsT1.dff_primitiv | 26           | false
  e}
  DLX_CORE.PC_REG.STORED_VALUE_reg23_24.__iNsT2.dff_primitive            | 27           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg7_8.__iNsT2.dff_primitive              | 28           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg22_23.__iNsT2.dff_primitive            | 29           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg8_9.__iNsT2.dff_primitive              | 30           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg20_21.__iNsT2.dff_primitive            | 31           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg9_10.__iNsT2.dff_primitive             | 32           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg25_26.__iNsT2.dff_primitive            | 33           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg5_6.__iNsT2.dff_primitive              | 34           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg24_25.__iNsT2.dff_primitive            | 35           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg6_7.__iNsT2.dff_primitive              | 36           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg29_30.__iNsT2.dff_primitive            | 37           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg0_1.__iNsT2.dff_primitive              | 38           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg28_29.__iNsT2.dff_primitive            | 39           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg2_3.__iNsT2.dff_primitive              | 40           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg27_28.__iNsT2.dff_primitive            | 41           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg17_18.__iNsT2.dff_primitive            | 42           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg13_14.__iNsT2.dff_primitive            | 43           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg14_15.__iNsT2.dff_primitive            | 44           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg11_12.__iNsT2.dff_primitive            | 45           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg15_16.__iNsT2.dff_primitive            | 46           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg10_11.__iNsT2.dff_primitive            | 47           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg16_17.__iNsT2.dff_primitive            | 48           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg19_20.__iNsT2.dff_primitive            | 49           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg12_13.__iNsT2.dff_primitive            | 50           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg18_19.__iNsT2.dff_primitive            | 51           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg30_31.__iNsT2.dff_primitive            | 52           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg3_4.__iNsT2.dff_primitive              | 53           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg26_27.__iNsT2.dff_primitive            | 54           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg1_2.__iNsT2.dff_primitive              | 55           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg21_22.__iNsT2.dff_primitive            | 56           | false
  DLX_CORE.PC_REG.STORED_VALUE_reg4_5.__iNsT2.dff_primitive              | 57           | true
  DLX_CORE.PC_REG.STORED_VALUE_reg_0.__iNsT2.dff_primitive               | 58           | false                |
------------------------------------------------------------------------------------------------------------

End of Observe_Chain Report [end TUI_722]
```

The above listing generated from report_chains showing all of the connected observe flops that Modus recognizes. The Observe Flop in Position 1 is the closest flop to the **SO** pin. The analysis stopped at Observe Flop 58. This is the last Flop that Modus recognizes as a valid scan flop in the chain. This is the flop to start tracing backwards for debug.

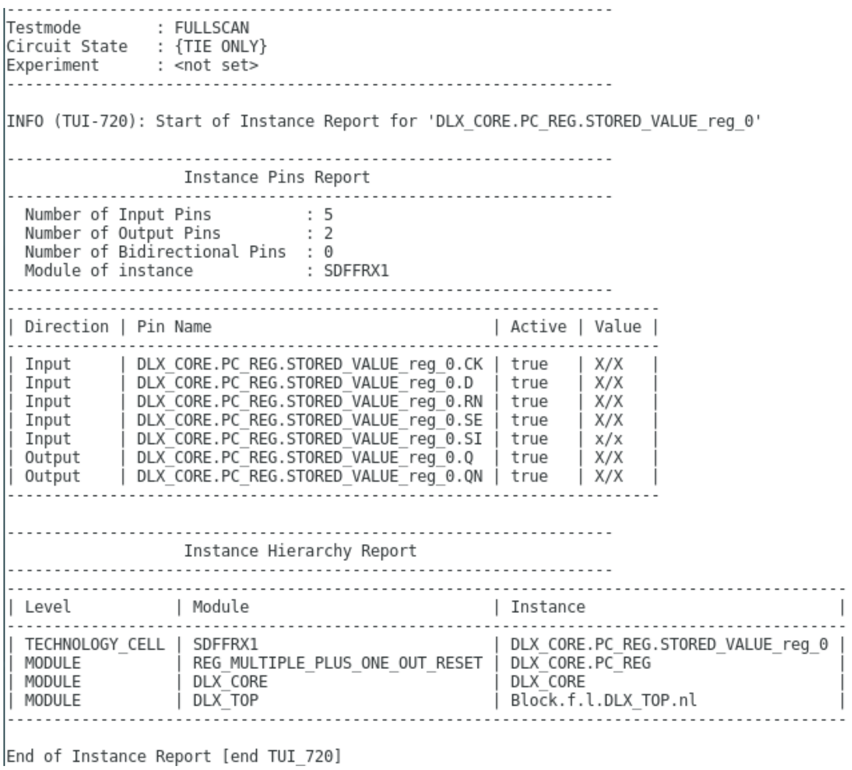
Notice the last FF instance name:
`DLX_CORE.PC_REG.STORED_VALUE_reg_0.__iNsT2.dff_primitive`

Note: The last 2 elements of this name highlighted in red are Modus primitive level elements. There is no need to include these in the instance name when debugging.

2. Fetch the information of the last valid scan flop and try to find issues around it. To fetch more information on the last scan element, use the command:

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

The output of the report instance is:



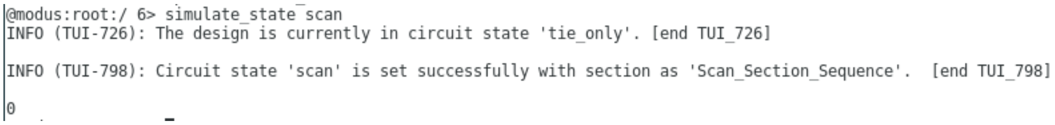
```
-----------------------------------------------------------------
Testmode        : FULLSCAN
Circuit State   : {TIE ONLY}
Experiment      : <not set>
-----------------------------------------------------------------

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

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

-----------------------------------------------------------------
                   Instance Hierarchy Report
-----------------------------------------------------------------
------------------------------------------------------------------------------------------
| Level           | Module                          | Instance                           |
------------------------------------------------------------------------------------------
| TECHNOLOGY_CELL | SDFFRX1                         | DLX_CORE.PC_REG.STORED_VALUE_reg_0 |
| MODULE          | REG_MULTIPLE_PLUS_ONE_OUT_RESET | DLX_CORE.PC_REG                    |
| MODULE          | DLX_CORE                        | DLX_CORE                           |
| MODULE          | DLX_TOP                         | Block.f.l.DLX_TOP.nl               |
------------------------------------------------------------------------------------------

End of Instance Report [end TUI_720]
```

The above report_instances output gives you the type of technology cell, all the specific pins, names and values along with the hierarchy report of the instance. Notice that the values for all the pins are at **X**. In this case it is because we have not yet setup the circuit in any particular simulated state. The available states are the same as those available in the GUI.

3. Let’s set the circuit in scan state by using the command:

   ```
   simulate_state scan
   ```

   The above command will set us in the scan state. This state will assert the shift enable signals and test constraints necessary for proper shifting of the scan chains.

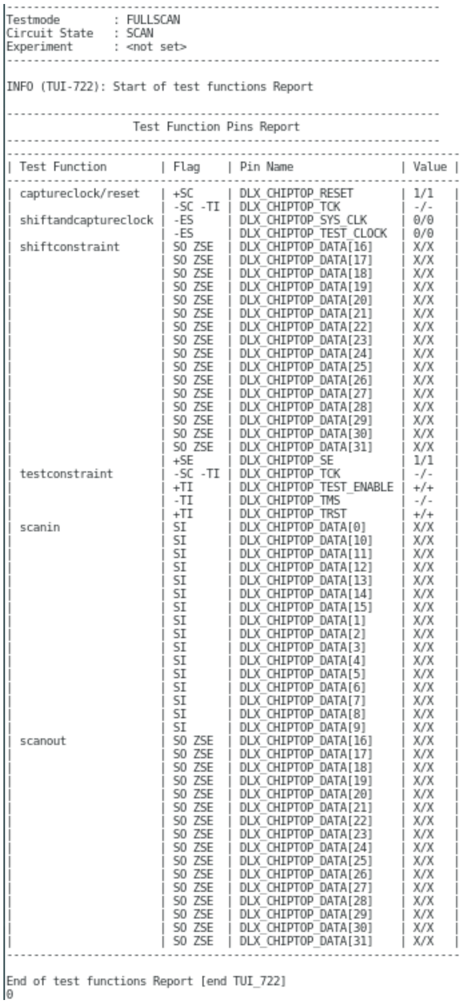


```
@modus:root:/ 6> simulate_state scan
INFO (TUI-726): The design is currently in circuit state 'tie_only'. [end TUI_726]

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

0
```

You can check that all the testfunction pins are correctly simulated. To report all the testfunction using the command:

```
report_testfunctions
```

The output of the report_testfunctions is:

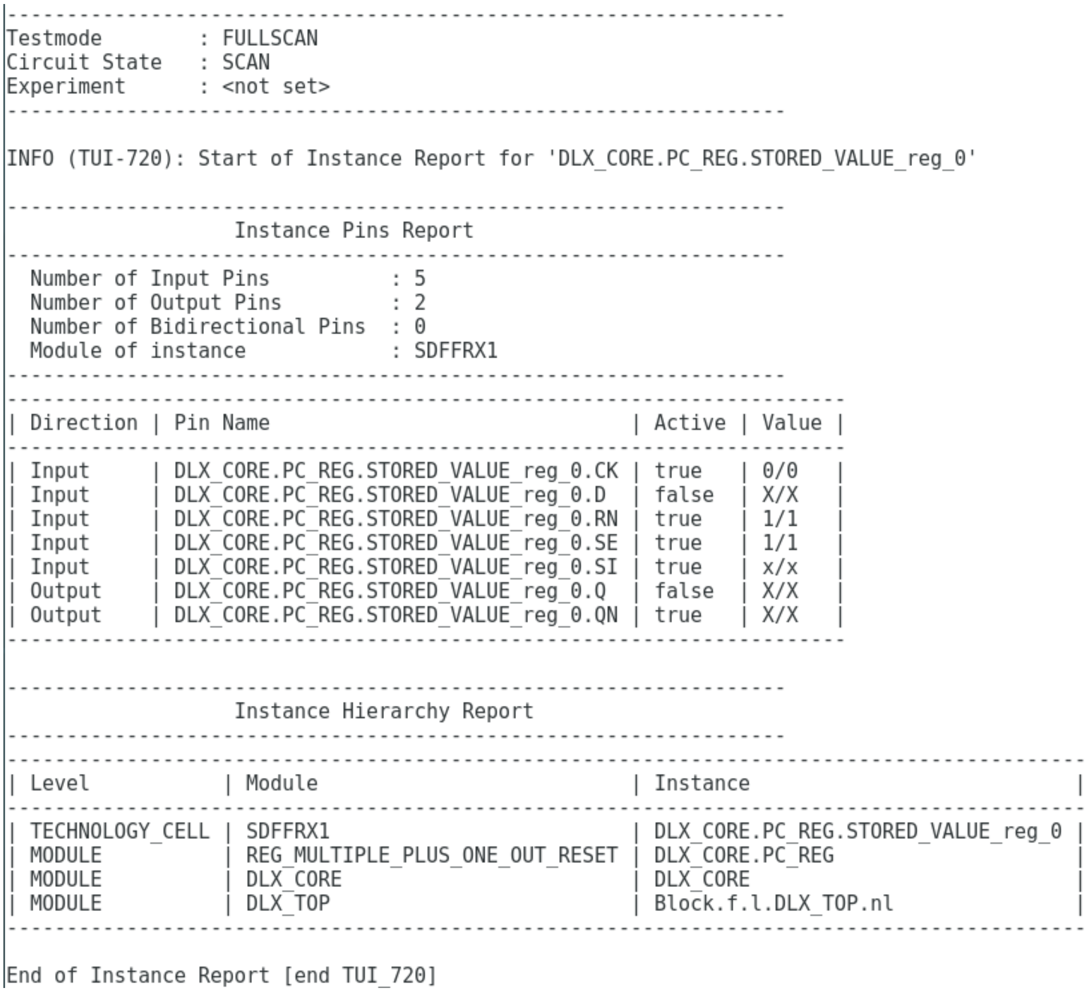


