ATPG Flow with Modus Software Solution

Course Version 22

Lab Manual

Printed in the United States of America.

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

Table of Contents

ATPG Flow with Modus DFT Softwa

Module 1: About This Co

No Labs for This Module

Module 2: Introduction t
Software Solu

Lab 2-1 Introduction to Modus ATPG Flow Dat Invoke the Modus Tool

Objective: To bring up and execute Modus tool and lea structure.

This lab is intended to introduce the invocation of Modus DFT Soft directory structure. Modus offers a Tcl based command line interfac the GUI.

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

```
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_PA
```

You can verify the availability of the tools in environment at the co

- `which modus`
- `which xrun`

Bring up and Execute Moduspwd

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 a

To bring up the new Cadence Help systems (cdnshelp), just type in 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
```

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
```

Module 3: The ATPG Flo

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

Objective: **To learn the Modus ATPG Flow and get famil Interface.**

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

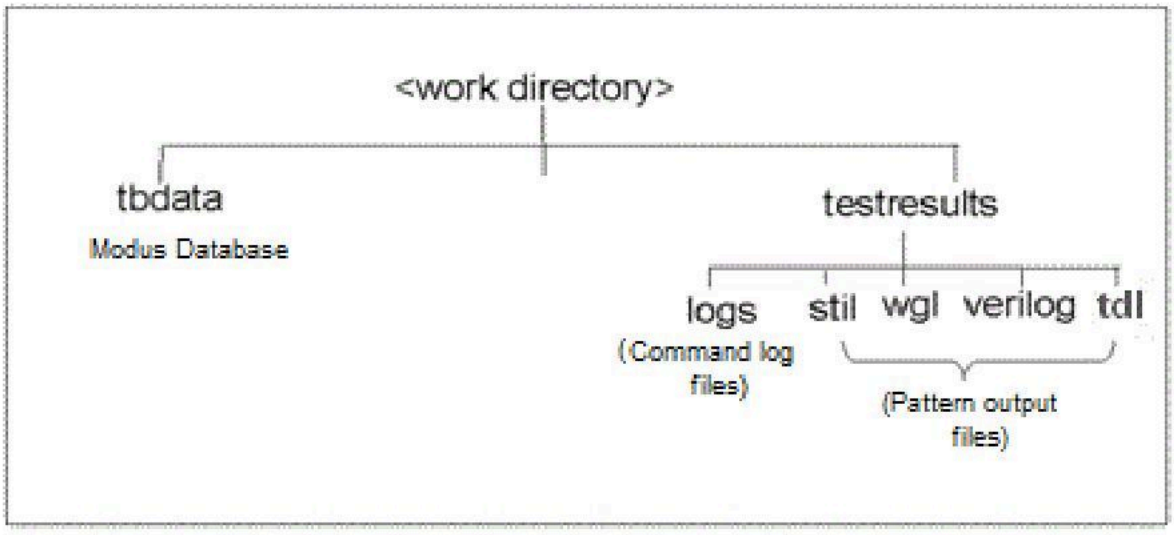
Invoke the Modus

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

Note: Once you invoke Modus you should see Modus Tcl p

Setup Modus Workdir

Modus stores its database in a work directory and the following figu created by Modus for a work directory.



On your Modus command line run the following command.

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

The command will read in your libraries and netlist to compile a Modus build_model processing you should find the fault model statistics summ 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.
```

You should also see a summary of any warnings/errors that tool enc
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 exter
 usage in the design. Cell contents file: '/home/shubham8/work/JumpStart
      1 WARNING (TEI-275): Mixed signal strengths not supported on line

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

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

Users are required to review and analyze any Errors/Warnings flagg

What are the TEI-110 messages?

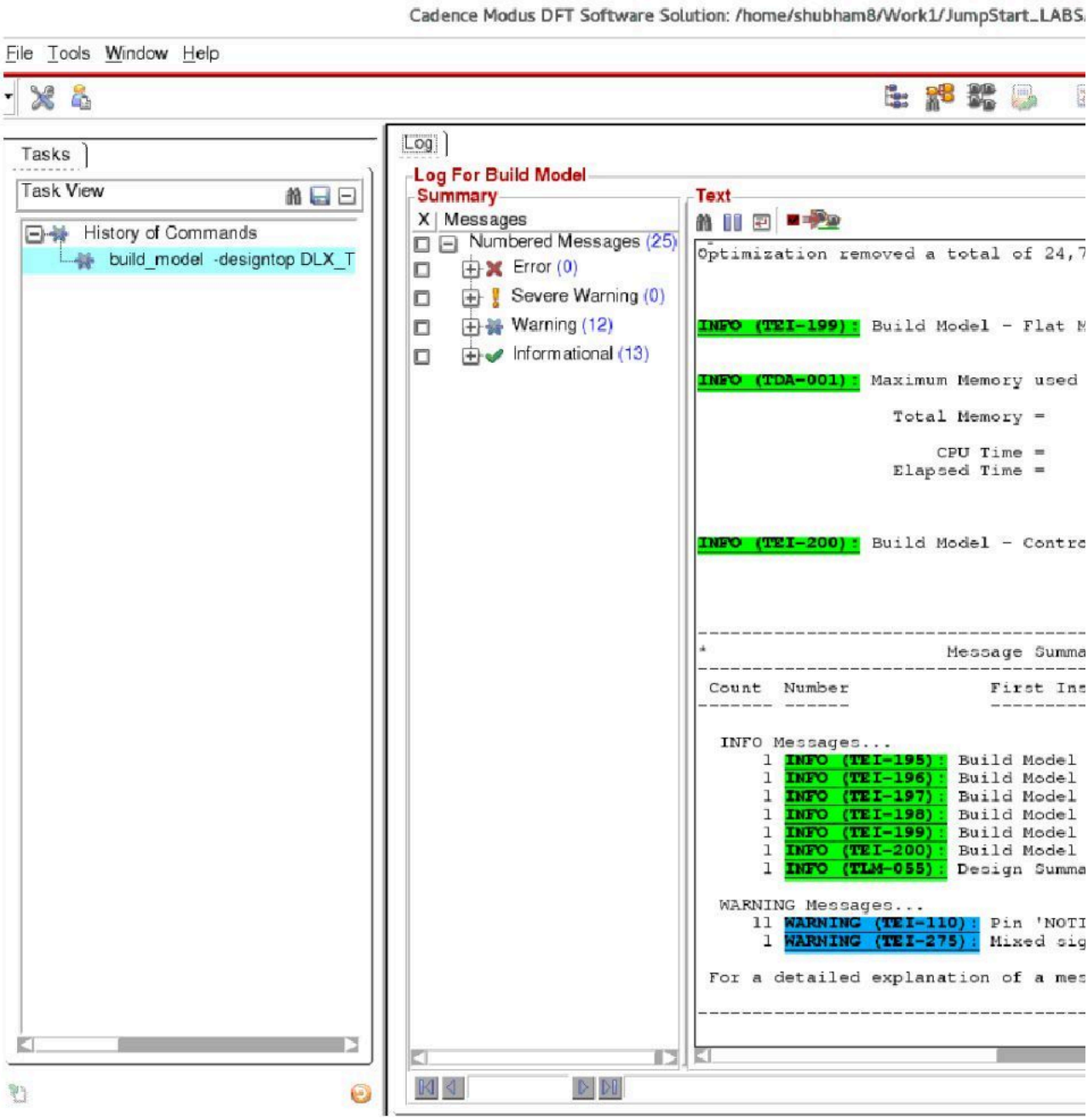
Review the Errors and Warnings Inside Modus GUI

Modus offers extensive GUI debugging capabilities. Let's take a fir
Errors/Warnings inside GUI. From your Modus command line, invo
command.

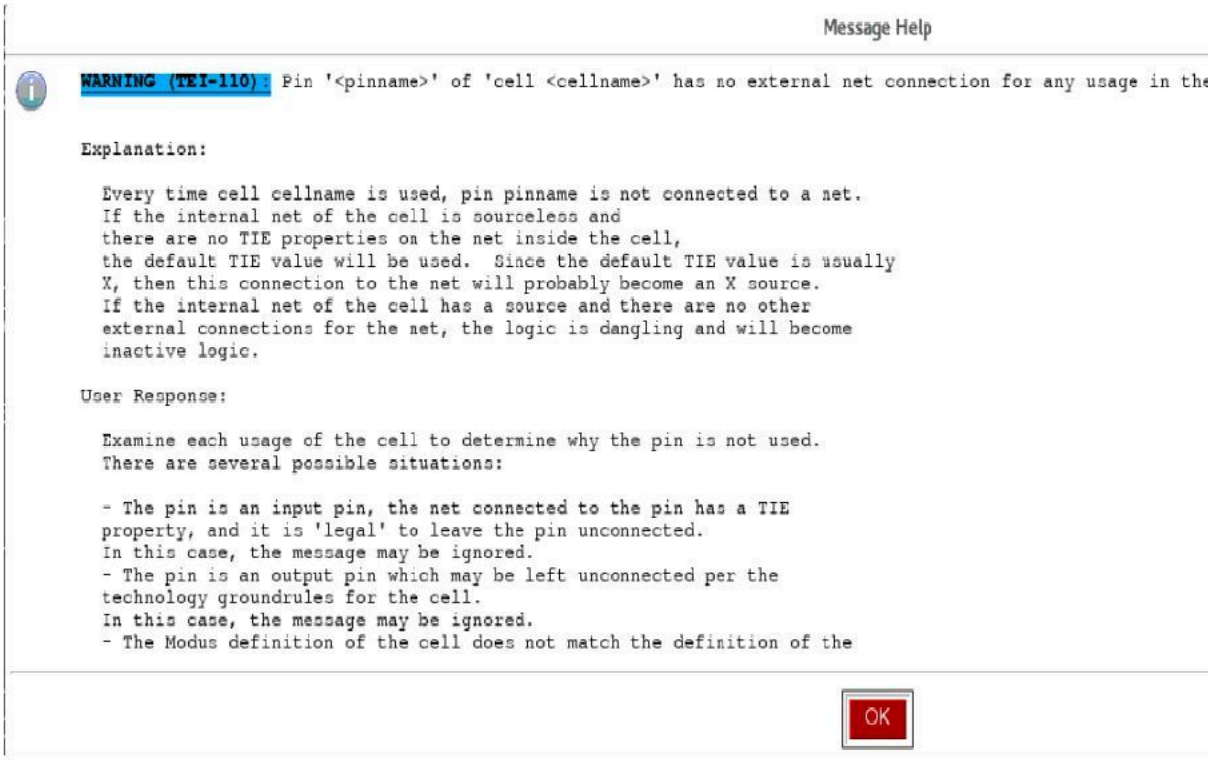
You must explicitly setup another "set_db workdir. "This is necessa
build_model because, this is a new tbdata database. Setting the com
build_model will not retain this value. You will get the following er

```
ERROR (TUI-809): gui_open invocation requires an
```

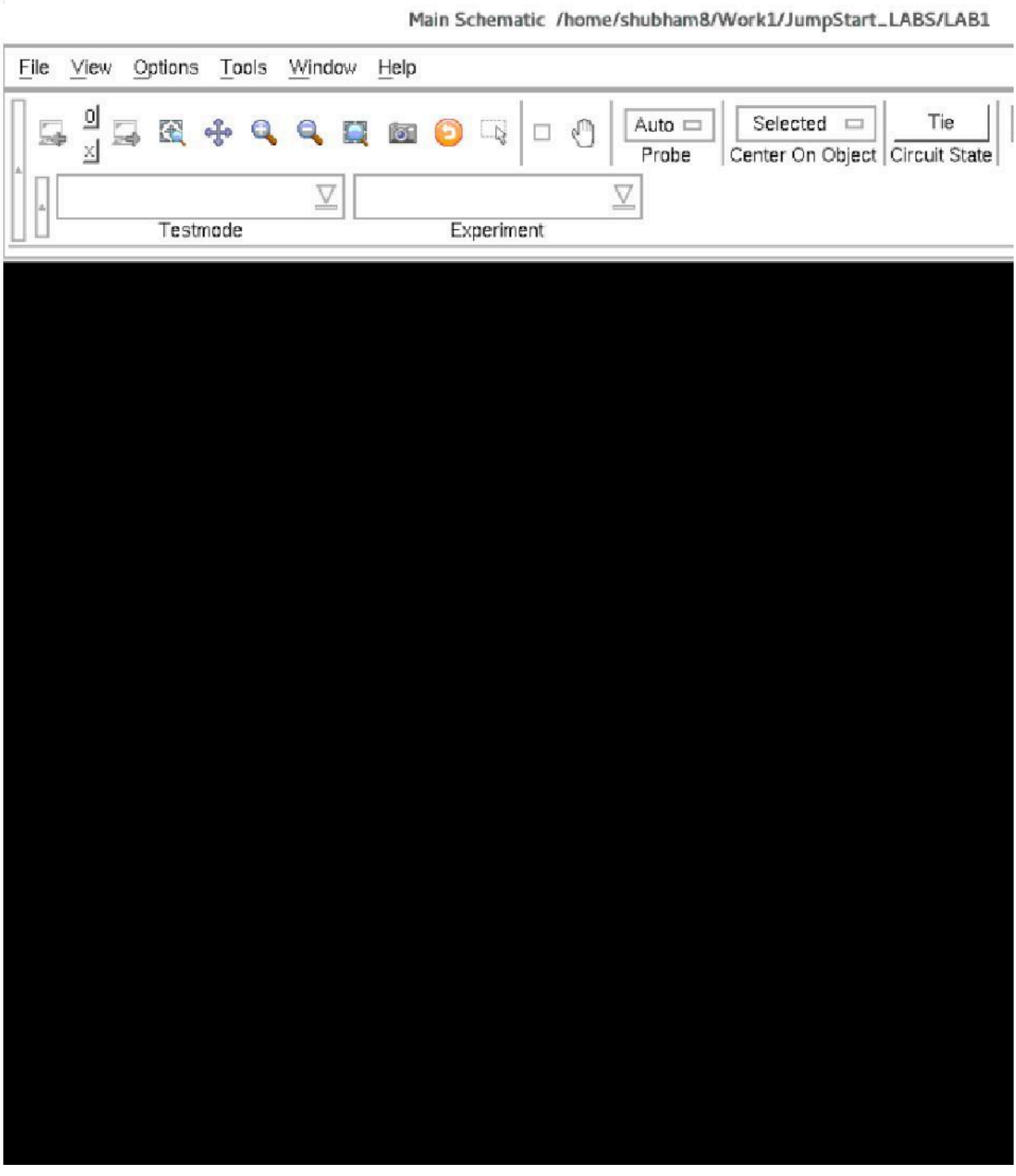
You should now see the GUI window with a TASKs pane on the left. C
that was run. If you click on one of the Warnings in the Log file output
information is shown in the Log.



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



Let's try out the Hyperlinking. In the Log Tab, select the "Toggle Highl
will also pop open the schematic window. You can close this window. I
rule checking.



- **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 fau
    - This means during shift it will be set to its de-asserte 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 al few examples to get you started:

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

Once the command is executed, look for any Severe Warning messages

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, 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
      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.  Glob
      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 <messa

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

Report Test Structures

Before we run full design-rule checking, we can see if our testmode is s 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 fo

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
```