```
----------------------------------------------------------------
Testmode        : FULLSCAN
Circuit State   : SCAN
Experiment      : <not set>
----------------------------------------------------------------

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

----------------------------------------------------------------
                   Test Function Pins Report
----------------------------------------------------------------
-------------------------------------------------------------------
| Test Function       | Flag    | Pin Name                 | Value |
-------------------------------------------------------------------
| captureclock/reset  | +SC     | DLX_CHIPTOP_RESET        | 1/1   |
|                     | -SC -TI | DLX_CHIPTOP_TCK          | -/-   |
| shiftandcaptureclock | -ES    | DLX_CHIPTOP_SYS_CLK      | 0/0   |
|                     | -ES     | DLX_CHIPTOP_TEST_CLOCK   | 0/0   |
| shiftconstraint     | SO ZSE  | DLX_CHIPTOP_DATA[16]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[17]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[18]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[19]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[20]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[21]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[22]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[23]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[24]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[25]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[26]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[27]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[28]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[29]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[30]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[31]     | X/X   |
|                     | +SE     | DLX_CHIPTOP_SE           | 1/1   |
| testconstraint      | -SC -TI | DLX_CHIPTOP_TCK          | -/-   |
|                     | +TI     | DLX_CHIPTOP_TEST_ENABLE  | +/+   |
|                     | -TI     | DLX_CHIPTOP_TMS          | -/-   |
|                     | +TI     | DLX_CHIPTOP_TRST         | +/+   |
| scanin              | SI      | DLX_CHIPTOP_DATA[0]      | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[10]     | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[11]     | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[12]     | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[13]     | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[14]     | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[15]     | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[1]      | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[2]      | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[3]      | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[4]      | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[5]      | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[6]      | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[7]      | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[8]      | X/X   |
|                     | SI      | DLX_CHIPTOP_DATA[9]      | X/X   |
| scanout             | SO ZSE  | DLX_CHIPTOP_DATA[16]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[17]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[18]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[19]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[20]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[21]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[22]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[23]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[24]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[25]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[26]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[27]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[28]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[29]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[30]     | X/X   |
|                     | SO ZSE  | DLX_CHIPTOP_DATA[31]     | X/X   |
-------------------------------------------------------------------

End of test functions Report [end TUI_722]
0
```

4. Reverify the values at the previously reported scan flop. Again, report the DLX_CORE.PC_REG.STORED_VALUE_reg_0 instance to check the values at every pin using the command:

   `report_instances DLX_CORE.PC_REG.STORED_VALUE_reg_0`

   The output of the report_instances command is:



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

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

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

-----------------------------------------------------------------
                   Instance Hierarchy Report
-----------------------------------------------------------------
------------------------------------------------------------------------------------------
| Level           | Module                          | Instance                           |
------------------------------------------------------------------------------------------
| TECHNOLOGY_CELL | SDFFRX1                         | DLX_CORE.PC_REG.STORED_VALUE_reg_0 |
| MODULE          | REG_MULTIPLE_PLUS_ONE_OUT_RESET | DLX_CORE.PC_REG                    |
| MODULE          | DLX_CORE                        | DLX_CORE                           |
| MODULE          | DLX_TOP                         | Block.f.l.DLX_TOP.nl               |
------------------------------------------------------------------------------------------

End of Instance Report [end TUI_720]
```

We now have the logical values on the flop we are starting our debug with. What are the pins we care about on a scan flop during shift? Clocks, Resets, Shift Enable, Shift Input pins. The flop's pin values are:

a. CK = 0 (At it's OFF state)

b. RN=1 (Reset is OFF)

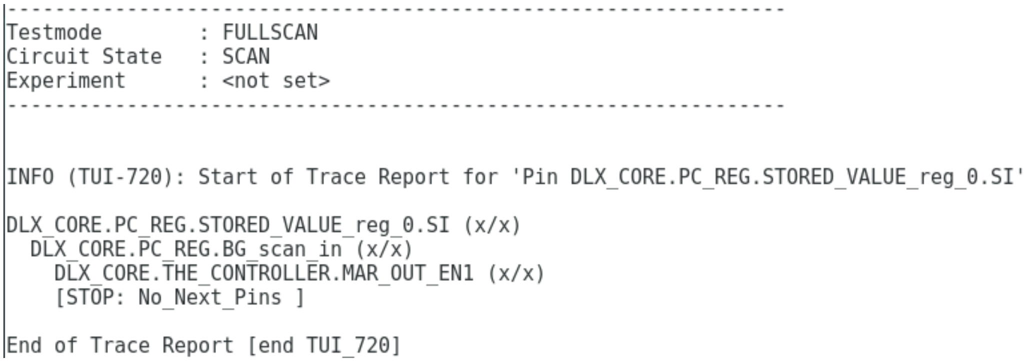
c. SE=1 (Shift Enable is active)

d. SI=x This can be an issue, a small x means it is not driven by any logic.

5. We can do a trace back to verify the connected structure. To trace back from SI pin of this flop and search the logic driving the SI pin of this flop, use the command:

```
trace_circuit -backward DLX_CORE.PC_REG.STORED_VALUE_reg_0.SI
```

Result of the trac back is:



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


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

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

End of Trace Report [end TUI_720]
```

We have now trace back the SI pin of the flop DLX_CORE.PC_REG.STORED_VALUE_reg_0, looking at the summary of the trace_circuit report answer the following questions:

What does the trace results from above tell us?

Does it match what you found in the GUI debug section?

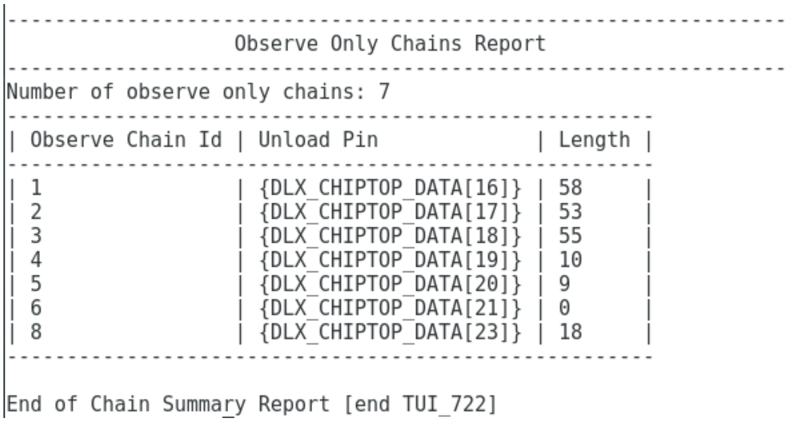
Debugging Observe Scan Chain 4

Let’s try one more to get the hang of it. Look at Observe Chain 3 next.

1. Report the scan chain summary, by using the command:

   ```
   report_chains –summary
   ```

   Look at the observe only chain on the report:

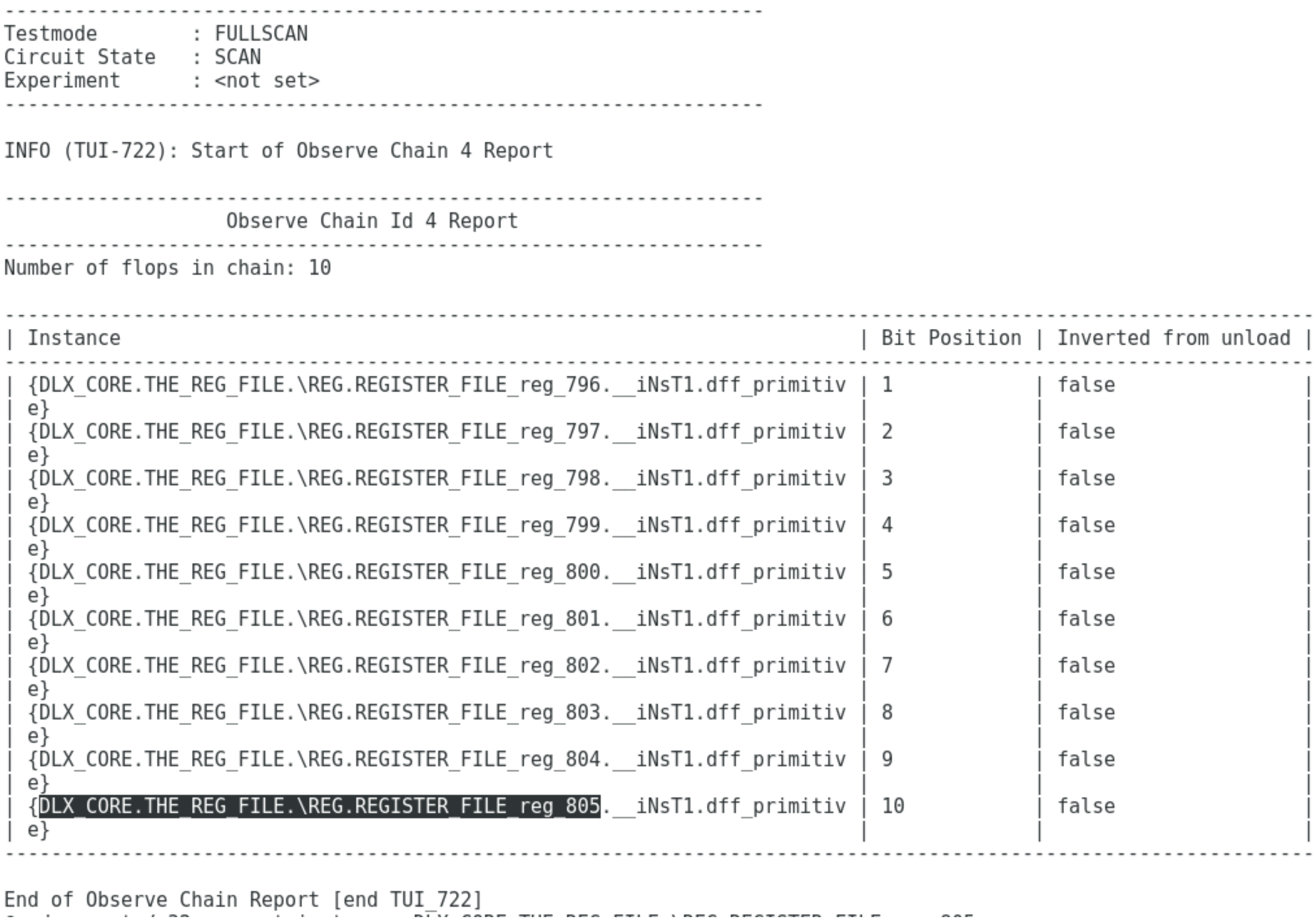


```
------------------------------------------------------------------
                   Observe Only Chains Report
------------------------------------------------------------------
Number of observe only chains: 7
-------------------------------------------------------
| Observe Chain Id | Unload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

End of Chain Summary Report [end TUI_722]
```

2. Report the observe only chain DLX_CHIPTOP_DATA[18] with chian ID 3, bu using the command:

   ```
   report_chains -observechain 4
   ```

   The output report is:



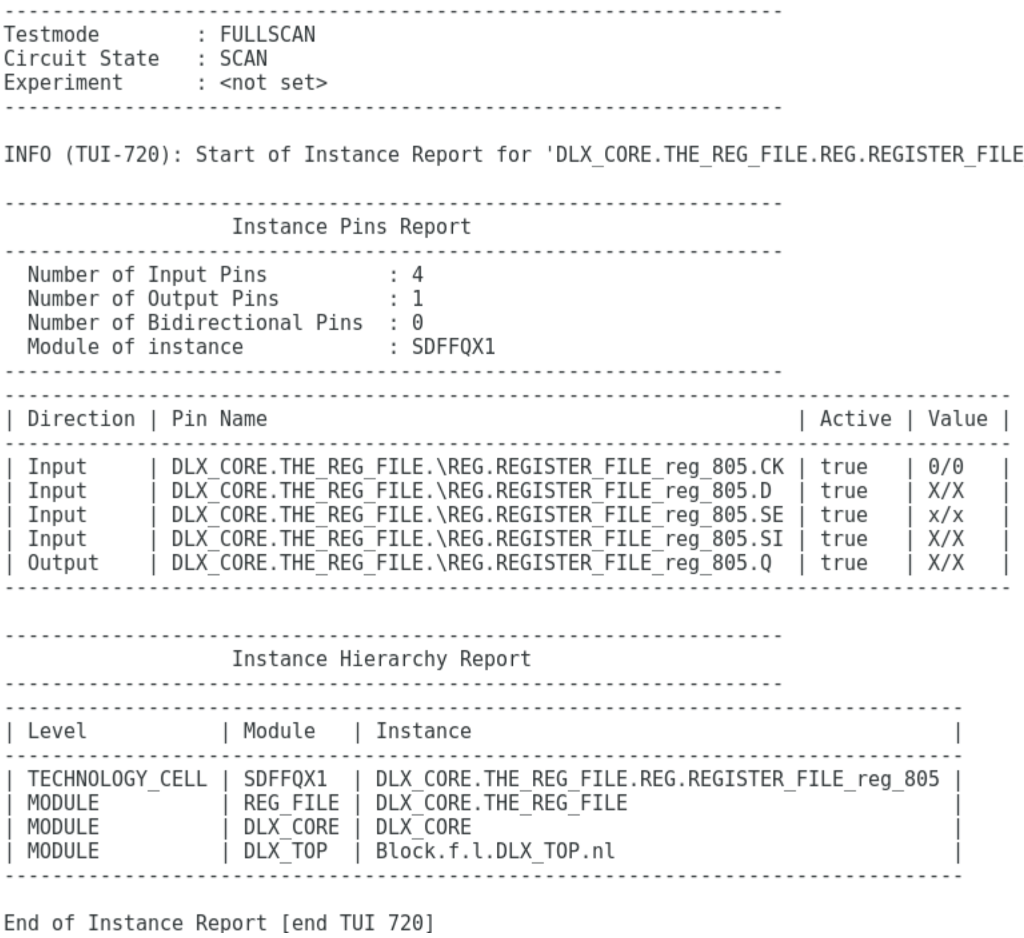
```
-----------------------------------------------------------------
Testmode        : FULLSCAN
Circuit State   : SCAN
Experiment      : <not set>
-----------------------------------------------------------------

INFO (TUI-722): Start of Observe Chain 4 Report

-----------------------------------------------------------------
                   Observe Chain Id 4 Report
-----------------------------------------------------------------
Number of flops in chain: 10

-----------------------------------------------------------------------------------------------------------------
| Instance                                                          | Bit Position | Inverted from unload |
-----------------------------------------------------------------------------------------------------------------
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_796.__iNsT1.dff_primitiv | 1            | false                |
| e}                                                                | 1            |                      |
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_797.__iNsT1.dff_primitiv | 2            | false                |
| e}                                                                |              |                      |
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_798.__iNsT1.dff_primitiv | 3            | false                |
| e}                                                                |              |                      |
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_799.__iNsT1.dff_primitiv | 4            | false                |
| e}                                                                |              |                      |
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_800.__iNsT1.dff_primitiv | 5            | false                |
| e}                                                                |              |                      |
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_801.__iNsT1.dff_primitiv | 6            | false                |
| e}                                                                |              |                      |
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_802.__iNsT1.dff_primitiv | 7            | false                |
| e}                                                                |              |                      |
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_803.__iNsT1.dff_primitiv | 8            | false                |
| e}                                                                |              |                      |
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_804.__iNsT1.dff_primitiv | 9            | false                |
| e}                                                                |              |                      |
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.__iNsT1.dff_primitiv | 10           | false                |
| e}                                                                |              |                      |
-----------------------------------------------------------------------------------------------------------------

End of Observe Chain Report [end TUI_722]
```

Notice the analysis stopped at Observe Flop 10. This is the last Flop that Modus recognizes as a valid scan flop in the chain. This is the flop to start tracing backwards for debug.

3. Report the last good instance by using the command:

   ```
   report_instances DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805
   ```

   The output report will show the values at pins:



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

INFO (TUI-720): Start of Instance Report for 'DLX_CORE.THE_REG_FILE.REG.REGISTER_FILE_reg_805'

-----------------------------------------------------------------
                   Instance Pins Report
-----------------------------------------------------------------
  Number of Input Pins          : 4
  Number of Output Pins         : 1
  Number of Bidirectional Pins  : 0
  Module of instance            : SDFFQX1
-----------------------------------------------------------------
------------------------------------------------------------------------------------
| Direction | Pin Name                                             | Active | Value |
------------------------------------------------------------------------------------
| Input     | DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.CK | true   | 0/0   |
| Input     | DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.D  | true   | X/X   |
| Input     | DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.SE | true   | x/x   |
| Input     | DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.SI | true   | X/X   |
| Output    | DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.Q  | true   | X/X   |
------------------------------------------------------------------------------------

-----------------------------------------------------------------
                   Instance Hierarchy Report
-----------------------------------------------------------------
--------------------------------------------------------------------------------
| Level           | Module   | Instance                                         |
--------------------------------------------------------------------------------
| TECHNOLOGY_CELL | SDFFQX1  | DLX_CORE.THE_REG_FILE.REG.REGISTER_FILE_reg_805 |
| MODULE          | REG_FILE | DLX_CORE.THE_REG_FILE                            |
| MODULE          | DLX_CORE | DLX_CORE                                         |
| MODULE          | DLX_TOP  | Block.f.l.DLX_TOP.nl                             |
--------------------------------------------------------------------------------

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

We now have the logical values on the flop we are starting our debug with. What are the pins we care about on a scan flop during shift? Clocks, Resets, Shift Enable, Shift Input pins. The flop's pin values are:

a. CK = 0 (At it's OFF state)

b. D=X (Data pin looks good)

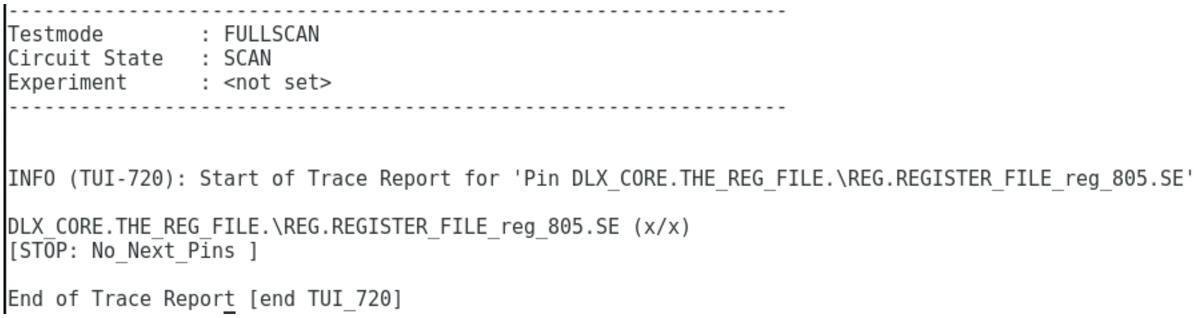
c. SE=x (This can be an issue, a small x means it is not driven by any logic)

d. SI=X (Scan in pin is at correct value)

4. Let's trace back the SE pin by using the command:

```
trace_circuit -backward DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.SE
```

The output of the trace_circuit is:



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


INFO (TUI-720): Start of Trace Report for 'Pin DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.SE'

DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.SE (x/x)
[STOP: No_Next_Pins ]

End of Trace Report [end TUI_720]
```

We have now trace back the SE pin of the instance, looking at the summary of the trace_circuit report and found that SE pin is not connected.

5. Go ahead and try to debug all the other broken chains and match them to your results with the GUI debug!

Exiting the Modus

Close the Modus tool by entering the command:

```
exit
```

End of Lab

Module 5: Analyze Initialization Sequence Using Modus GUI

Lab 5-1 Analyzing Initialization Sequences Using Modus GUI

Objective: To get familiarize with the process of sequences analysis and debug GUI.

In this lab, the design is processed from Build Model to Verify Test Structures. This will help you learn how to bring up the analyze sequences GUI with a watchlist. This method is very valuable when your test initialization sequence does not get you into the proper test mode or the proper number of scan chains.

Invoking Modus

Let’s create a new test model and mode to learn sequence analysis. Open Modus with GUI and source run_lab4.tcl present in the Jumpstart_LABS/LAB3.

1. Change the work directory to LAB3 by:

```
cd JumpStart_LABS/LAB3
```

2. Open the Modus tool with GUI by using the command:

```
modus -gui
```

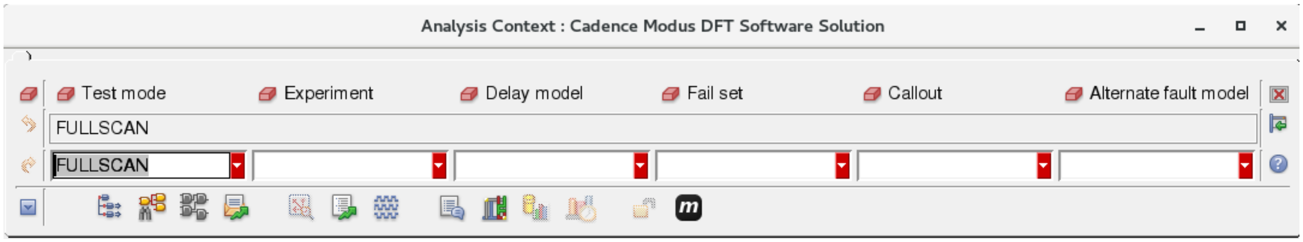
3. Source the run_lab4.tcl file in Modus:

```
source ./run_lab4.tcl
```

Note: This Tcl file should run the build_model, build_testmode and verify_test_structures. Open the run_lab4.tcl file and view the content.