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 S
    Load Point Pin Index/Name  44/port:DLX_CH
    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 S
    Unload Point Pin Index/Name  51/port:DLX_
    Unload Point In Phase with Load Point
    Length: 85
    Scan Section: Scan_Section_Sequence
```

Note: The Bit Lengths are different. When the log file print they will also not be listed together. This is because t they are supposed to belong to the same scan chain. in LAB2 when we have broken chains.

Note: When you get into compression structures the format will have multiple Control Reg points feeding to mul points.

Once the command is executed look for Informational message **TSV-37** complete scan chains were found. The summary of verify_test_structure

```
--------------------------------------------------------------------------------
*                        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
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 en
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
      1 INFO (TSV-568): There are 16 observable scan chains feeding to Scan Out
      1 INFO (TSV-569): There are 0 controllable scan chains fed by on-product P
      1 INFO (TSV-570): There are 0 observable scan chains feeding to on-product

      1 INFO (TSV-900): verify_test_structures processing has started Fri Nov 17
      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 during the same scan cycle as scan chain flop or latch block DLX_CORE.THE
dff_primitive. The upstream and downstream flops or latches are changing on the
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
TDA-009

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

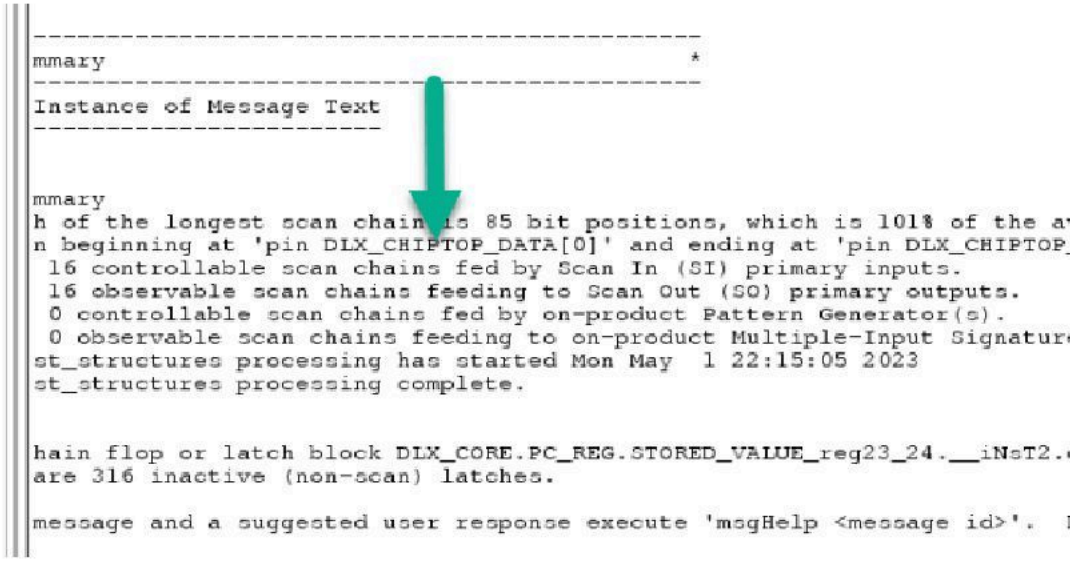
Note: Look for any Severe Warning or Error messages in the M section of the log file. If these occur, you need to verify t them and handle each one appropriately.

Do you have any TSV-390 messages in the verify_test_structures summ listing of all the floating/non-scan flops?

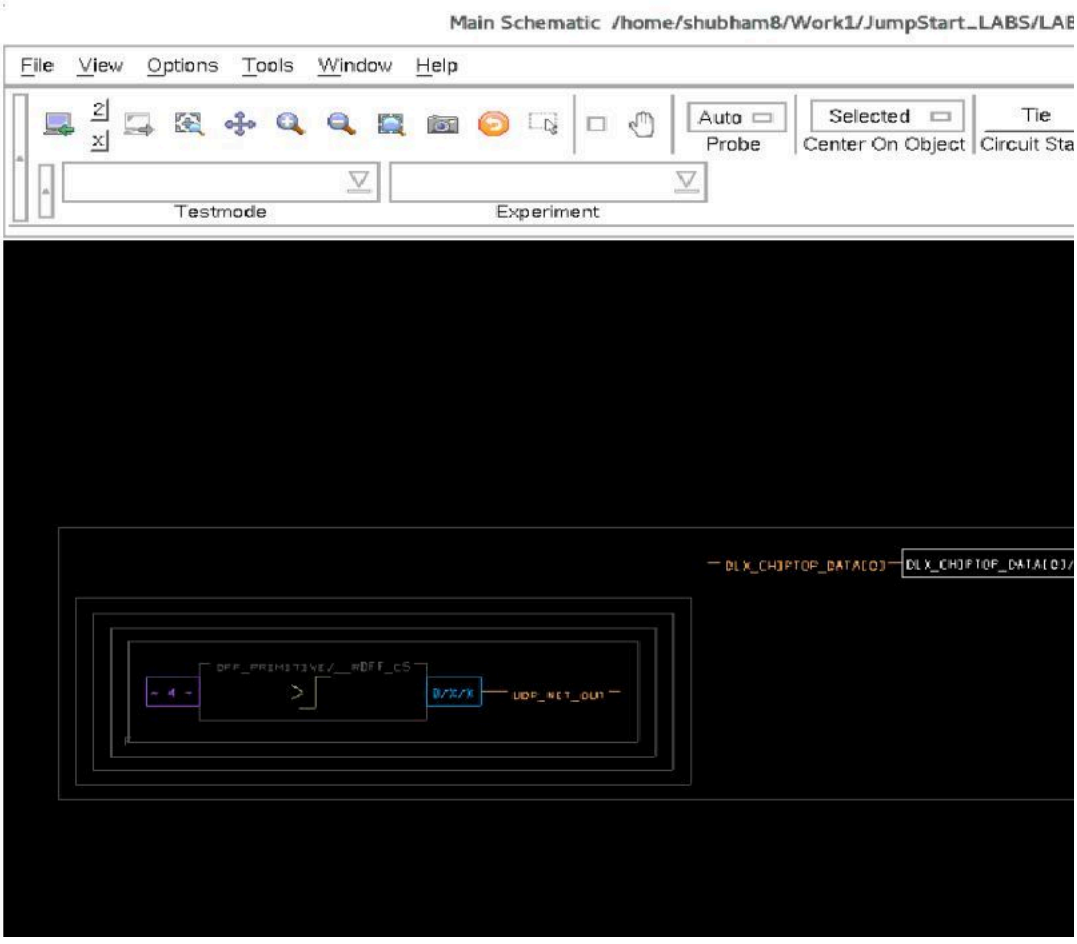
To get the list of floating/inactive latches, use the command:

```
report_test_structures -testmode FULLSCAN -reportre
```

This command will report all the inactive latches.



5. Click on the hyperlink. This will pop up the schematic view in the circuit.



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/
     1 INFO (TFM-103): Creating faultStatus file /home/shubham8/work/JumpStart_LABS/LAB1
     1 INFO (TFM-109): Build Fault Model has completed with highest level severity messa
     1 INFO (TFM-704): Maximum Global Test Coverage Statistics:

 For a detailed explanation of a message and a suggested user response execute 'msgHelp <
TDA-009

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

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

Running **build_faultmodel** will only include static fault modeling by d faults also, user can turn on dynamic fault by using the **-includedynam**

By default, the faults are only marked at the cell boundary as per accept also tell the tool to apply faults inside the cell boundary. For this lab yo apply the faults on the cell boundary. Observe the build_faultmodel log 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 generat close the GUI by clicking the **X** in the upper right-hand side of the GUI **GUI and Console**, or use the command:

Lab 3-2 Generating ATPG Vector

Objective: To perform ATPG vector generation, and wri WGL patterns.

In this lab, you will generate the ATPG vectors and write out the Ve

Invoke Modus

In the same directory **JumpStart_LABS/LAB1** invoke Modus by u

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

Once the Modus is invoked, you can see that the session imports the created in LAB1, you can continue where you left off in LAB1. You

```
INFO (TUI-300): Detected the presence of Modus database in present working di
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 -experimen
```

Once the above command is run, take a look at the log file and mak Warnings. Look at the initial coverage section you should see that tl

You can take a quick look at some other option with create_logic_tests purpose. Some options provide instructions to the tool on how to handle Examples:

a. **-contentionremove**: This option controls the way ATPG ha contentions. If you select hard, the tool will throw out any ve contention but will keep patterns with soft contention.

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

For an ATPG run it is important to know the final coverage and the nun Look for the information in the log file. Find the answers to following q

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 Fa vectors. if you feel the vector set is good and meets the required coverag you can commit these tests. Once the tests are committed no other tests, to go after the faults tested with these set of vectors. To commit test vec

```
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):
```

Look at the initial coverage section in the log file, you should see th
coverage.

Testmode and Global statistics prior to commit_tests

	ATCov		Testmode Faults					
	Testmode	Global	Total	Tested	Possibly	Redundant	Untested	Total
Total Static	0.00	0.00	63060	0	0	0	63060	66723
Collapsed Static	0.00	0.00	41190	0	0	0	41190	43382
PI Static	0.00	0.00	81	0	0	0	81	88
PO Static	0.00	0.00	156	0	0	0	156	156
Total Dynamic	0.00	0.00	59712	0	0	0	59712	66644
Collapsed Dynamic	0.00	0.00	49332	0	0	0	49332	54960
PI Dynamic	0.00	0.00	72	0	0	0	72	88
PO Dynamic	0.00	0.00	156	0	0	0	156	156
Parametric								
IDDq	0.00	0.00	66723	0			66723	66723
Path	0.00	0.00	0	0			0	0

Writing Vectors

This steps writes out the vectors that can be used for manufacturing
and verification. Typically, the vector formats that are used for man
TDL. The format used for cad simulation and verification is Verilog
which are committed or the patterns which are not committed. Whil
experiment name using the option **–inexperiment** to write the unco
you have committed the vectors, hence you will not use –inexperim

1. Write out WGL Patterns.

   Use the following command and write out the patterns in W

   ```
   write_vectors -testmode FULLSCAN -language wg
   ```

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

   What is your Total Cycle Count?

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

write_vectors offers two scan formats of patterns with Verilog. A seria
patterns serially on the design using the scan inputs is representative of
manufacturing tests. A parallel load is also available, which directly for
flops. This saves significant simulation time and is achieved by adding
command. The above command writes out the Verilog vectors in serial
default.

Exiting the Modus Software

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

```
exit
```

Module 4: Debug Scan C
GUI and Tcl In

Lab 4-1 Debugging Broken Scan Chains with Mo

Objective: To debug broken scan chains violations found i report using Modus circuit display.

In this lab, you will debug the design rule violations found in the verify circuit display.