Setting the Analysis Context and Opening the Sequence Analyzer

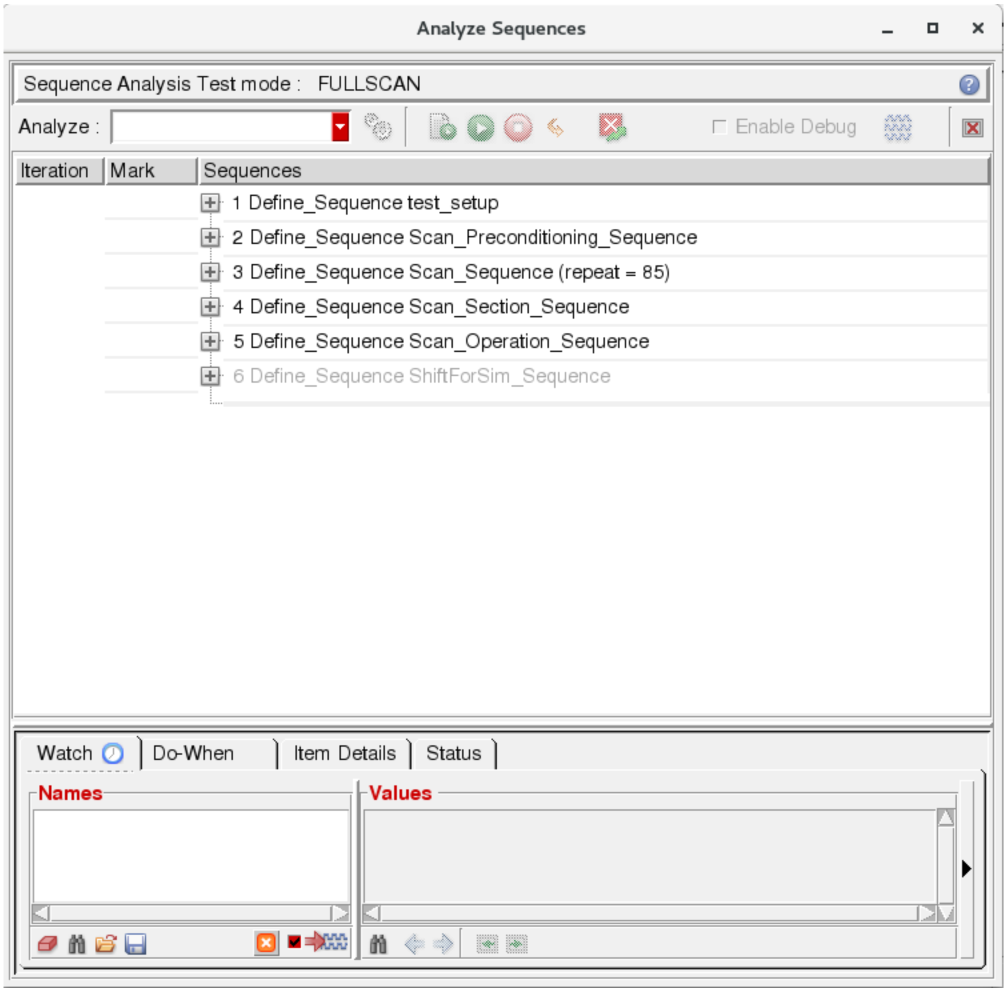
1. In the Modus GUI and click on the **Analysis Context** icon (3rd from far right) and set the Test mode to FULLSCAN in the pulldown. Click yellow arrow to apply and then close the window. This will set the GUI into the testmode FULLSCAN for analysis.



Analysis Context : Cadence Modus DFT Software Solution

Test mode: FULLSCAN

2. Now click on the **Analyze Sequences** icon (7th from left of main group). This will bring up the “Analyze Sequences” window.

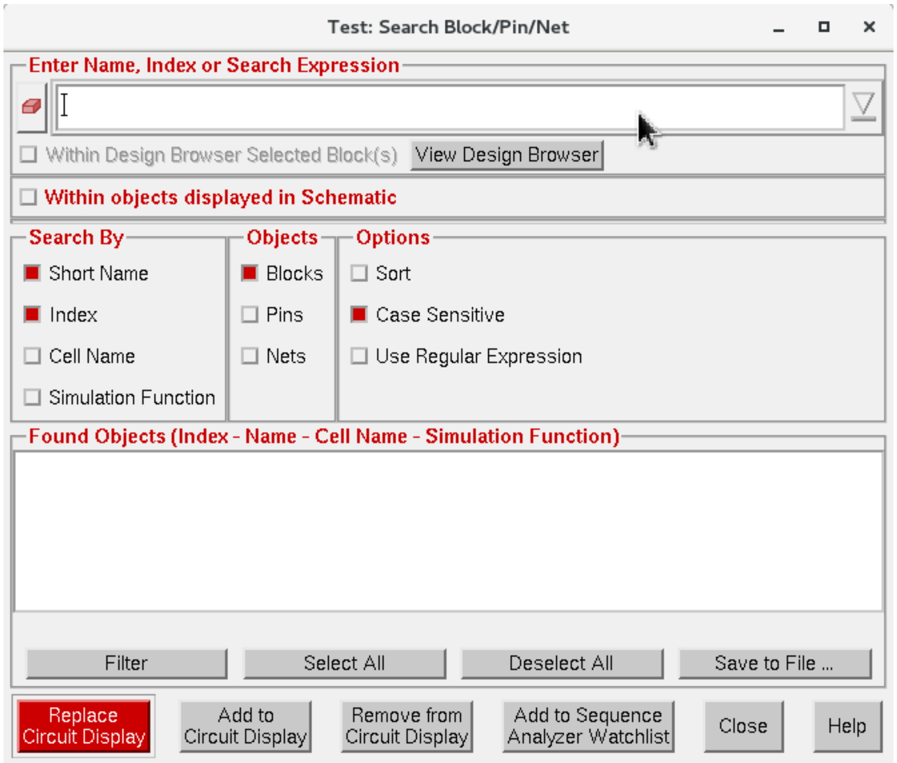


Analyze Sequences
Sequence Analysis Test mode : FULLSCAN
Analyze :
□ Enable Debug
Iteration
Mark
Sequences
1 Define_Sequence test_setup
2 Define_Sequence Scan_Preconditioning_Sequence
3 Define_Sequence Scan_Sequence (repeat = 85)
4 Define_Sequence Scan_Section_Sequence
5 Define_Sequence Scan_Operation_Sequence
6 Define_Sequence ShiftForSim_Sequence
Watch
Do-When
Item Details
Status
Names
Values

Opening the Schematic and Creating a Watch List

We now want to go to the schematic GUI and find signals we want to trace and look at inside this window.

1. In main GUI click icon of main analysis group. Mouse over the icon should say **View DesignSearch (Block)**.



Test: Search Block/Pin/Net

Enter Name, Index or Search Expression

□ Within Design Browser Selected Block(s)

□ Within objects displayed in Schematic

Search By

☒ Short Name

☒ Index

□ Cell Name

□ Simulation Function

Objects

☒ Blocks

□ Pins

□ Nets

Options

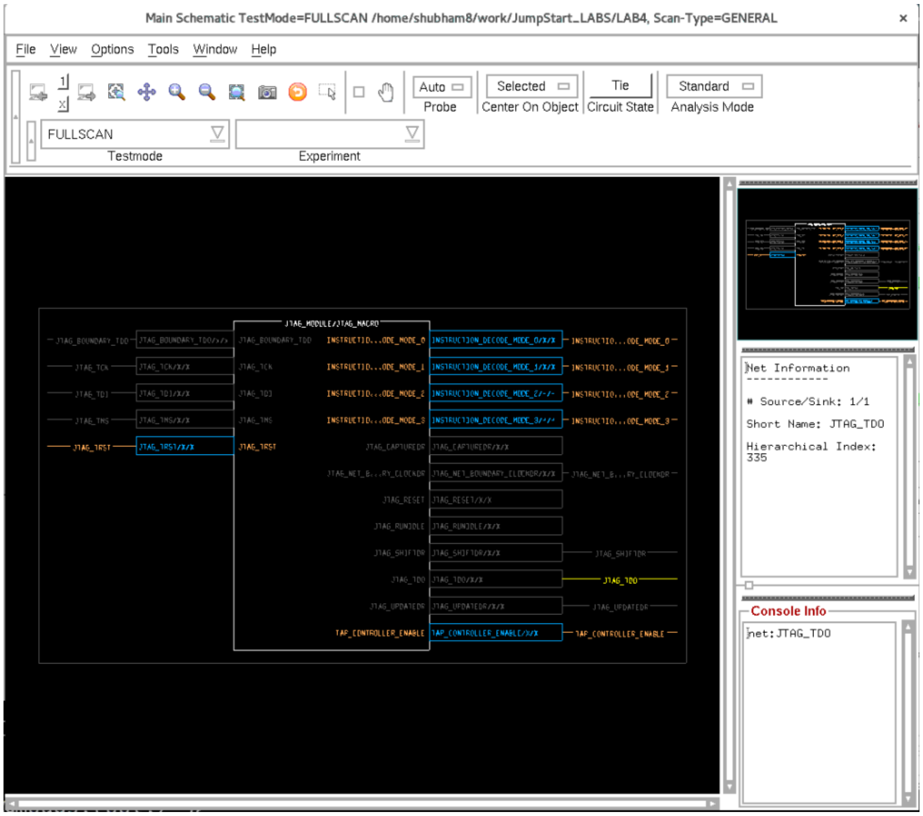
□ Sort

☒ Case Sensitive

□ Use Regular Expression

Found Objects (Index - Name - Cell Name - Simulation Function)

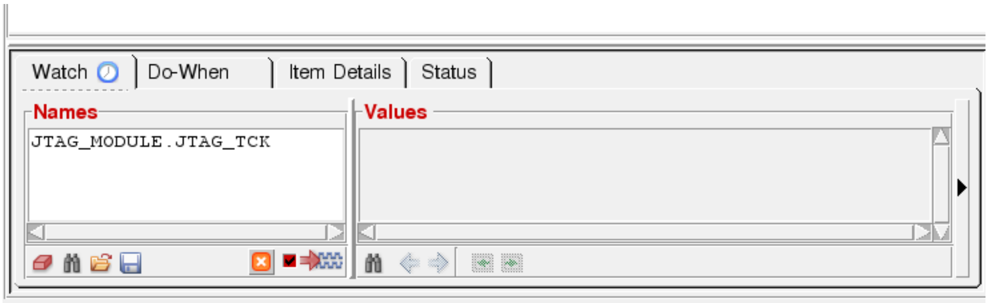
2. Type in **JTAG_MODULE** into search field and hit return. This will bring up the JTAG module into the schematic view. We want to capture all of the pins into a watch list, so we can track them in the analyze sequence GUI.



```
Net Information
---------------
# Source/Sink: 1/1
Short Name: JTAG_TDO
Hierarchical Index:
335
```

```
net:JTAG_TDO
```

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



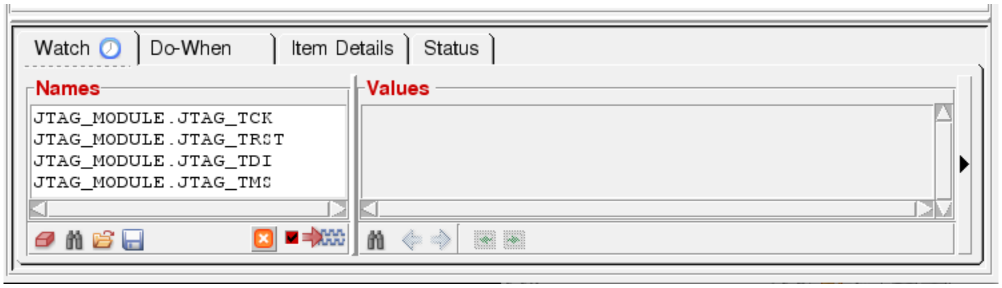
```
JTAG_MODULE.JTAG_TCK
```

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

Analyzing Sequences

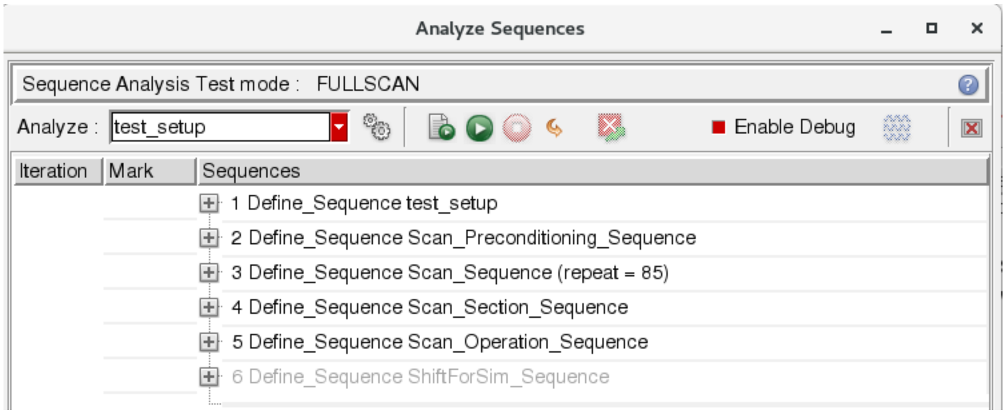
We now want to analyze the sequences.