Invoke Modus Software

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

   ```
   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 b

   ```
   source run_modus.tcl
   ```

   Note: This is a command line file that executes build_model, b report_test_structures and verify_test_structures. The sa Wait for the script to finish. This will create your test m **verify_test_structures** is also run towards the end of th log files of each command separately in the ./testresults/

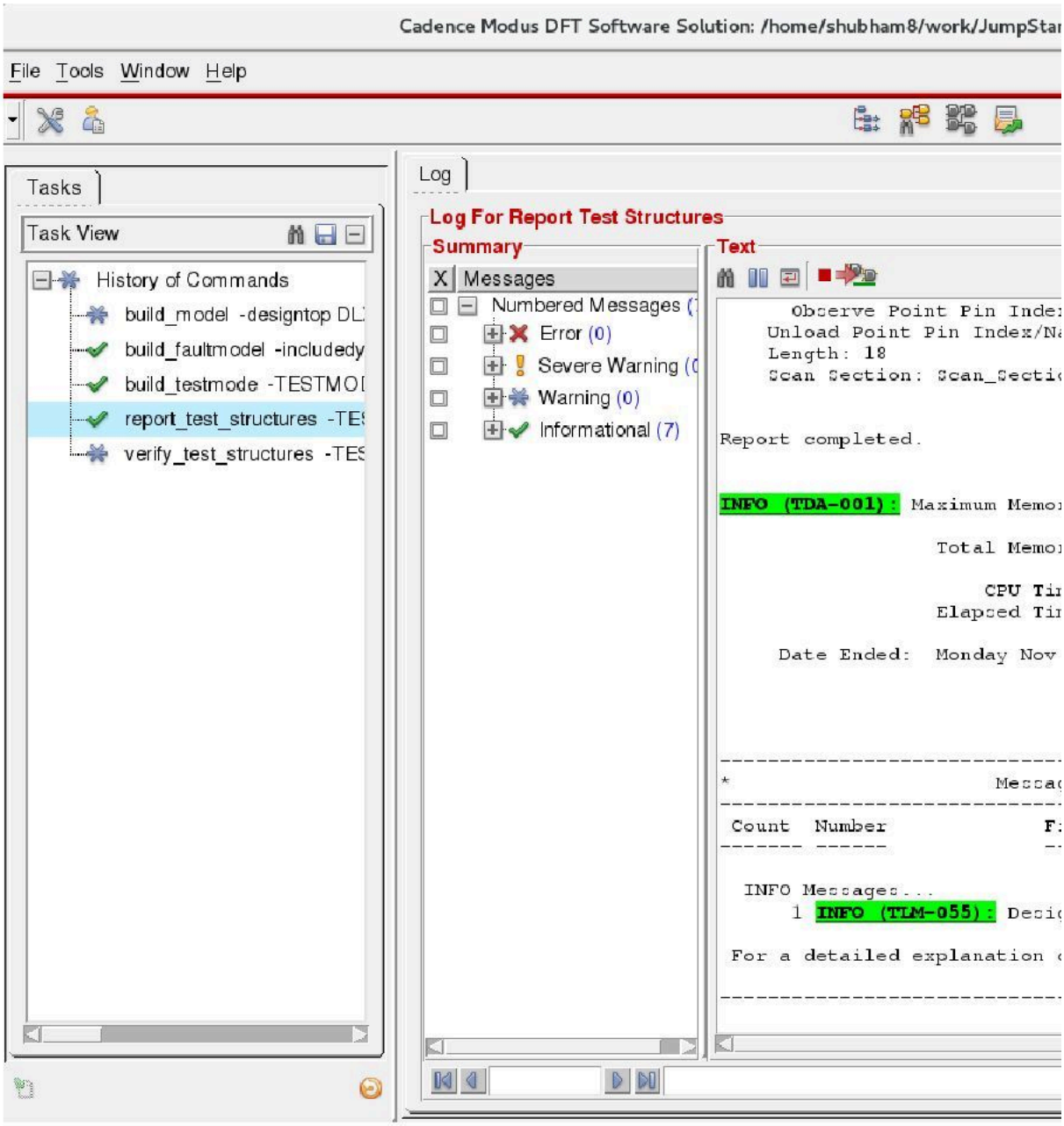
   You should now review the outcome of verify_test_structur 384 and several TSV-40X (for exampleTSV-401 TSV-402) following questions:

   Are there any broken scan chains?

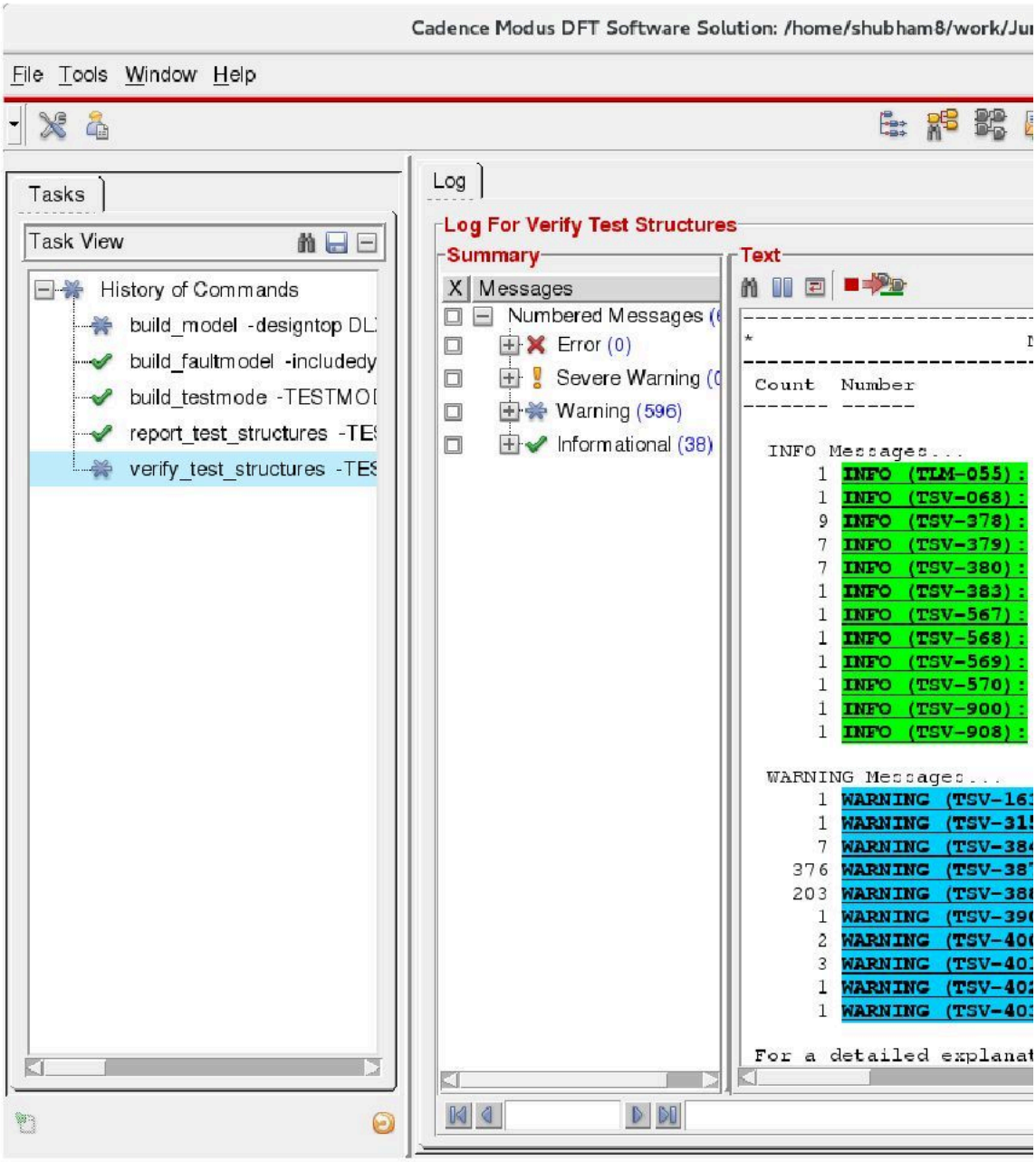
   How many broken scan chains are there?

The GUI window should come up and the Tasks pane should show

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



4. Click on the **verify_test_structures** run and review the Error and W

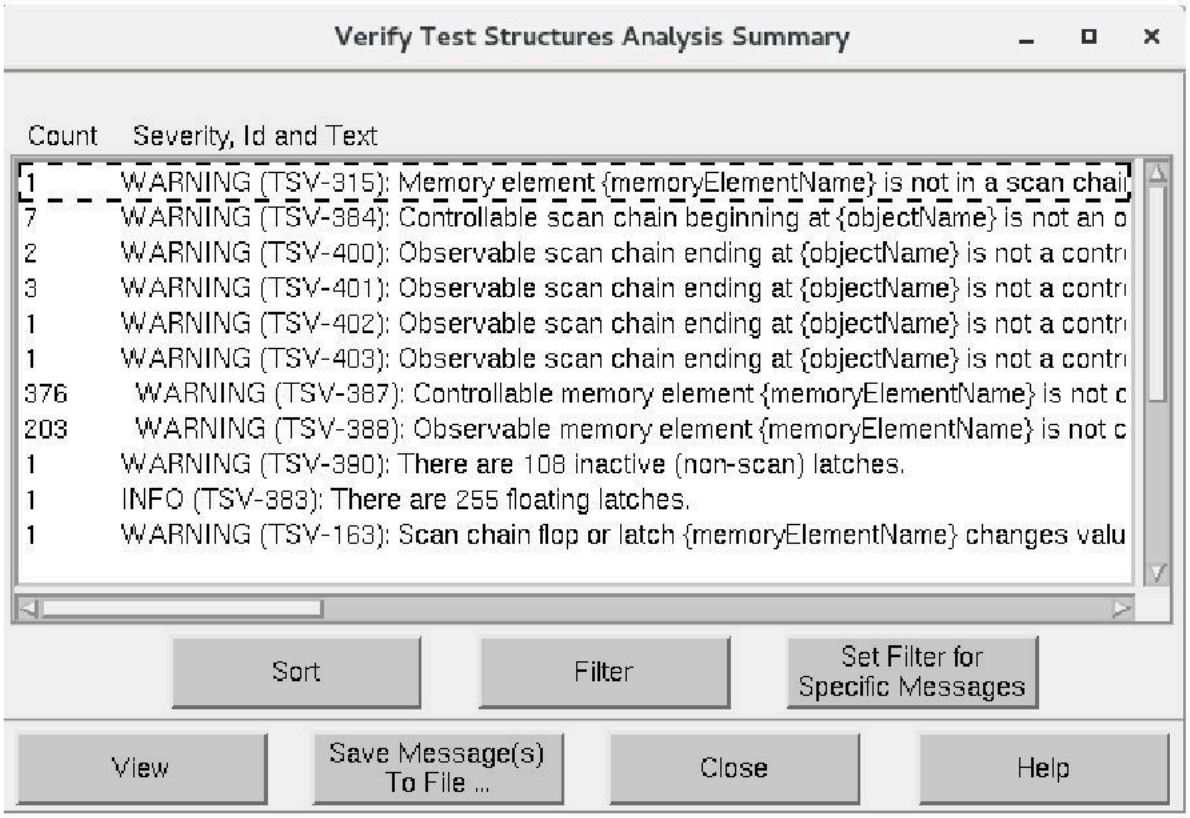


Setting Analysis Context

Before going to debug the broken scan chain, we need to set the analysi
respective broken scan chains.

1. You can see what the context is by clicking on the **"Arrow/Pen**

3. Once the context is set click the **view messages icon** on the Ana
This will open the new messages window which contains the summ

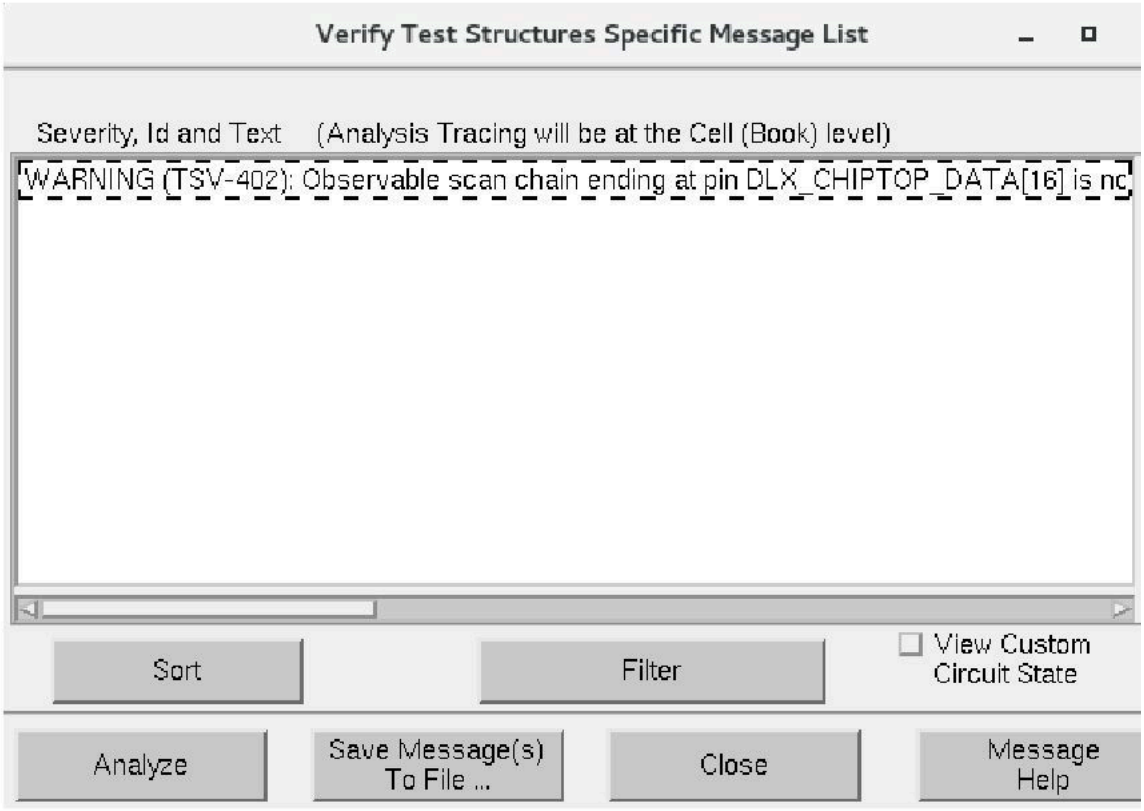


4. Look for **TSV-384** and several **TSV-40X (for exampleTSV-401 T**
Verify Test Structures Analysis Summary window, these messages
chains (number of observable scan chains that are not controllable a
chains which are not observable).

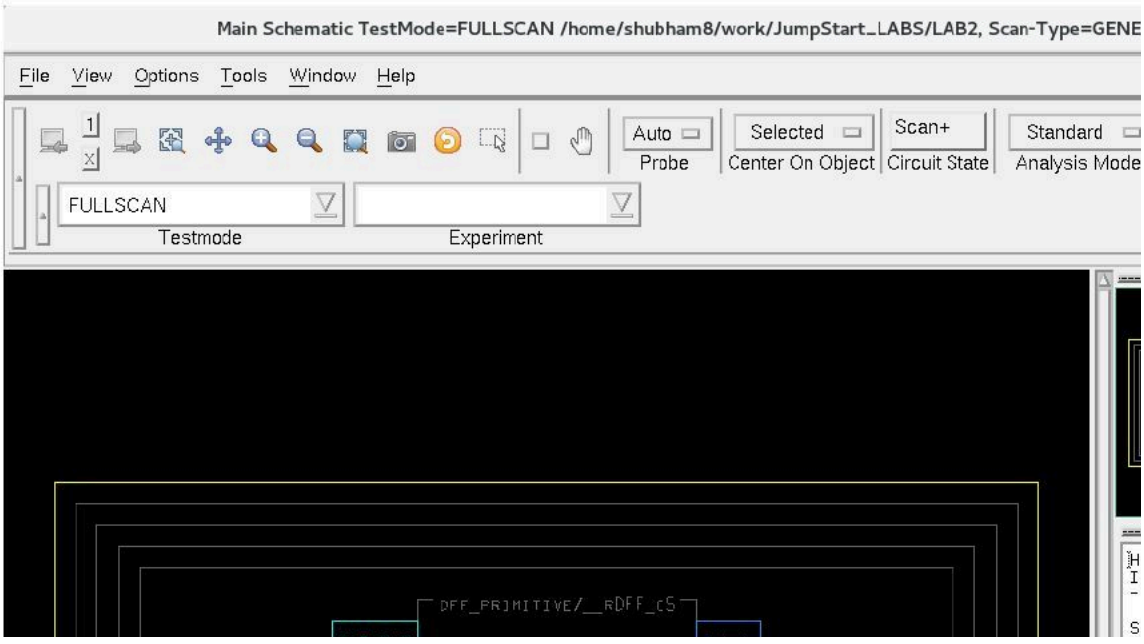
   Note: Generally, it is easier to debug the broken observable chains
   controllable chains as there is usually less choices in logic w
   comparison to tracing forward.

Going forward in this lab we will debug the Observable scan chains which

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



6. Select the message (DLX_CHIPTOP_DATA[16]) and click An schematic window on the top and will navigate to the last good analysis.



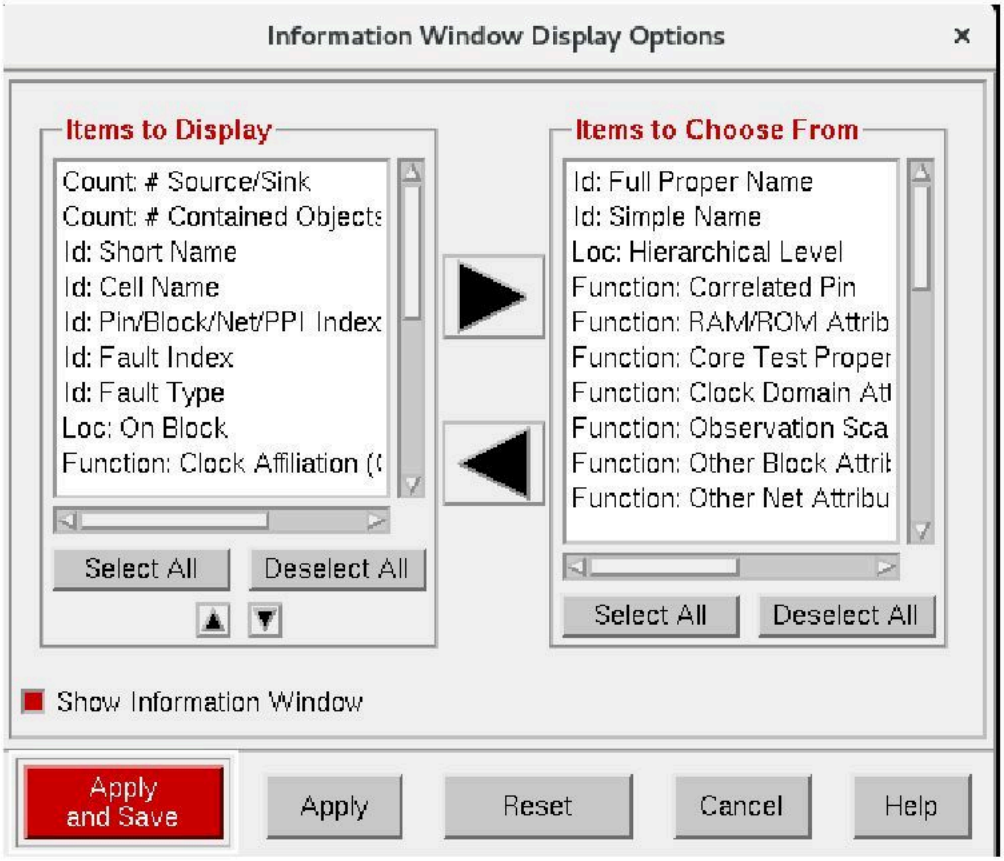
Customizing Schematic Window

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

1. **Block Information Window**

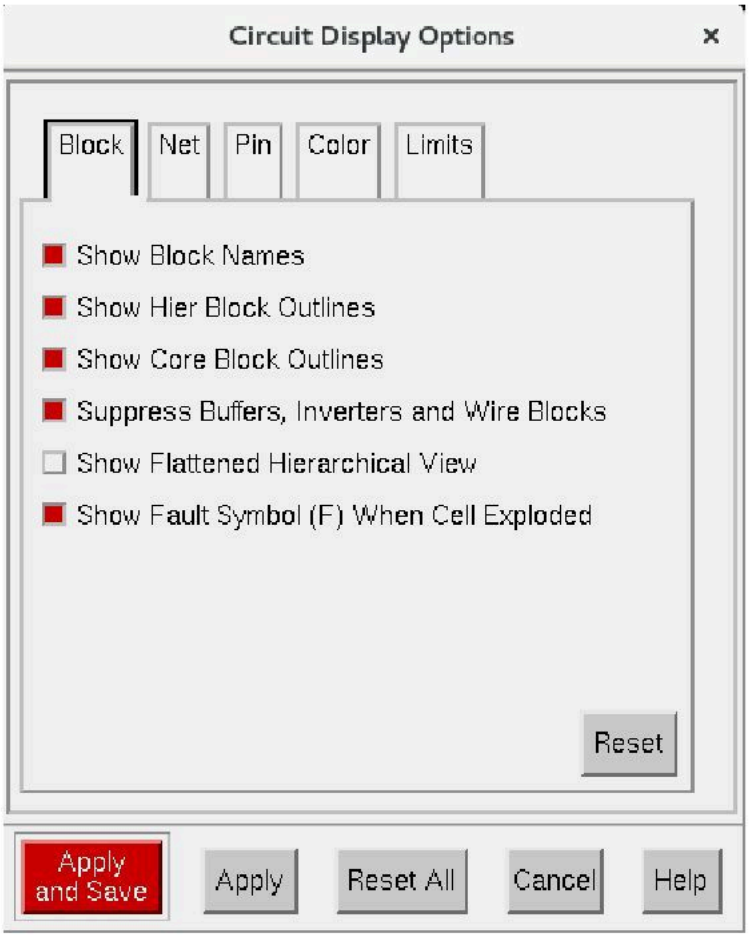
   First let's customize what we see in the **Block Information** win hand side.

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



   b. The right side has the items to add. Recommended Items to Add