1. In the Analyze Sequences window. You will see the 4 pins in the Watch window and just below them are 6 icons. Make sure the icon on the right side is activated. This will enable waveform recording of the simulation of these pins.



JTAG_MODULE.JTAG_TCK
JTAG_MODULE.JTAG_TRST
JTAG_MODULE.JTAG_TDI
JTAG_MODULE.JTAG_TMS

2. At top of window select test_setup in the Analyze pulldown. This selects the Initialization sequence that we used in the **DLX.seqdef** file.



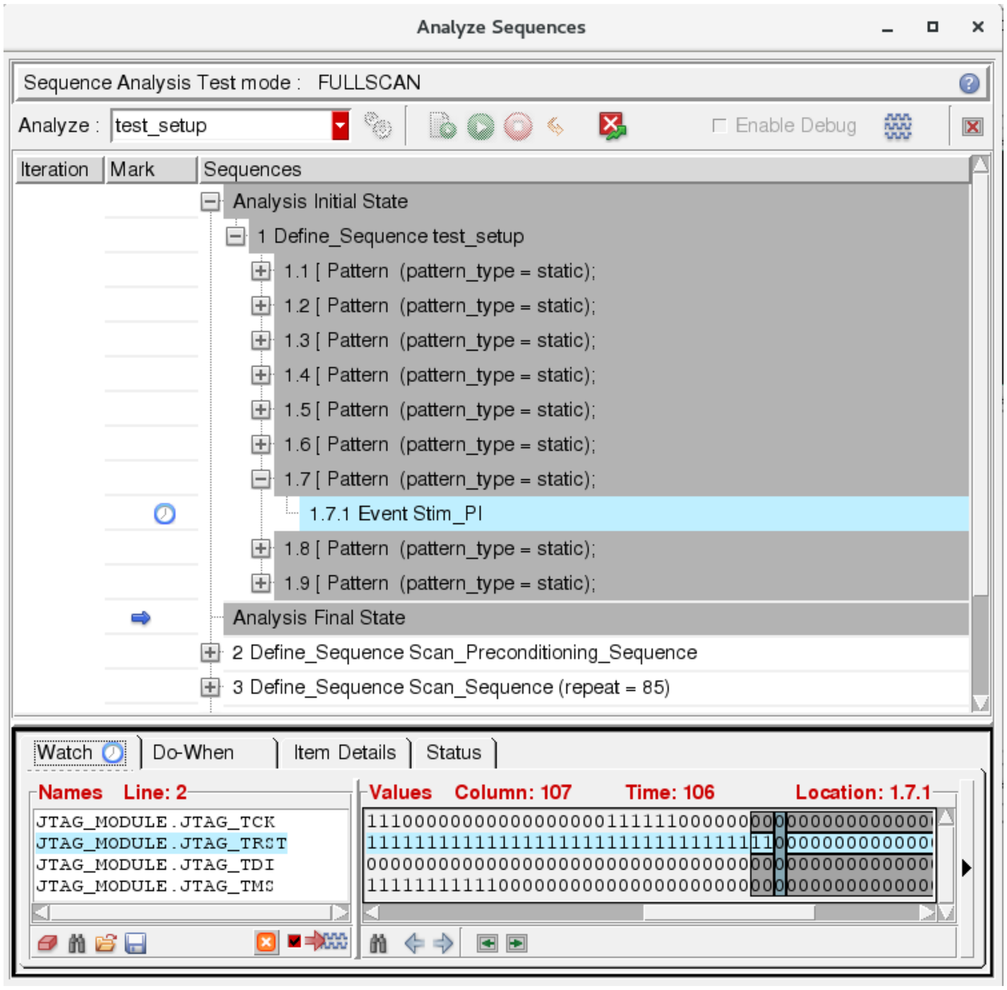
Sequence Analysis Test mode : FULLSCAN

Analyze : test_setup

1 Define_Sequence test_setup
2 Define_Sequence Scan_Preconditioning_Sequence
3 Define_Sequence Scan_Sequence (repeat = 85)
4 Define_Sequence Scan_Section_Sequence
5 Define_Sequence Scan_Operation_Sequence
6 Define_Sequence ShiftForSim_Sequence

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

4. Now, click the Large green play button . The full initialization sequence will run to completion. In the bit stream output below, click on any of the bit values and notice the changes in the different windows.



Analyze Sequences

Sequence Analysis Test mode : FULLSCAN

Analyze : test_setup

□ Enable Debug

Iteration	Mark	Sequences
		Analysis Initial State
		1 Define_Sequence test_setup
		1.1 [Pattern (pattern_type = static);
		1.2 [Pattern (pattern_type = static);
		1.3 [Pattern (pattern_type = static);
		1.4 [Pattern (pattern_type = static);
		1.5 [Pattern (pattern_type = static);
		1.6 [Pattern (pattern_type = static);
		1.7 [Pattern (pattern_type = static);
	🕒	1.7.1 Event Stim_PI
		1.8 [Pattern (pattern_type = static);
		1.9 [Pattern (pattern_type = static);
	➡	Analysis Final State
		2 Define_Sequence Scan_Preconditioning_Sequence
		3 Define_Sequence Scan_Sequence (repeat = 85)

Watch | Do-When | Item Details | Status

Names Line: 2

```
JTAG_MODULE.JTAG_TCK
JTAG_MODULE.JTAG_TRST
JTAG_MODULE.JTAG_TDI
JTAG_MODULE.JTAG_TMS
```

Values Column: 107 Time: 106 Location: 1.7.1

```
11100000000000000000111111000000000000000000000000
11111111111111111111111111111111110000000000000
00000000000000000000000000000000000000000000000
11111111111000000000000000000000000000000000000
```

5. Now let us set a condition that will automatically stop when JTAG_TRST goes from 1 to 0.

   a. Click the Red X w/green arrow icon at the top in the Analyze Sequence window. This will terminate and reset the analysis. Close out the SimVision windows.

   b. Select test_setup again from the Analyze pull-down menu.

   c. Click on Do-When tab Do-When near the bottom, we will set the condition to stop the analysis when **JTAG_MODULE.JTAG_TRST** goes 1 to 0.

   d. Click **green +** icon.

   e. Select **JTAG_TRST** from the pulldown menu.

   f. Select From : **1** To : **0** and click **OK**. You should see this condition.

   g. Click back on the Watch tab and Go back to top and click the small green play button again and then click the large green play button and answer the following questions.

- Where is the Blue Mark arrow?
- Is the proper transition highlighted in the Watch window?

Exiting the Modus

Close the Modus GUI, by entering the command:

```
exit
```

End of Lab

Module 6: Debugging the Test Pattern

Lab 6-1 Simulating and Debugging Vectors

Objective: To learn the vector simulation and debug environment.

This lab helps to simulate and debug the vectors in SimVision™ and Xmverilog environment.

This lab uses the following softwares

- Modus 22.10
- XCELIUM xm-Verilog simulator XCELIUM22.01.001

Xcelium and Modus Invoking Flow

- First make sure that the **XCELIUM (xmverilog)** software is in your path and that it is before the Modus software. Type **which xmverilog** and make sure it comes from an IUS path and not Modus path.
- We will create a standard faultmodel (stuck-at) and generate tests for them.
  - Simulate the tests in **xmsim** and verify they pass.
  - Bring up the passing vectors in **Modus** and trace them in the schematic viewer.
- Create a **Bridging Faultmodel** which contains only a small set of faults.
  - Create targeted tests for these few faults and simulate them.
  - Simulate the patterns against a faulty circuit and see how the tests detect the failures.
  - Bring up the tests in simulation debug environment and analyze the failures.

Invoking Modus and Generating Vectors

1. Change the work directory to JumpStart_LABS/LAB4 by using the command:

   ```
   cd JumpStart_LABS/LAB4
   ```

2. Invoke Modus tool by using the command:

   ```
   modus
   ```

3. Once you have the Modus command line prompt available, run the script **run_modus.atpg.tcl** to execute the flow and generate the vectors. To run the script use the command

   ```
   source run_modus.atpg.tcl
   ```

Note: The script will execute build_model -> build_testmode -> verify_test_structures -> build_faultmodel -> create_logic_tests -> write_vectors. You should review the script and should be able understand it now. Look at the write_vectors command we will be writing out the parallel vectors in Verilog language.

Running Good Simulation

In this section we will learn to run the simulation of ATPG generated vectors.

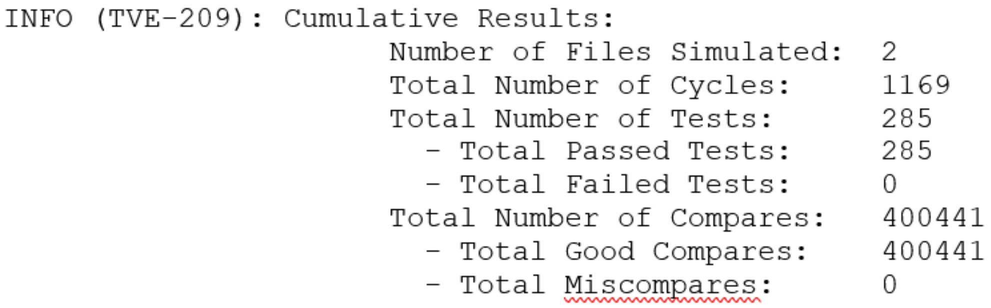
1. The verilogsim/run_sim_good script contains the command to run the simulation of Verilog vectors. Open it and go through the content by using the command:

   ```
   gvim verilogsim/run_sim_good
   ```

   Go ahead and run the script on a new shell by using the command:

   ```
   user_prompt:/> ./verilogsim/run_sim_good
   ```

   In the output summary Notice, that there are two test files in the script corresponding to scan chain tests and logic tests. you should now see 0 miscompares:



```
INFO (TVE-209): Cumulative Results:
                      Number of Files Simulated:  2
                      Total Number of Cycles:     1169
                      Total Number of Tests:      285
                        - Total Passed Tests:     285
                        - Total Failed Tests:     0
                      Total Number of Compares:   400441
                        - Total Good Compares:    400441
                        - Total Miscompares:      0
```

Running Bad Simulation

Let’s run the simulation on a faulty netlist and see check if we get some miscompares. To induce a failure we have inserted a **stuck at fault** inside the **DLX_CORE** netlist. We can now run Verilog simulation against the faulty netlist **DLX_CORE_fault.v**.

1. To execute the run_sim_bad script, use the command:

   ```
   user_prompt:/> ./verilogsim/run_sim_bad
   ```

   After looking at the output summary, answer the following questions:

   How many good comparing vectors are there?

   How many bad comparing vectors are there?

You should now see some miscompares and we will debug them in the next section.

Debugging Simulation Miscompares

For debugging the scan-based designs, the first step is to identify the failing/faulty flop. There are multiple ways to get this information. Since, we are using **scanformat parallel** we can write out a Verilog testbench which can directly provide information about the failing flop.

1. To write the vectors use the command at the Modus prompt:

   ```
   write_vectors -testmode FULLSCAN -inexperiment logic -language verilog \
   -scanformat parallel -includelatchnames
   ```

   Note: Use the **–includelatchnames** keyword with write_vectors to write out the testbench. The **-includelatchnames** is only supported with parallel Verilog vectors. If you are running a serial simulation this is not possible.

2. Rerun the Verilog simulation (./verilogsim/run_sim_bad) with newly written simulation testbench. You should see 9 miscompares. Open the log file **xmverilog_FULLSCAN.log** and search for miscompares. You can identify that the failing location is **DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_0**.

3. Another method is shown in the script **run_findflops.tcl**. This is known as diagnostics run. When you run Verilog testbenches with the +FAILSET switch, the code will write out a file **“./testresults/verilog/VER.FULLSCAN.logic.data.logic.ex1.ts2.verilog_FAILSET**” in CPP format which we use for diagnostics.

   a. Open the run_findflops.tcl script file and go through the content by using the command:

      ```
      gvim ./verilogsim/run_findflops.tcl
      ```

   b. Source the script file in the Modus prompt by using the command:

      ```
      source ./verilogsim/run_findflops.tcl
      ```

   c. Check out the log created in the ./testresults/logs/log_analyze_failset_FULLSCAN_logic_logic and search for **Measure Location**.

   d. You will also see the odometer fail locations and SO pin information. Once again you can identify that the failing location is **DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_0**. This is where we should concentrate debug.

Good Machine Simulation

We need to compare the difference between the good simulation and the faulty simulation. The Good simulation is referred as the simulation done by Modus. While the bad simulation is one performed with a faulty netlist using xrun.

Let’s also look at the GOOD machine values in SimVision. You need to create a list of blocks/instances or pins you want to visualize in waveform viewer. One such list available in file **watchlist.txt**.

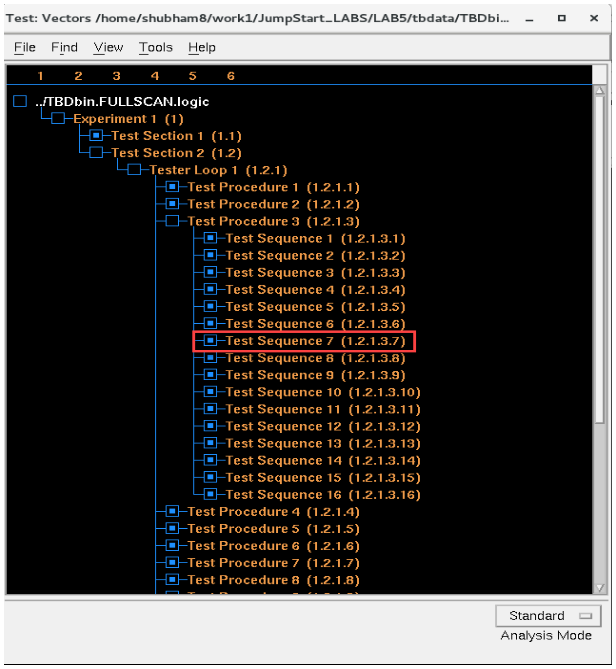
1. Run analyze_vectors at the Modus prompt to generate the SimVision database of the required pins.

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

   Note: Above run will generate the waveforms for the required failing patterns. Since we want to look at **1.2.1.3.7**. Let’s keep the vectors range to **1.2.1.3.1** to **1.2.1.4.1**

2. Bring it up Modus GUI using **gui_open** and set the analysis context with testmode as FULLSCAN and experiment as logic.

3. On the bottom-right of the analysis context window click on the **View Vectors** icon which is the 5th from the left of the large icons. This will bring up the TBDbin vector set we just simulated. Now we want to find that failed vector from the xmsim log file. Expand out the tree until you get to Pattern **1.2.1.3.7**


Test: Vectors /home/shubham8/work1/JumpStart_LABS/LAB5/tbdata/TBDbi...
File
Find
View
Tools
Help
1
2
3
4
5
6
Standard
Analysis Mode

```
../TBDbin.FULLSCAN.logic
  Experiment 1 (1)
    Test Section 1 (1.1)
    Test Section 2 (1.2)
      Tester Loop 1 (1.2.1)
        Test Procedure 1 (1.2.1.1)
        Test Procedure 2 (1.2.1.2)
        Test Procedure 3 (1.2.1.3)
          Test Sequence 1 (1.2.1.3.1)
          Test Sequence 2 (1.2.1.3.2)
          Test Sequence 3 (1.2.1.3.3)
          Test Sequence 4 (1.2.1.3.4)
          Test Sequence 5 (1.2.1.3.5)
          Test Sequence 6 (1.2.1.3.6)
          Test Sequence 7 (1.2.1.3.7)
          Test Sequence 8 (1.2.1.3.8)
          Test Sequence 9 (1.2.1.3.9)
          Test Sequence 10 (1.2.1.3.10)
          Test Sequence 11 (1.2.1.3.11)
          Test Sequence 12 (1.2.1.3.12)
          Test Sequence 13 (1.2.1.3.13)
          Test Sequence 14 (1.2.1.3.14)
          Test Sequence 15 (1.2.1.3.15)
          Test Sequence 16 (1.2.1.3.16)
        Test Procedure 4 (1.2.1.4)
        Test Procedure 5 (1.2.1.5)
        Test Procedure 6 (1.2.1.6)
        Test Procedure 7 (1.2.1.7)
        Test Procedure 8 (1.2.1.8)
```

4. You can now click on it once and then **right**-click your mouse and choose **View Circuit Values**. This will actual go off and simulate the full vector set up to and including that point. So now you can bring up elements in the circuit and trace back their values.

   Note: The schematic window should have also popped up if it already wasn't.

   Try this out by clicking **View->Block** and type in **DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_0** and hit return. This will bring up the measure fail flop. If you now hit the **b** button it will start tracing back the logic and you can mouse over the pins and see the actual circuit values of the GOOD machine.

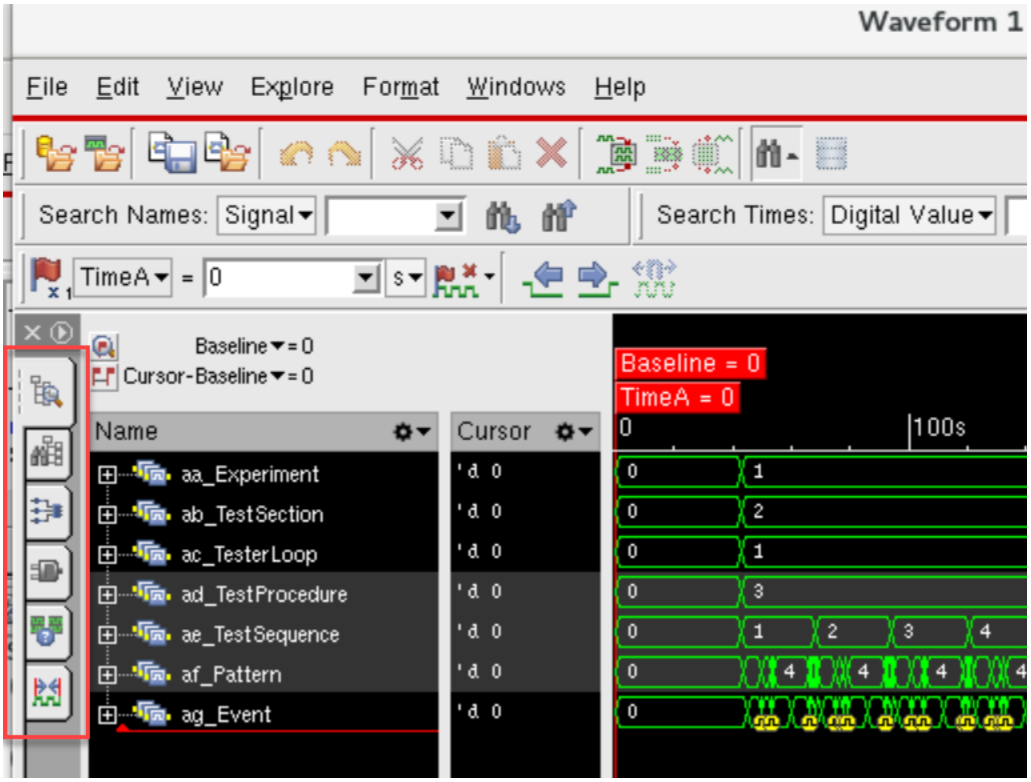
   What is the GOOD machine expected value on the Q pin?

   Does this match the expected value in our Verilog sim?

5. Go back to the main MODUS GUI and click on the **View Waveforms icon** which is on the right-side, 8th from the left of the large icons. A popup window comes up. Load the TBscope.FULSCAN.analyzed.trn file to open SimVision.

   a. In SimVision will see Experiment, TestSection, TestProcedure, TestSequence, etc., on the left side of the waveform window. You want to find the vertical binary combination of 1 2 1 3 1 in the waveform window. So, the first '1' value is the Experiment, the next '1' is the TestSection, etc.

   b. On the waveform window you have 6 Tabs on the left. Click on the top one. This brings up the **Design Browser**.



c. Remember when we Analyzed the Vectors and imported a watch list? These signals are now available in this Design Browser. Traverse through the design hierarchy and select the signal you wish to view in the waveforms. You should be able to easily find the flop **DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_0**.

d. You can zoom in/out using the icons . Explore the waveforms using these icons.

Do the values seen on waveform match that on schematic?

The above was an exercise to show you two different avenues to view the **GOOD** machine values expected by **MODUS**. One was through the **MODUS** schematic and the other through **SimVision**.

Bad Machine Simulation

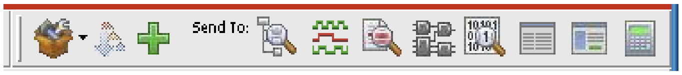
The next logical step would be to bring up SimVision on the BAD machine simulation. This means loading in our faulty Verilog simulation and comparing waveforms.