```
Function: Scan Path Data : It shows whether pins ar
Function: Flop/Latch Scan Data : It Shows the flop
```



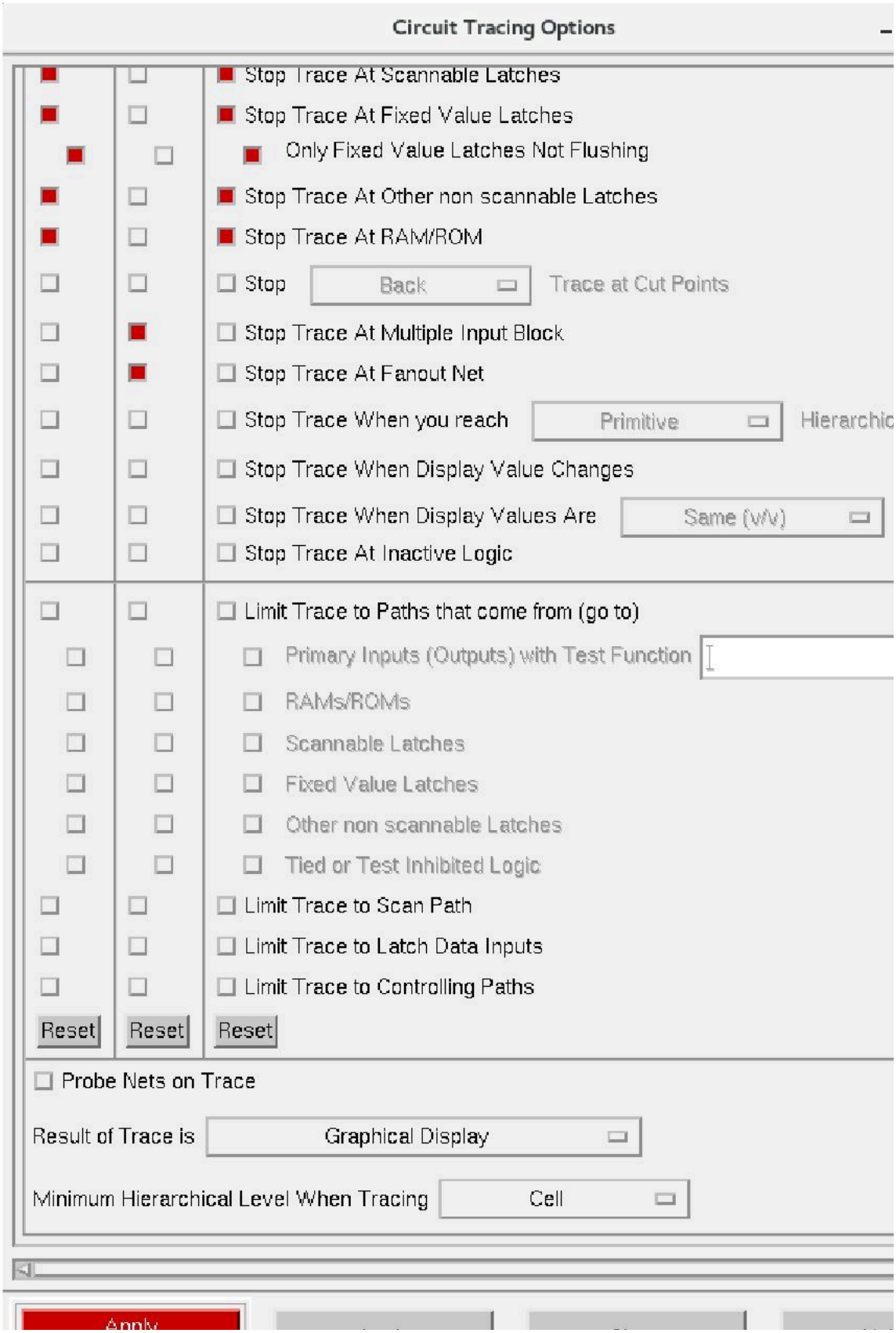
b. Click on Pin Tab and select on Show All Pins: This will sho instead of collapsing them.

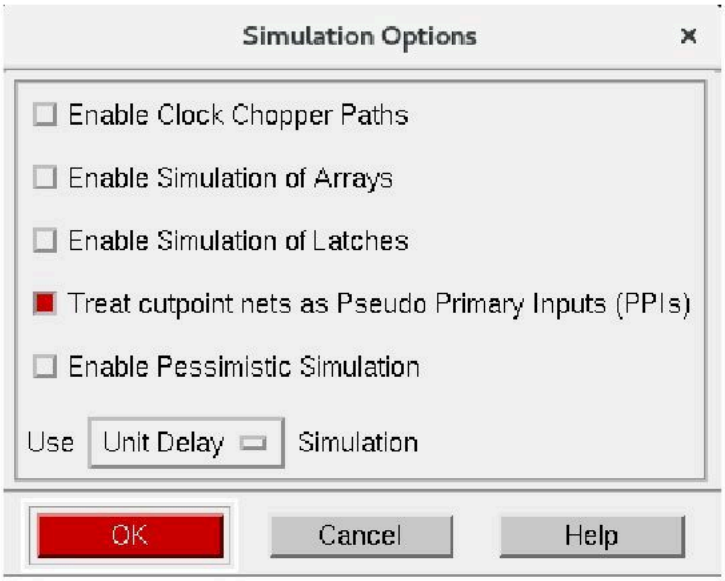
c. Click "**Apply** and **Save**" to apply.

3. **Circuit Tracing Window**

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

a. From the Pulldown menu **Options -> Circuit Tracing…** Sc Tracing window.





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

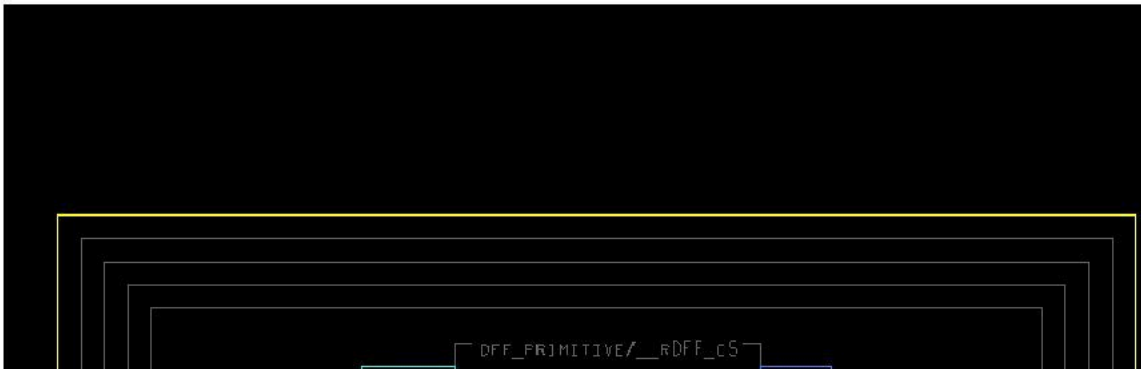
c. Select the option and click **OK**.

Interact with Schematic Display Window

In this section, we will understand some features of the Schematic Circu

1. **Schematic Hierarchies**

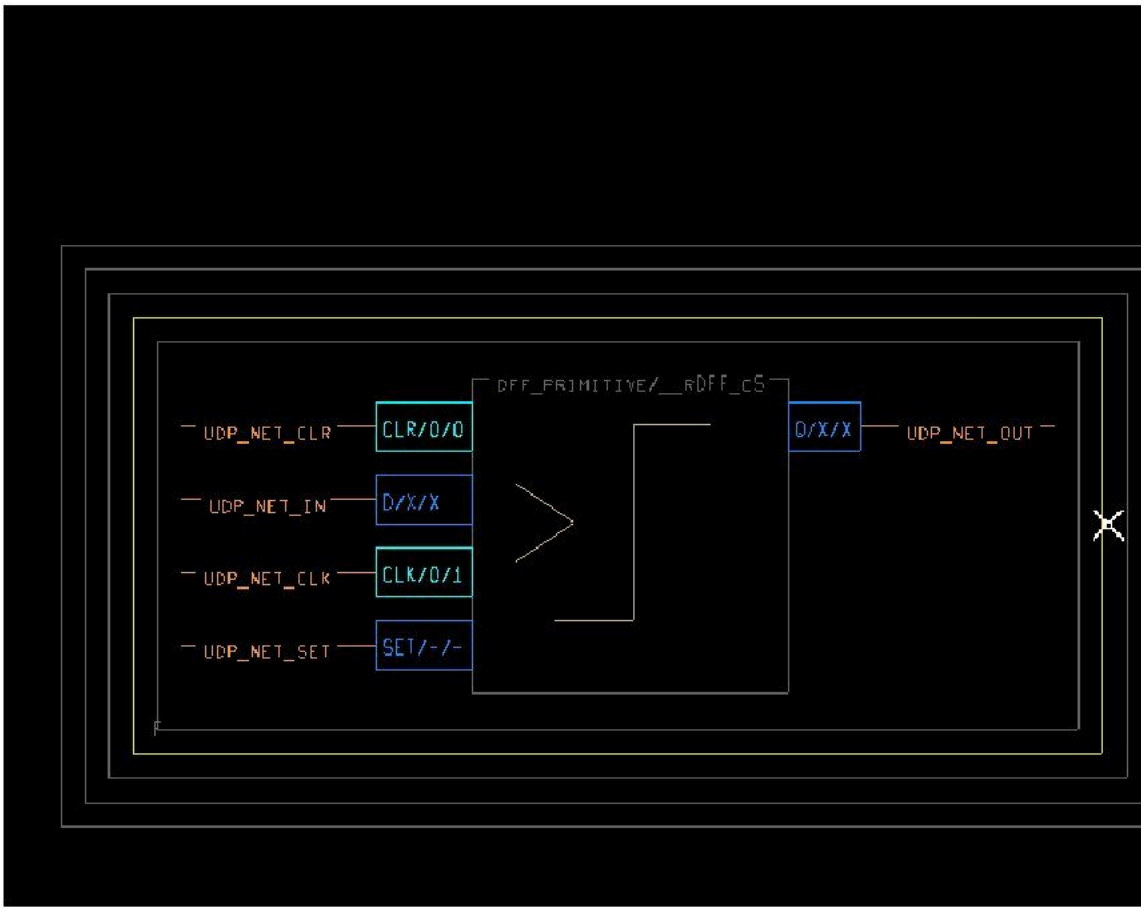
   You will see several white rectangles around the cell. Each of th hierarchy. Mouse over each and see what changes in the Block I



2. Imploding and Exploding a Hierarchy

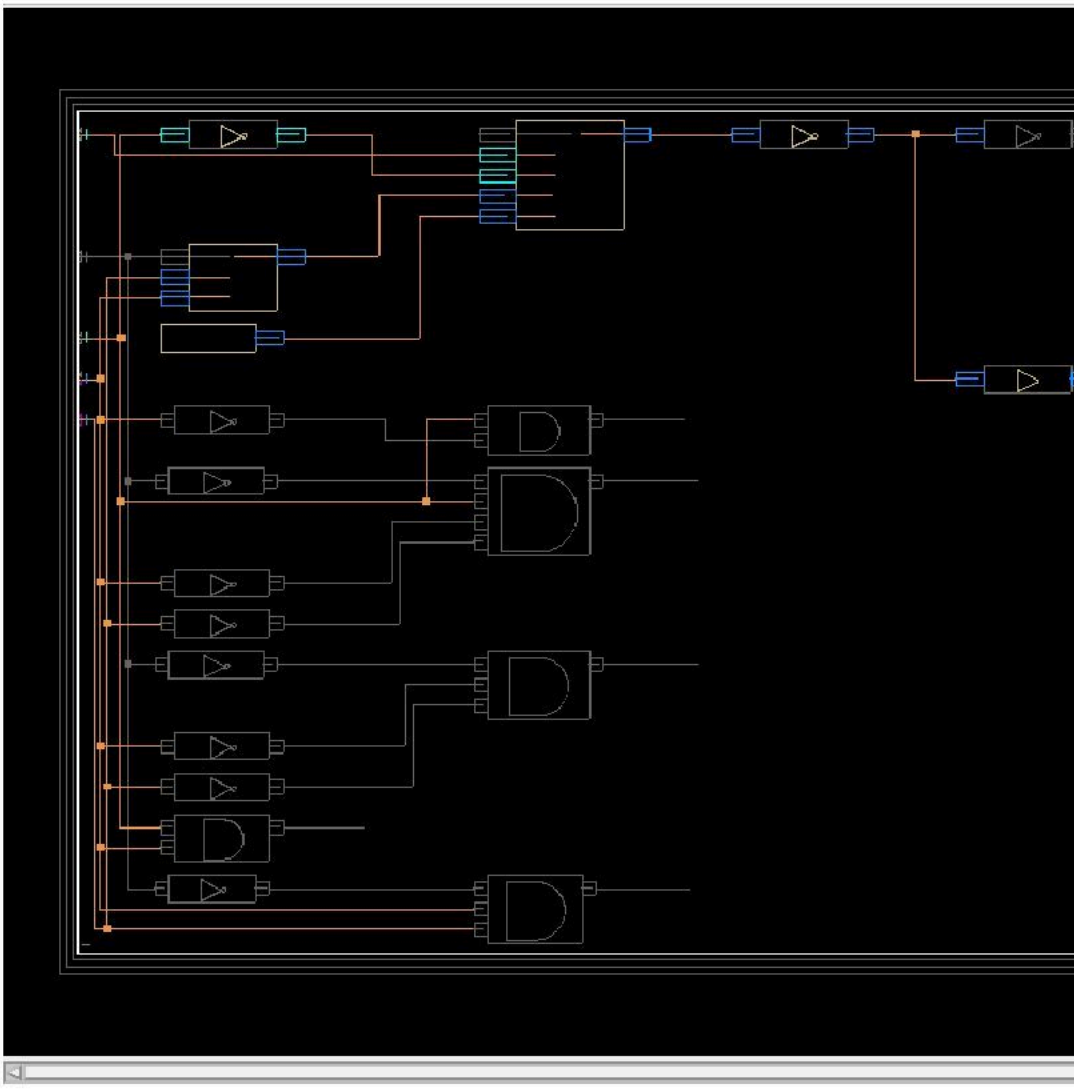
To implode and explode a hierarchy, use the below steps:

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



b. **Right**-click on the rectangle and select **Implode**: This will colla hierarchy selected and you should now see the library cell SDFI

c. Select the **SDFFRX1 cell** -> **right**-click -> Click **Explode**: I whole scan flop cell.

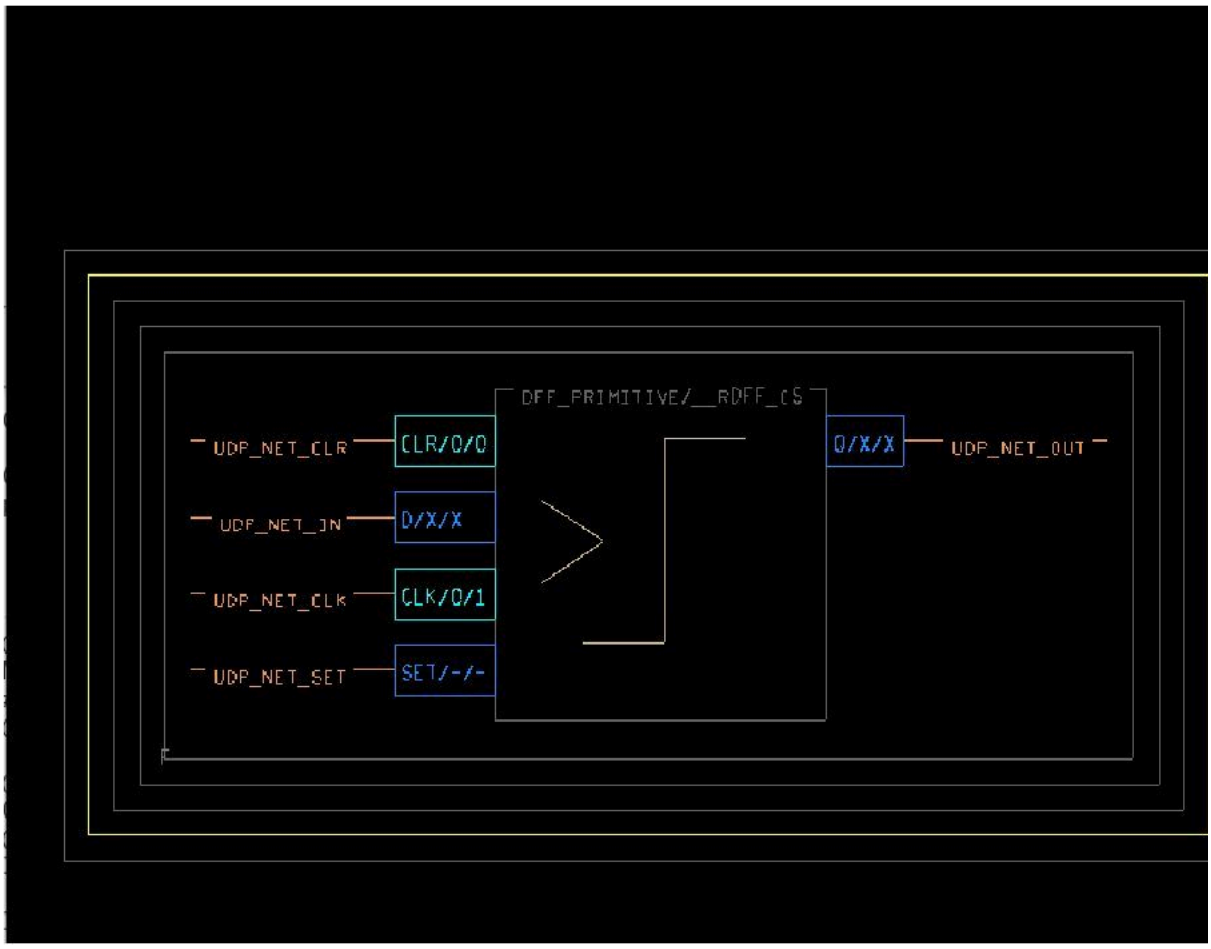


d. Getting back to previous circuit state view by the view previ

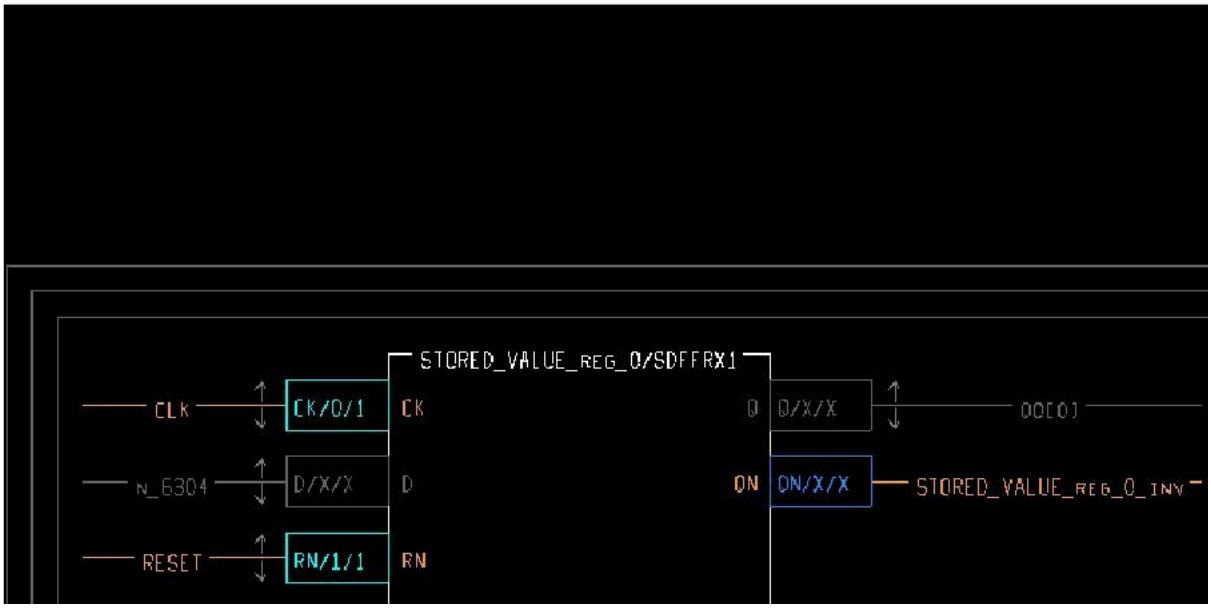
to traverse across your displays.

3. **Zoom in and Zoom out**

If you have a scroll wheel on your mouse, hold the control key c to zoom in and out. If nothing happens, click on **Options** -> **Co** there.

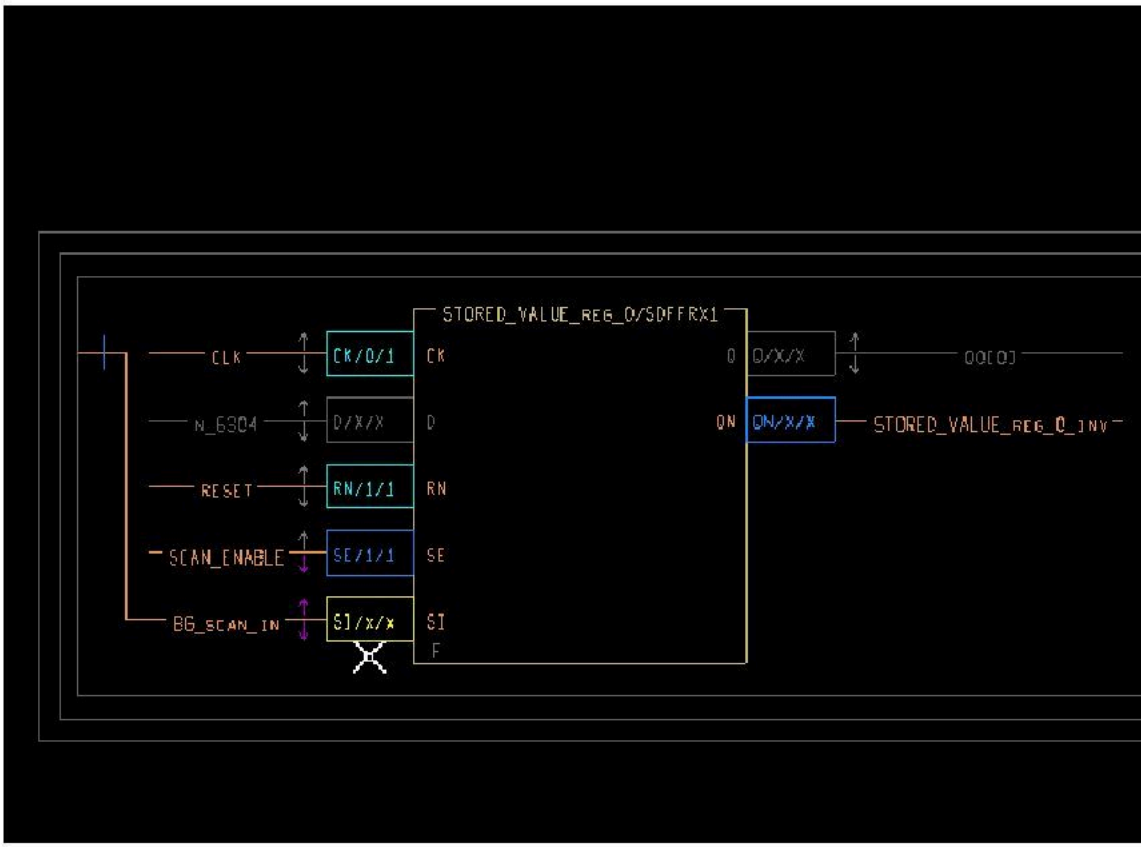


2. Select the **SDFFRX1 cell** hierarchy and click **Implode**.



4. Look at the values specified on the pins of the cell and compare
in the lecture. In this case for the broken chain **DLX_CHIPTOI**
Scan-in pin is at lower "x" state, means this is pin is unconnecte

5. **Left**-click on a pin and type **b** to trace backward and **f** to trace f
the Scan-in pin and trace backward.



Note: Noticed that the Scan-in pin is not connected and not

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?

Lab 4-2 Debugging Broken Scan Chains with Tcl Cc

Objective: To debug the broken scan chains with the help of c

This lab helps users to learn the debugging techniques using the command l

Command Line Utilities to Use During Debugging

In this section, we will discuss the important commands to use while debug using command line interface.

1. **set_context**

   Context refers to a circuit testmode/experiment configuration. This analysis is performed. As discussed in **Lab 4-1** context can be set us line.

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

2. **simulate_state**

   This utility does a simulation of a specific simulation state and allov state before analyzing the circuit. This reports the values that are pre 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 informati report_chains. In order to list all the scan chains, use the following c

   ```
   report_chains -summary
   ```

   Note: The chains reported by report_chains either have a obser

5. **report_testfunctions**

   Users can report the test functions grouped with a specific type, scan and capture clocks, scan constraints, test constraints, scan i

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

6. **trace_circuit**

   Similar to tracing in GUI, users can do circuit tracing on the con you find the active fanin or fanout cone in a specific state. Use t backward to perform the tracing. Users can also use the –show 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 cha 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 automatic You can fetch the list of existing testmodes using the

3. Get the testmode by using the command:

   ```
   get_db testmodes
   ```

4. Set the analysis context by using the command:

Finding a Broken Scan Chain

Let's use the **report_chains** utility to fetch the information on the broken s
you the information on good chains and broken chains. To report the summ
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[2
| 9                | 15               | 85     | {DLX_CHIPTOP_DATA[8]}  | {DLX_CHIPTOP_DATA[2
| 10               | 16               | 85     | {DLX_CHIPTOP_DATA[9]}  | {DLX_CHIPTOP_DATA[2
| 11               | 2                | 85     | {DLX_CHIPTOP_DATA[10]} | {DLX_CHIPTOP_DATA[2
| 12               | 3                | 85     | {DLX_CHIPTOP_DATA[11]} | {DLX_CHIPTOP_DATA[2
| 13               | 4                | 85     | {DLX_CHIPTOP_DATA[12]} | {DLX_CHIPTOP_DATA[2
| 14               | 5                | 85     | {DLX_CHIPTOP_DATA[13]} | {DLX_CHIPTOP_DATA[2
| 15               | 6                | 85     | {DLX_CHIPTOP_DATA[14]} | {DLX_CHIPTOP_DATA[3
| 16               | 7                | 85     | {DLX_CHIPTOP_DATA[15]} | {DLX_CHIPTOP_DATA[3
-------------------------------------------------------------------------------------------------

--------------------------------------------------------------
                  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     |
```