1. Open the script ./verilogsim/run_sim_bad and add the **+gui** option to the xrun command. Your command should look like:

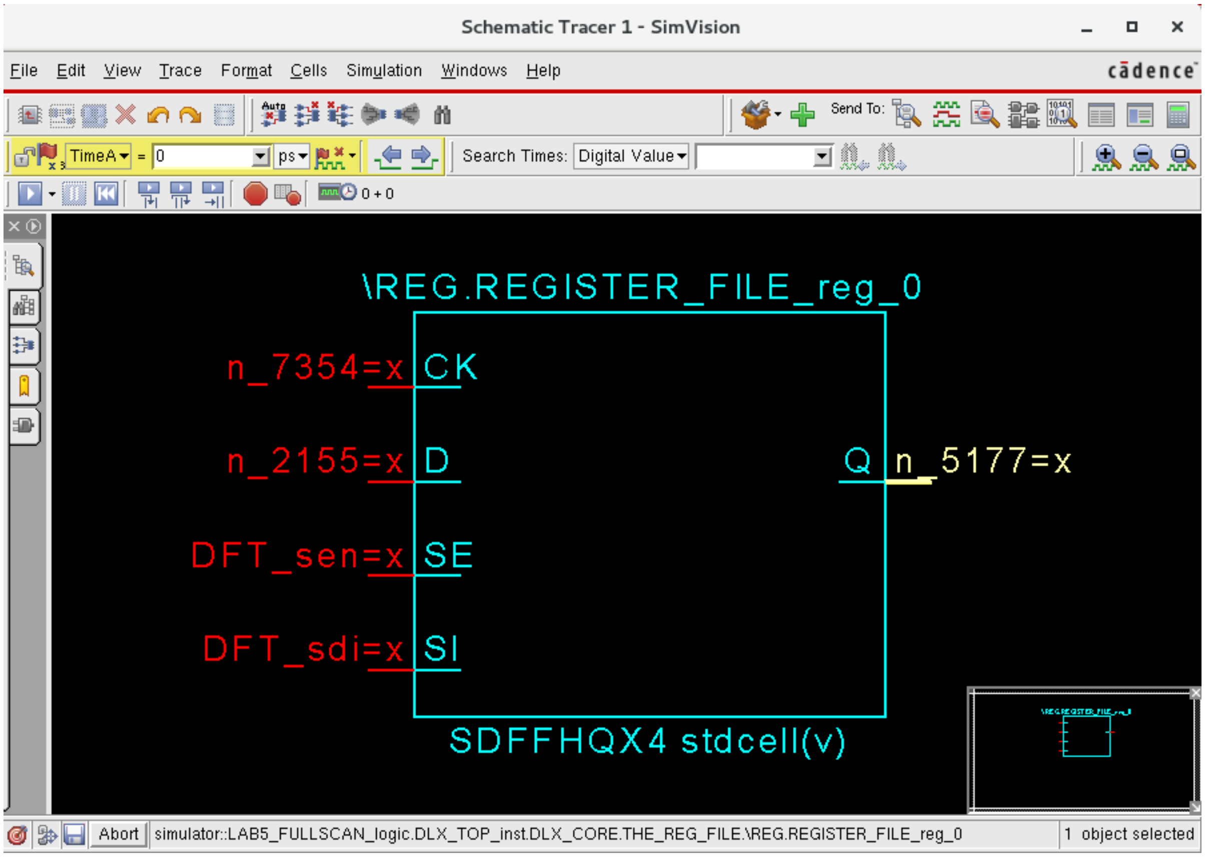
```
xrun \
   +access+rwc \
   +gui \
   +xmstatus \
   +xm64bit \
   +TESTFILE1=./testresults/verilog/VER.FULLSCAN.logic.data.scan.ex1.ts1.verilog \
   +TESTFILE2=./testresults/verilog/VER.FULLSCAN.logic.data.logic.ex1.ts2.verilog \
   +HEARTBEAT \
   +FAILSET \
   +xmtimescale+1ns/1ps \
   +xmoverride_timescale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

On the separate shell run ./verilogsim/run_sim_bad (or ./verilogsim/run_sim_bad_gui with +gui option already added) this should bring up Xcelium simulation window with SimVision.

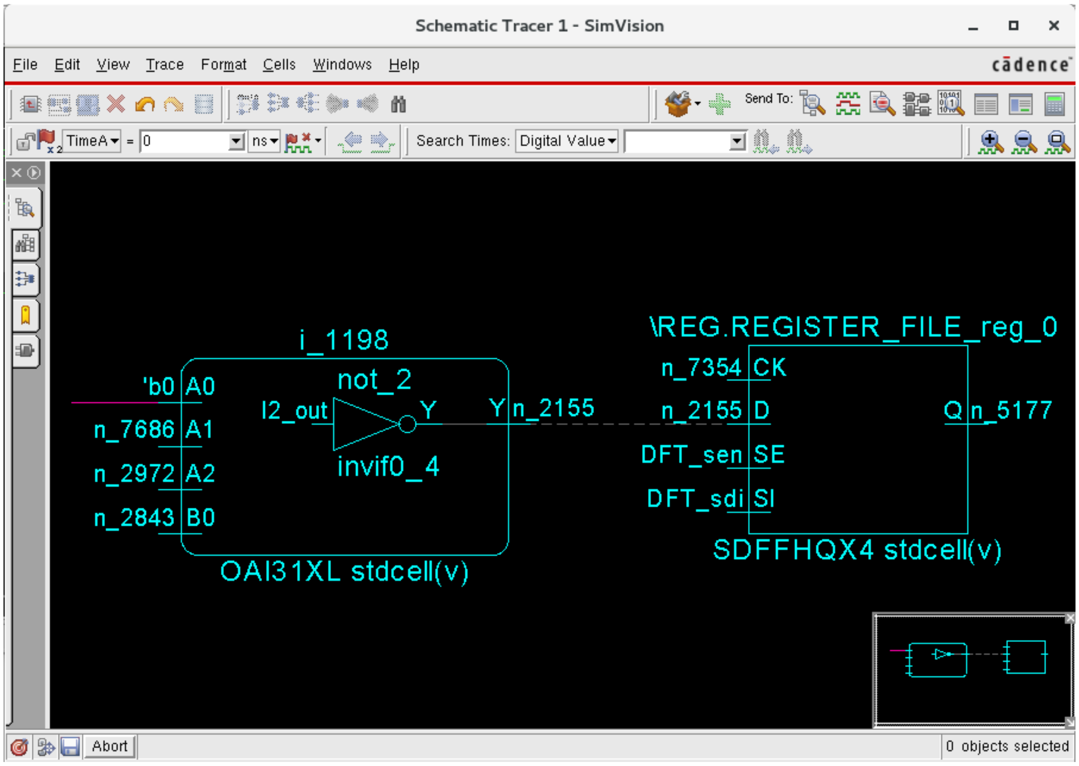
2. In the design browser window traverse the design hierarchy to locate the flop **DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_0** and click on the schematic tracer to open the schematic.



3. Once the flop is loaded in the schematic, you can back trace the logic cone by double clicking on the pins. You can add any pins in the schematic to your waveform window by selecting the pin and then clicking the waveform Icon in the bar shown below.


Schematic Tracer 1 - SimVision
File Edit View Trace Format Cells Simulation Windows Help
cādence
Send To:
TimeA = 0 ps
Search Times: Digital Value
0 + 0
\REG.REGISTER_FILE_reg_0
n_7354=x
CK
n_2155=x
D
Q
n_5177=x
DFT_sen=x
SE
DFT_sdi=x
SI
SDFFHQX4 stdcell(v)
Abort
simulator::LAB5_FULLSCAN_logic.DLX_TOP_inst.DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_0
1 object selected

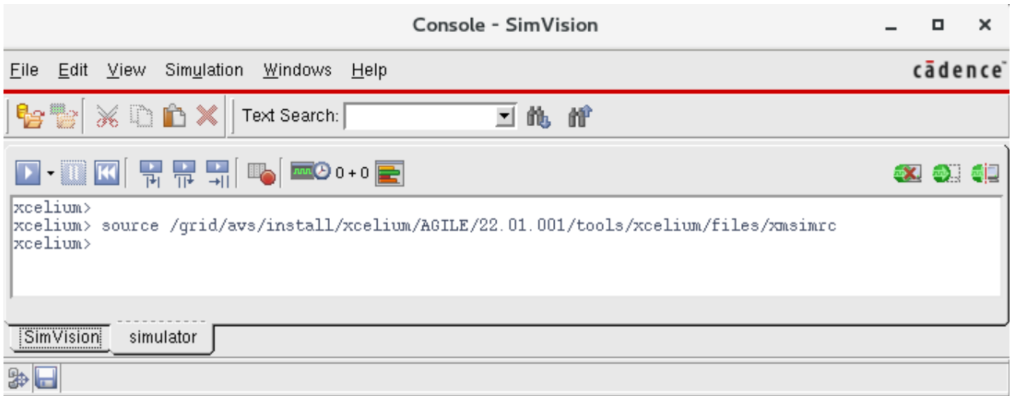
4. Back trace the pin D of the flop. You should reach the instance i_1198. Add all the pins of this cell to waveform window by clicking on the cell and then **right**-click and select **Send to Waveform Window**.



5. Go back to design browser and select the testbench name “**LAB5_FULLSCAN_logic**”. You can see the nets on the right pane. Search for “**pattern**” and send it to waveform window.

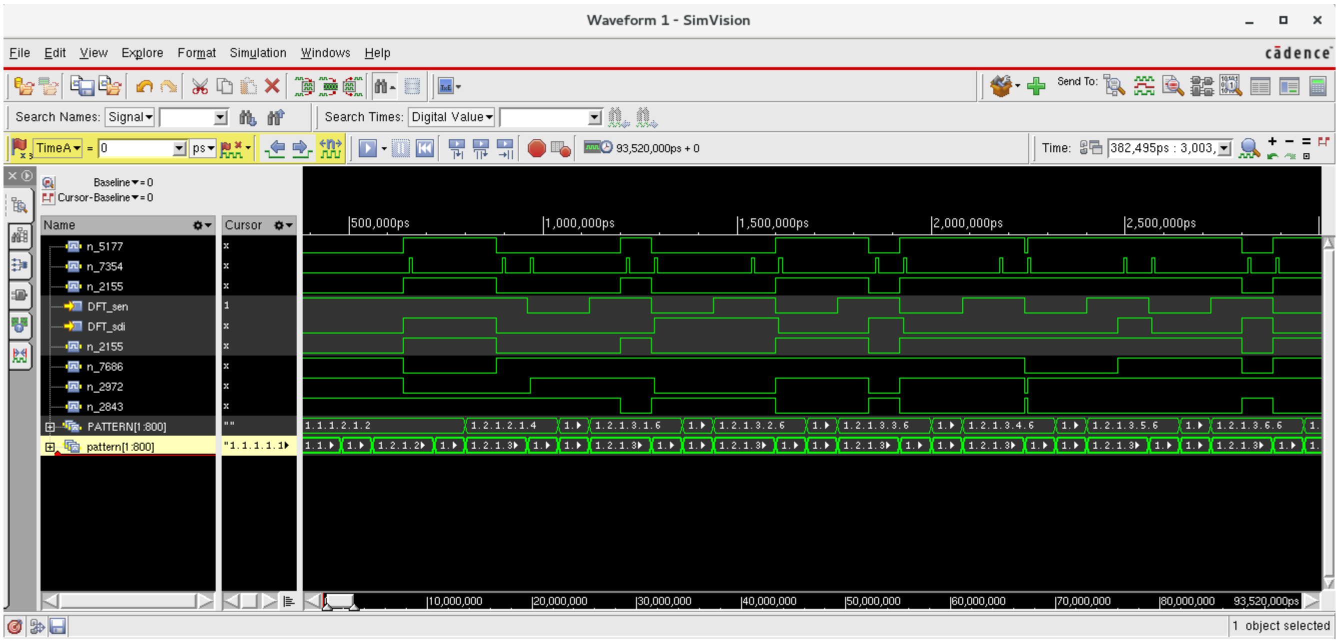
6. On the waveform window change the radix of the waveform “pattern” by **right**-clicking the waveform element under “cursor” click on Radix/Mnemonic and change from default to ASCII. This should show the pattern odometer on x axis and will help you locate the failing odometer. **1.2.1.3.1**.

7. Go back to the SimVision Console and type run at the SimVision prompt and hit return. This will simulate the testbench.

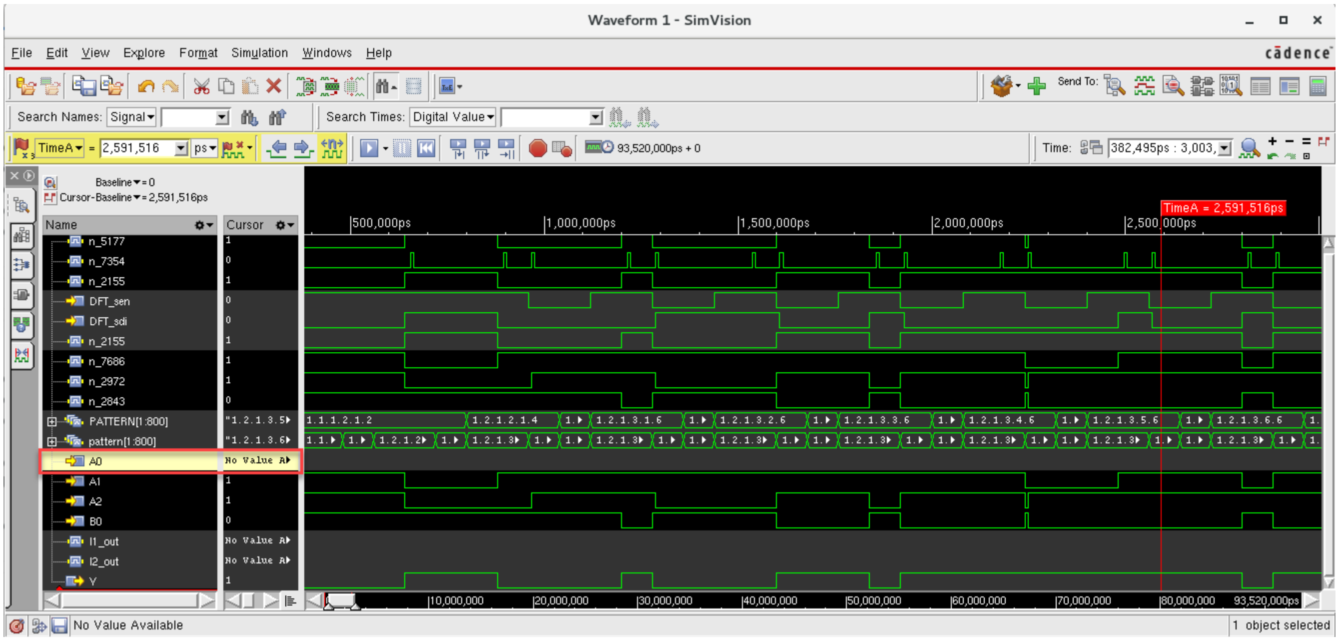