Note: It is easier to trace backward on broken chains then f
first broken Observe Chain.

Debugging Observe Scan Chain 1

Start with the first one based on the SO pin **DLX_CHIPTOP_DATA[1**
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 TVI_722]
```

The above listing generated from report_chains showing all of the c
Modus recognizes. The Observe Flop in Position 1 is the closest flo
stopped at Observe Flop 58. This is the last Flop that Modus recogn

```
-------------------------------------------------------------
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_
| 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 tech pins, names and values along with the hierarchy report of the values for all the pins are at **X**. In this case it is because we h in any particular simulated state. The available states are the 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 v signals and test constraints necessary for proper shifting of t

```
@modus:root:/ 6> simulate state scan
```

```
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, repo DLX_CORE.PC_REG.STORED_VALUE_reg_0 instance to check using the command:

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

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

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

-------------------------------------------------------------------
                  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_RE
| MODULE          | REG_MULTIPLE_PLUS_ONE_OUT_RESET | DLX_CORE.PC_RE
| MODULE          | DLX_CORE                        | DLX_CORE
| MODULE          | DLX_TOP                         | Block.f.l.DLX_
-------------------------------------------------------------------

End of Instance Report [end TUI_720]
```

We now have the logical values on the flop we are starting our c
care about on a scan flop during shift? Clocks, Resets, Shift Ena
pin values are:

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

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


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

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.STO
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      |
```

```
-------------------------------------------------------------------
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
-------------------------------------------------------------------
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_796.__iNsT1.dff_prim
| e}
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_797.__iNsT1.dff_prim
| e}
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_798.__iNsT1.dff_prim
| e}
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_799.__iNsT1.dff_prim
| e}
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_800.__iNsT1.dff_prim
| e}
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_801.__iNsT1.dff_prim
| e}
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_802.__iNsT1.dff_prim
| e}
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_803.__iNsT1.dff_prim
| e}
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_804.__iNsT1.dff_prim
| e}
| {DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.__iNsT1.dff_prim
| e}
-------------------------------------------------------------------

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

Notice the analysis stopped at Observe Flop 10. This is the last
valid scan flop in the chain. This is the flop to start tracing back

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

   ```
   report_instances DLX_CORE.THE_REG_FILE.\REG.REGI
   ```

   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.I

-------------------------------------------------------------
                  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                                                  | Acti
-------------------------------------------------------------------------
| Input     | DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.CK | true
| Input     | DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.D  | true
| Input     | DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.SE | true
| Input     | DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.SI | true
| Output    | DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_reg_805.Q  | true
-------------------------------------------------------------------------

-------------------------------------------------------------
                  Instance Hierarchy Report
-------------------------------------------------------------
-------------------------------------------------------------------------
| Level           | Module   | Instance
-------------------------------------------------------------------------
| TECHNOLOGY_CELL | SDFFQX1  | DLX_CORE.THE_REG_FILE.REG.REGISTER_FILE_r
| 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 debu
care about on a scan flop during shift? Clocks, Resets, Shift Enable,
pin values are:

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

b. D=X (Data pin looks good)

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


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

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 trace_circuit report and found that SE pin is not connected.

5. Go ahead and try to debug all the other broken chains and matcl the GUI debug!

Exiting the Modus

Close the Modus tool by entering the command:

```
exit
```

Module 5: Analyze Initia Sequence Usi GUI

Lab 5-1 Analyzing Initialization Sequences Using

Objective: To get familiarize with the process of sequence

In this lab, the design is processed from Build Model to Verify Test Str to bring up the analyze sequences GUI with a watchlist. This method is initialization sequence does not get you into the proper test mode or the

Invoking Modus

Let's create a new test model and mode to learn sequence analysis. Ope 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:

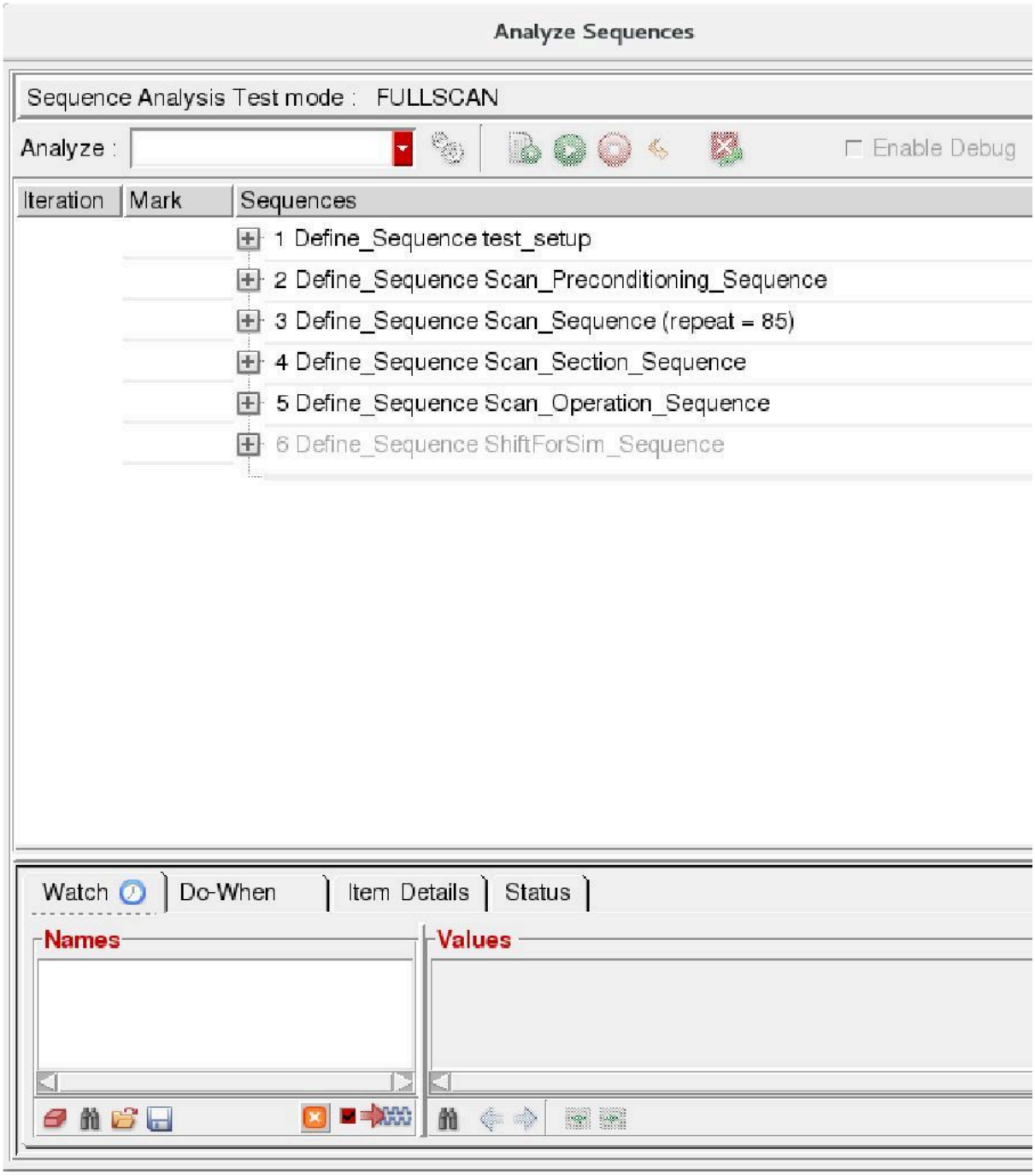
   ```
   source ./run_lab4.tcl
   ```

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

Setting the Analysis Context and Opening the Sequence Ana

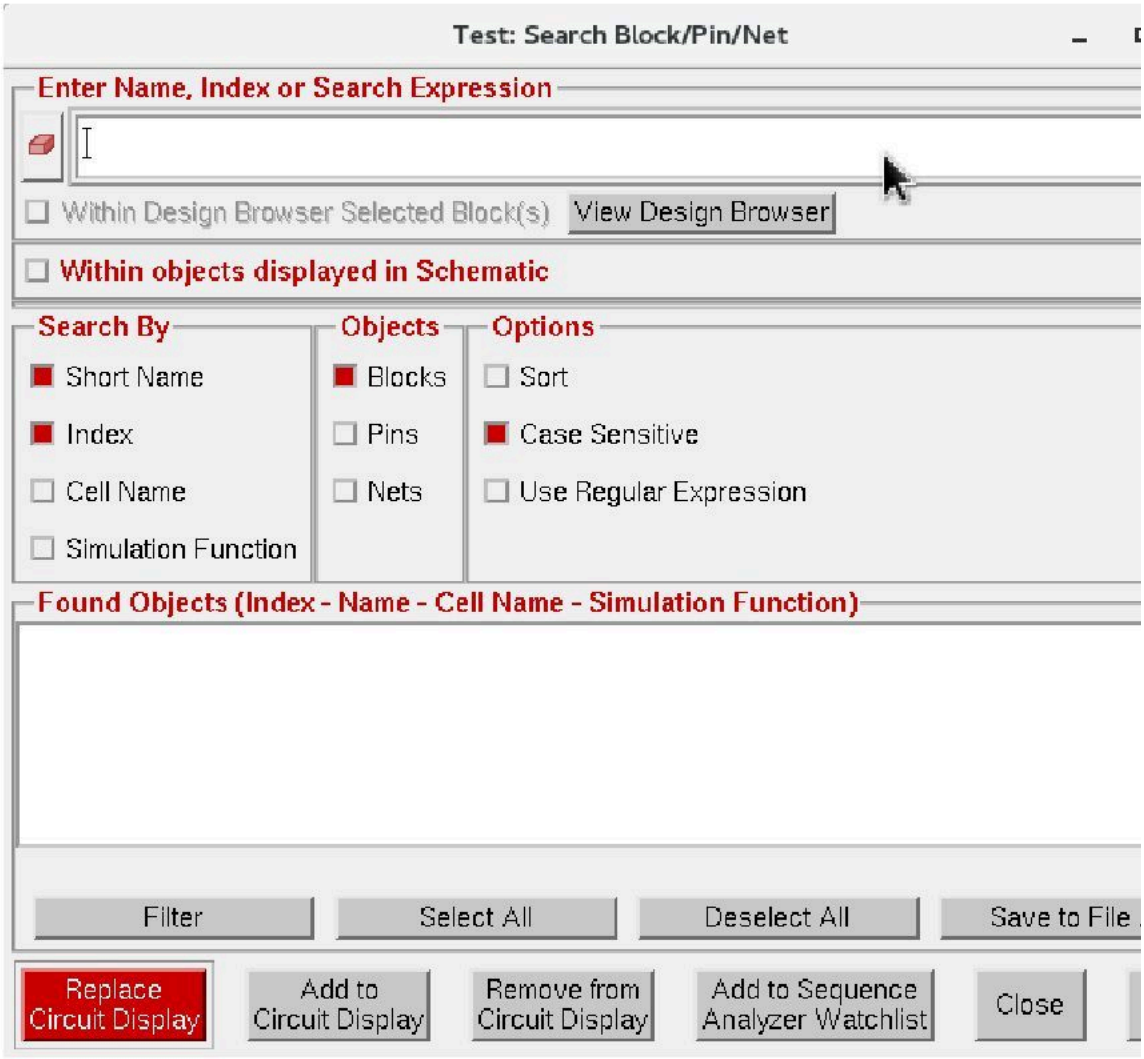
1. In the Modus GUI and click on the **Analysis Context** icon 

Analyze Initialization Sequence Using Modus GUI

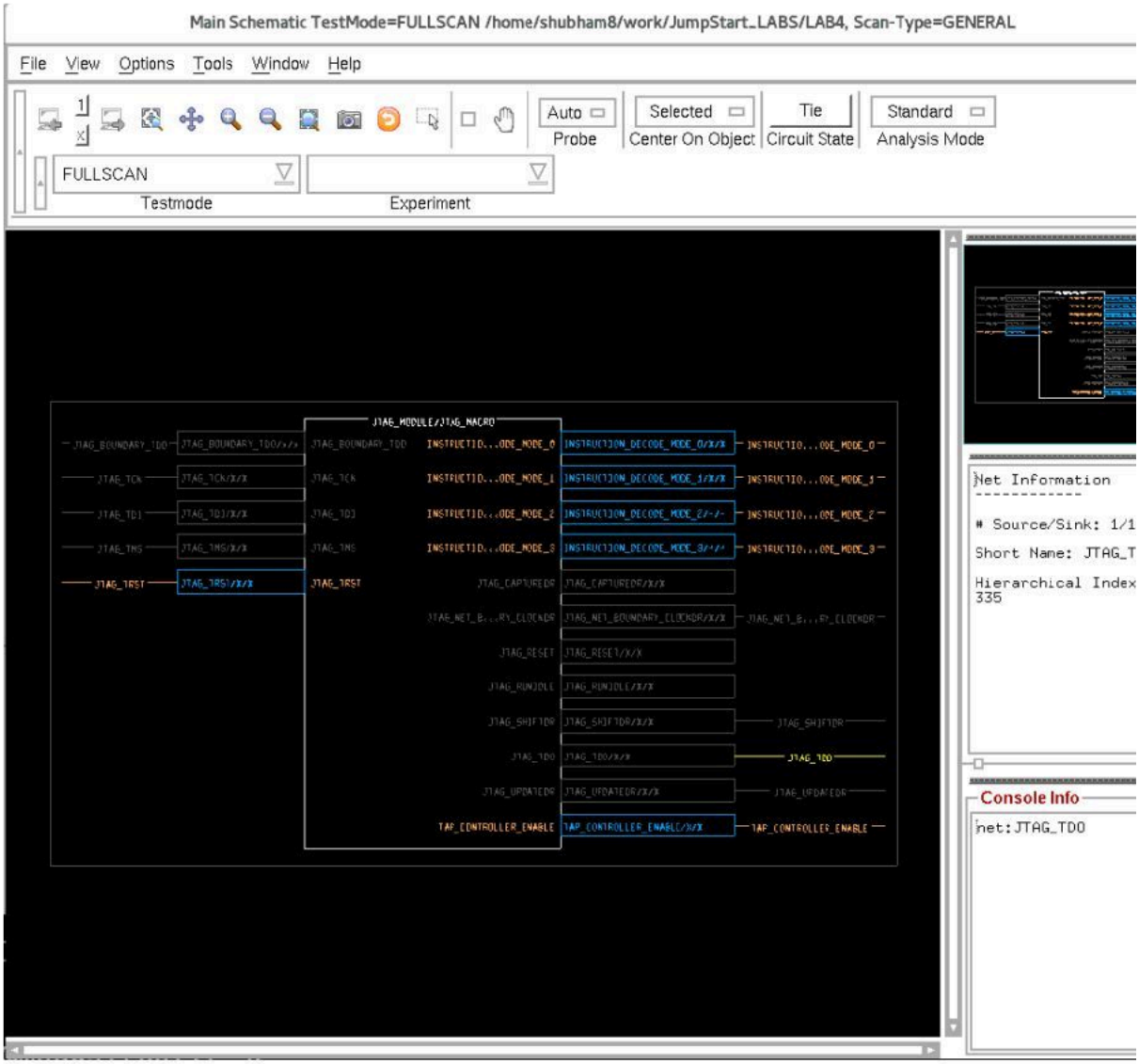


Opening the Schematic and Creating a Watch List

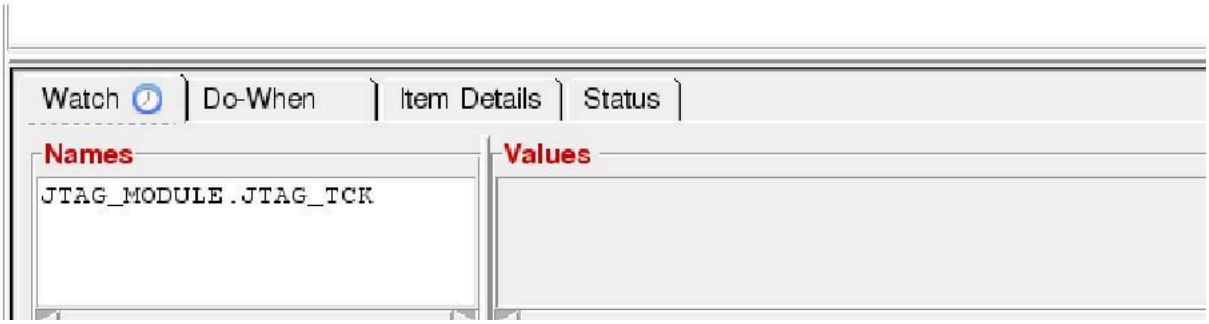
We now want to go to the schematic GUI and find signals we want to trace

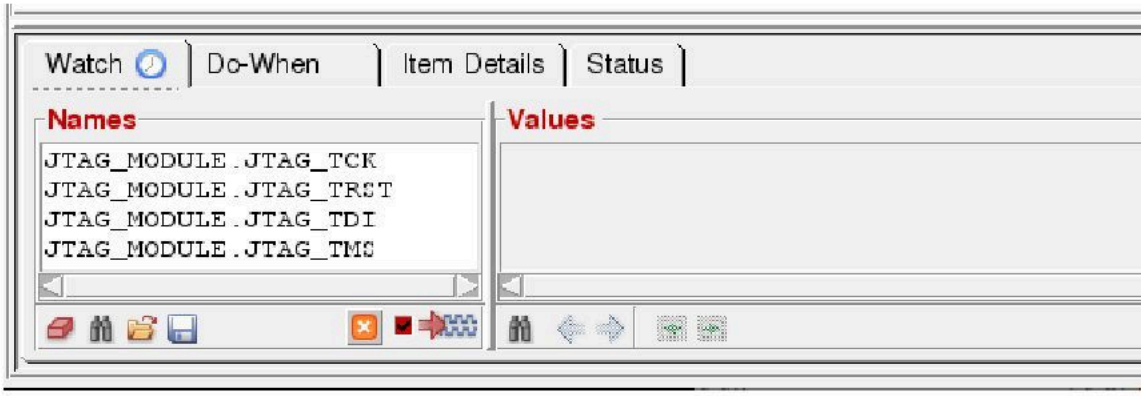


2. Type in **JTAG_MODULE** into search field and hit return. This module into the schematic view. We want to capture all of the p can track them in the analyze sequence GUI.

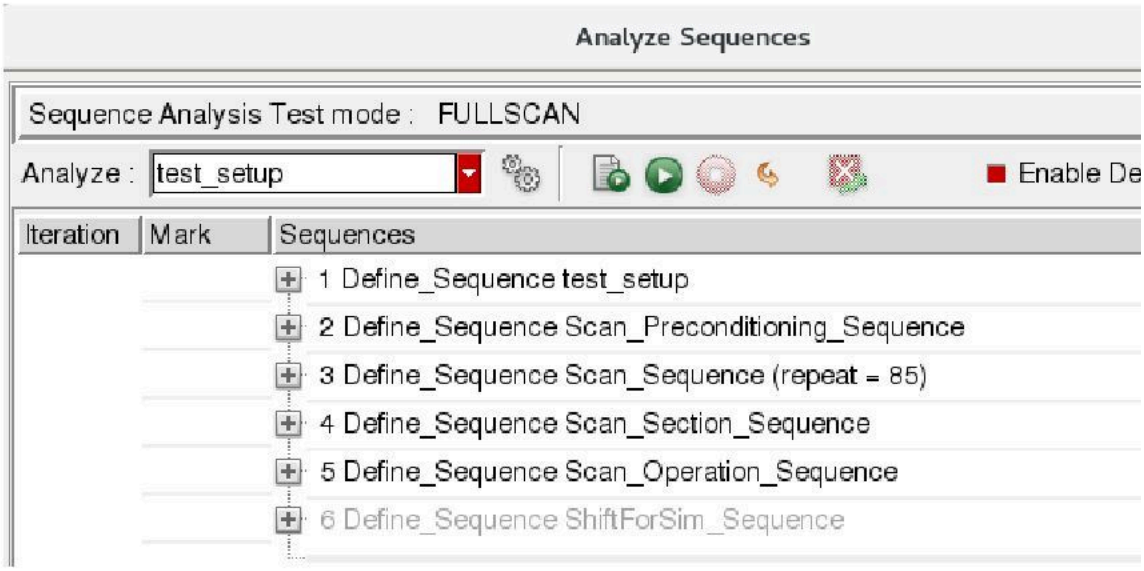


3. **Left**-click on the **JTAG_TCK** pin. Now, **right**-click and select **Ad**
Watch List from the pulldown menu If you will go back and look a
window you will see this signal added in the lower left corner.