```
xcelium>
xcelium> source /grid/avs/install/xcelium/AGILE/22.01.001/tools/xcelium/files/xmsimrc
xcelium>
```

8. Go to the required odometer now you have the Good Machine and Bad machine values. Compare the two to point to the fault.



9. You should be able to see that the value on A0 of instance **DLX_CORE.THE_REG_FILE.i_1198** is always 0, while in the good machine simulation that is not the case. Back trace the pin further and you can see the pin is tied to 1’b0.



Alternate way of Simulation Debug

There is one more way of bringing up **SimVision** on the **BAD** machine simulation. In this method we allow tool to dump the values of the nodes/ports in one directory (simvision.shm) and later we can directly invoke SimVision(UI) and can reach the desired flop.

1. Open the script ./verilogsim/run_sim_bad and add the **+simvision** option to the xrun command. Your command should look like:

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

   Or you may use the already created script ./verilogsim/run_sim_bad_dump.

2. On the separate shell run ./verilogsim/run_sim_bad (or ./verilogsim/run_sim_bad_dump with +simvision option already added) this should run the simulation with dump.

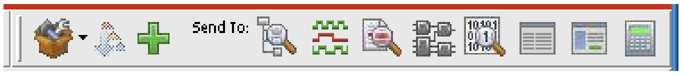
   Note: Once the simulation is over you can see a directory “simvision.shm” is added to your database.

3. Load the SimVision by entering at the command prompt:

```
simvision &
```

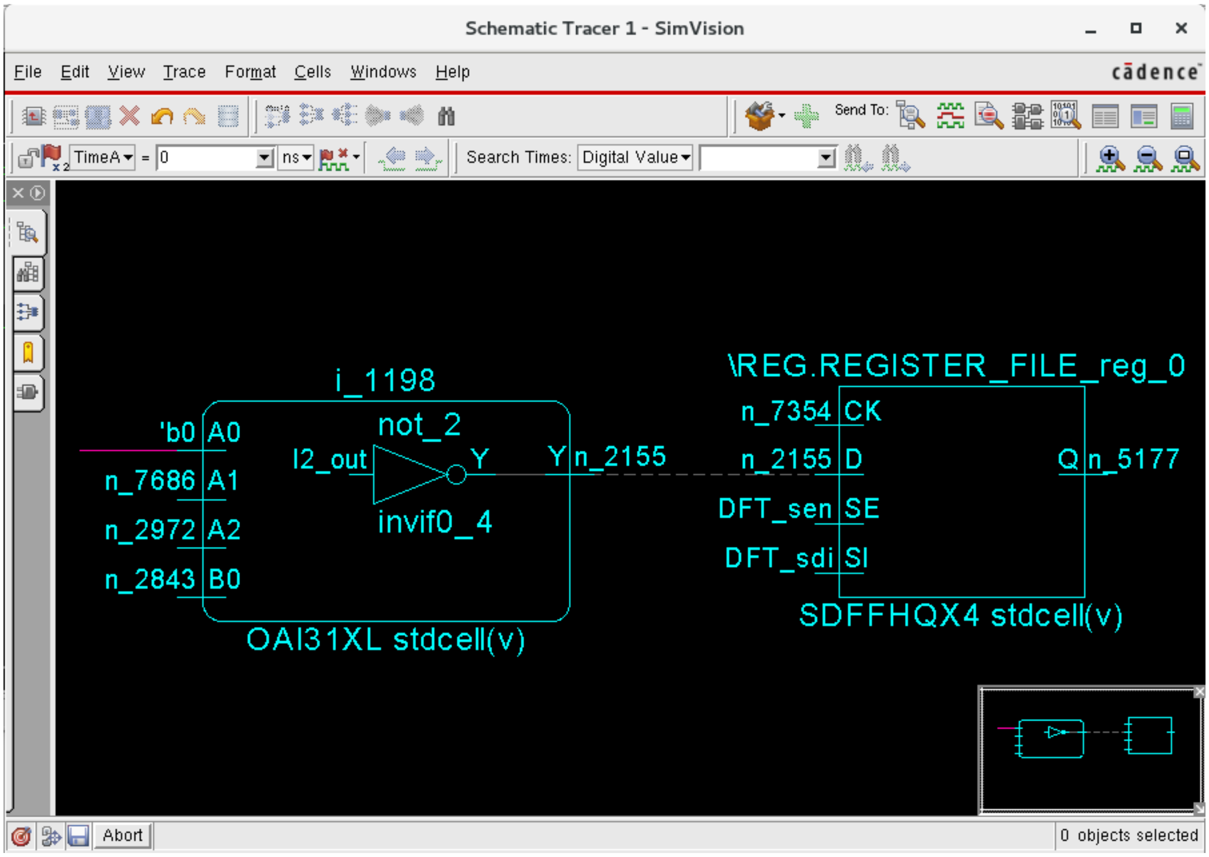
4. Open the database from **file -> open database** and select **simvision.shm** and click on **Open & Dismiss**.

5. In the design browser window traverse the design hierarchy to locate the flop DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_0 and click on the schematic tracer (8th from right) to open the schematic.



6. Once the flop is loaded in the schematic, you can back trace the logic cone by double clicking on the pins. You can add any pins in the schematic to your waveform window by selecting the pin and then clicking the waveform Icon in the bar show above.

7. Back trace the pin D of the flop. You should reach the instance **i_1198**. Add all the pins of this cell to waveform window by clicking on the cell and then **right**-click and select "**Send to Waveform Window**".



8. You should be able to see that the value on A0 of instance **DLX_CORE.THE_REG_FILE.i_1198** is always 0, while in the good machine simulation that is not the case. Backtrace the pin further and you can see the pin is tied to **1'b0**.

Hopefully, this exercise gave you an introduction to the power and flexibility of the Modus debug environment.

End of Lab

Appendix: Lab Answers

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

Build the Testmode

What is the active logic number and what does it mean?

Answer: 93.47 %

Any info on scan chains in this log file?

Answer: 1 INFO (TTM-357): There are 16 scan chains which are controllable and observable.

Do we have valid scan chains?

Answer: Yes, we have 16 valid scan chains.

Report Test Structures

How many Controllable chains do we have?

Answer: We have 7 controllable scan chains.

How many Observable chains do we have?

Answer: We have 7 observable scan chains.

Build the Fault Model

How many Static Faults are there in the entire design?

Answer: 66723

How many Static faults are active in the FULLSCAN mode?

Answer: 3663

Lab 3-2 ATPG Vector Generation

Generating ATPG Vectors

What is your resultant Fault Coverage for that Testmode?

Answer: 99 %

How about your Global Coverage?

Answer: 99.72

Writing Vectors

What is your total cycle count?

Answer: INFO (TVE-005): Created 24559 total cycles, of which 1609 are test cycles, 22950 are scan cycles, 0 are dynamic timed cycles and 0 are dynamic cycles that are not timed. [end TVE_005]

Lab 4-1 Debug Broken Scan Chains with Modus GUI

Why is Scan Out DLX_CHIPTOP_DATA[17] chain broken?

Answer:

- Feeding FF is not being clocked by a scan clock
- Clock is not connected.
- Flip/Flop block DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_934
- Unconnected clock at net DLX_CORE.THE_REG_FILE.__n1

Why is Scan Out DLX_CHIPTOP_DATA[18] chain broken?

Answer:

- Clear on feeding Flip/Flop is not controlled.
- Pin DLX_CORE.THE_CONTROLLER.TEMP_ENABLE_N52_reg.RN

Why is Scan Out DLX_CHIPTOP_DATA[19] chain broken?

Answer:

- Scan enable pin of scan flop is not connected.
- Pin DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.SE

Why is Scan Out DLX_CHIPTOP_DATA[20] chain broken?

Answer:

- Observable scan bit 9 Flip/Flop DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_719 receives it scan data from Observable scan bit 6.
- Tracing backward identifies a loop in the scan chain data that does not allow controllable scan data to enter in.
- Net DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_716 should not be driving Pin DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_719.SI

Why is Scan Out DLX_CHIPTOP_DATA[21] chain broken?

Answer:

- Scan Out is not driven by any scan FF
- Net SCAN_OUT[5] is not driven

Why is Scan Out DLX_CHIPTOP_DATA[23] chain broken?

Answer:

- The driving Flip/Flop DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_474 does not have a controllable clock.
- Tracing the clock on the above FF finds an AND gate DLX_CORE.THE_REG_FILE.i65129. The clock is not enabled through the AND gate. The B pin needs to be a 1 during the scan state. Instead it is driven by an internal FF.

Lab 4-2 Debug Broken Scan Chain with Tcl Interface

Debugging observe scan chain 1

What does the trace results from above tell us?

Answer: No further pin is connected.

Does it match what you found in the GUI debug section?

Answer: Yes, it is matching with GUI debug section.

Lab 5-1 Analyzing Initialization Sequences Using Modus GUI

Analyzing Sequences

Where is the Blue Mark arrow?

Answer: 1.7.1

Is the proper transition highlighted in the Watch window?

Answer: Yes

End of Lab