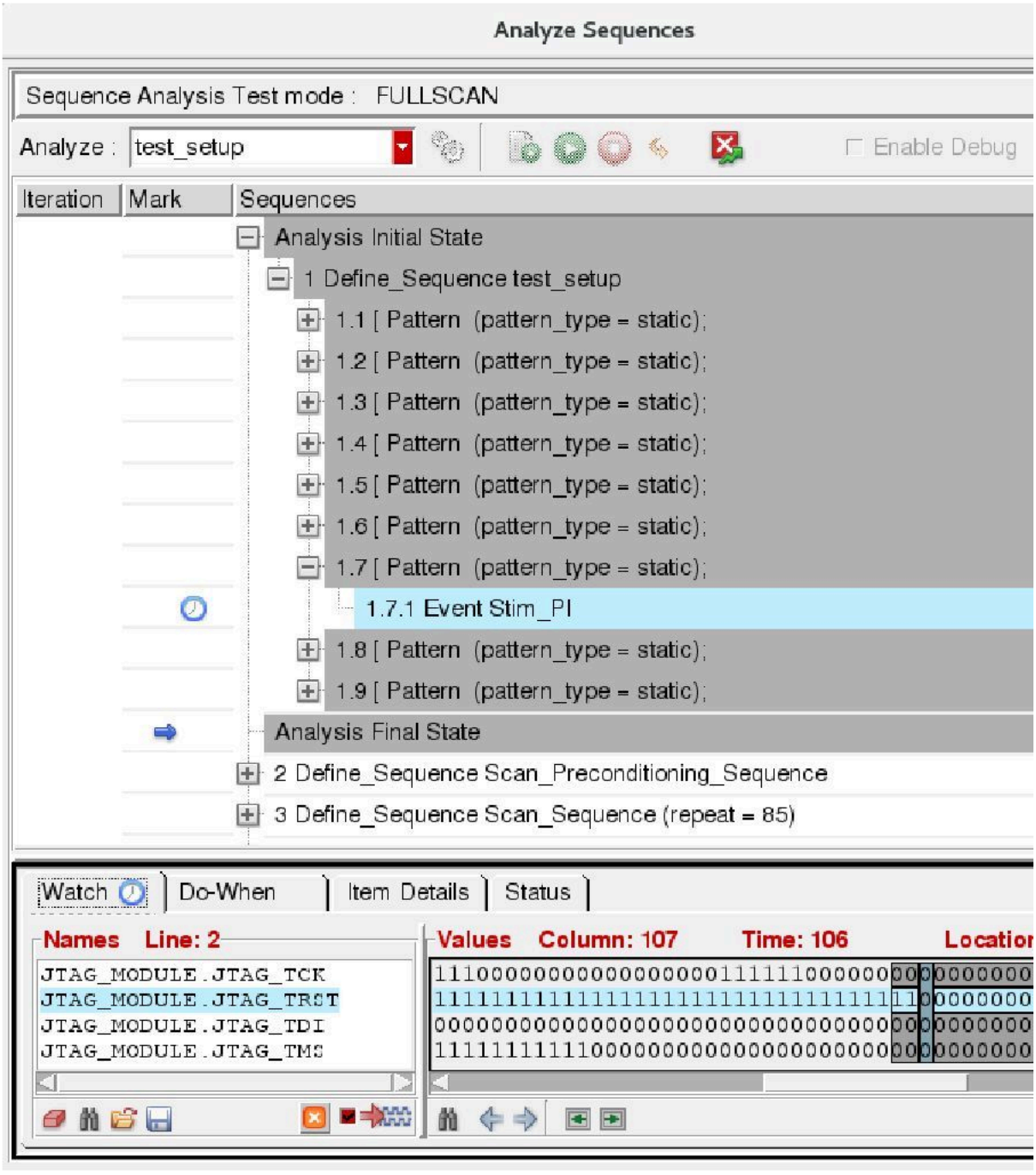
2. At top of window select test_setup in the Analyze pulldown. Th sequence that we used in the **DLX.seqdef** file.



3. Select the **Initialize Simulation** icon (small green play butt in the bottom pane and a blue Mark Arrow up top showing you sequence. SimVision will also start up with the watch signals in

4. Now, click the Large green play button . The full initializati completion. In the bit stream output below, click on any of the b changes in the different windows.

Analyze Initialization Sequence Using Modus GUI



5. Now let us set a condition that will automatically stop when JTAG_

   a. Click the Red X w/green arrow icon at the top in the Analyz
   This will terminate and reset the analysis. Close out the SimVisi

- 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
```

Module 6: Debugging th
Pattern

Lab 6-1 Simulating and Debugging Vectors

Objective: To learn the vector simulation and debug envirc

This lab helps to simulate and debug the vectors in SimVision™ and Xn

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 the Modus software. Type **which xmverilog** and make sure it cc not Modus path.
- We will create a standard faultmodel (stuck-at) and generate test
  - Simulate the tests in **xmsim** and verify they pass.
  - Bring up the passing vectors in **Modus** and trace them in the
- Create a **Bridging Faultmodel** which contains only a small set
  - Create targeted tests for these few faults and simulate them.
  - Simulate the patterns against a faulty circuit and see how the
  - Bring up the tests in simulation debug environment and anal

Invoking Modus and Generating Vectors

Note: The script will execute build_model -> build_testmode -> build_faultmodel -> create_logic_tests -> write_vector script and should be able understand it now. Look at the we will be writing out the parallel vectors in Verilog lan

Running Good Simulation

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

1. The verilogsim/run_sim_good script contains the command to run t 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 scr chain tests and logic tests. you should now see 0 miscompares:

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

Running Bad Simulation

Let's run the simulation on a faulty netlist and see check if we get some mi we have inserted a **stuck at fault** inside the **DLX CORE** netlist. We can n

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

Debugging Simulation Miscompares

For debugging the scan-based designs, the first step is to identify the fai
ways to get this information. Since, we are using **scanformat parallel** v
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 l
   -scanformat parallel -includelatchnames
   ```

   Note: Use the **–includelatchnames** keyword with write_ve
   testbench. The **-includelatchnames** is only supporte
   vectors. If you are running a serial simulation this is

2. Rerun the Verilog simulation (./verilogsim/run_sim_bad) with n
   testbench. You should see 9 miscompares. Open the log file **xm**
   and search for miscompares. You can identify that the failing lo
   DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_re

3. Another method is shown in the script **run_findflops.tcl**. This i
   When you run Verilog testbenches with the +FAILSET switch,
   "./testresults/verilog/VER.FULLSCAN.logic.data.logic.ex1.t
   CPP format which we use for diagnostics.

   a. Open the run_findflops.tcl script file and go through the con

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

   b. Source the script file in the Modus prompt by using the com

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

Good Machine Simulation

We need to compare the difference between the good simulation and the fa simulation is referred as the simulation done by Modus. While the bad simu faulty netlist using xrun.

Let's also look at the GOOD machine values in SimVision. You need to cr pins you want to visualize in waveform viewer. One such list available in fi

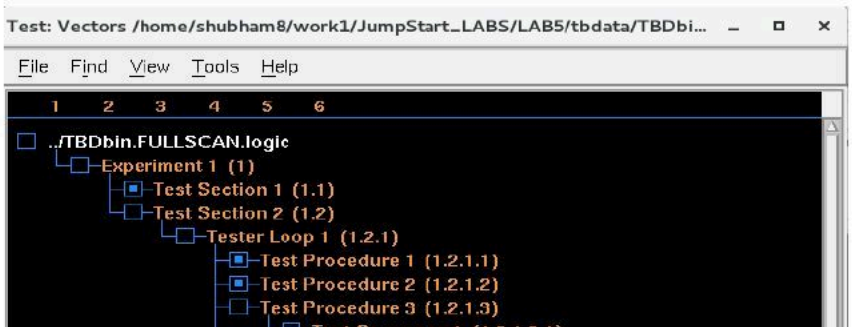
1. Run analyze_vectors at the Modus prompt to generate the SimVisio pins.

```
analyze_vectors -TESTMODE FULLSCAN -INEXPERIMENT lo
-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 want to look at **1.2.1.3.7**. Let's keep the vectors range to

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

3. On the bottom-right of the analysis context window click on the **Vie** the 5th from the left of the large icons. This will bring up the TBDb simulated. Now we want to find that failed vector from the xmsim l tree until you get to Pattern **1.2.1.3.7**



4. You can now click on it once and then **right**-click your mouse a **Values**. This will actual go off and simulate the full vector set u So now you can bring up elements in the circuit and trace back t

   Note: The schematic window should have also popped up i

   Try this out by clicking **View->Block** and type in **DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_re** bring up the measure fail flop. If you now hit the **b** button it wil you can mouse over the pins and see the actual circuit values of

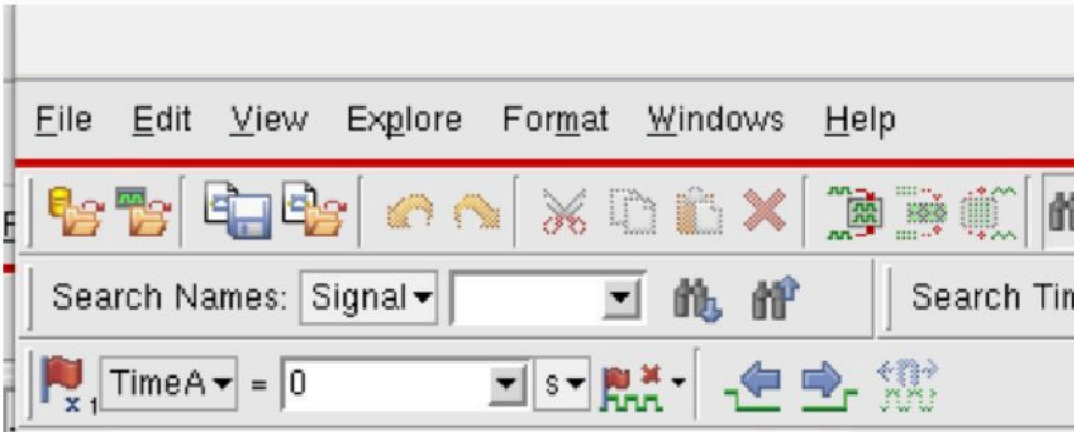
   What is the GOOD machine expected value on the Q pin?

   Does this match the expected value in our Verilog sim?

5. Go back to the main MODUS GUI and click on the **View Wave** the right-side, 8th from the left of the large icons. A popup wind TBscope.FULSCAN.analyzed.trn file to open SimVision.

   a. In SimVision will see Experiment, TestSection, TestProcedu the left side of the waveform window. You want to find the of 1 2 1 3 1 in the waveform window. So, the first '1' value '1' is the TestSection, etc.

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



c. Remember when we Analyzed the Vectors and imported a watc
now available in this Design Browser. Traverse through the desi
the signal you wish to view in the waveforms. You should be ab
DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE_re

d. You can zoom in/out using the icons  . Explore the w
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 **C**
by **MODUS**. One was through the **MODUS** schematic and the other throug

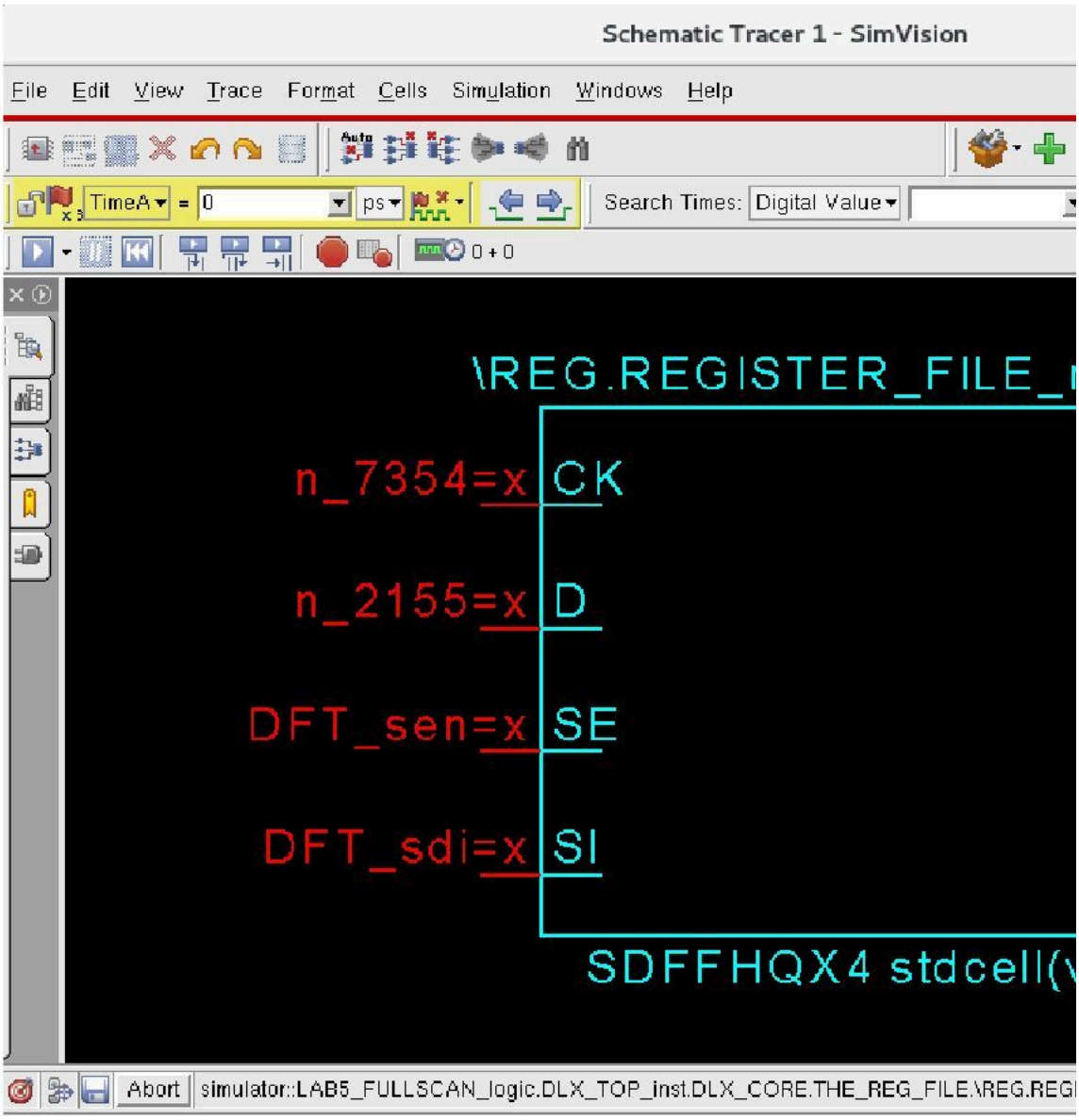
Bad Machine Simulation

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

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

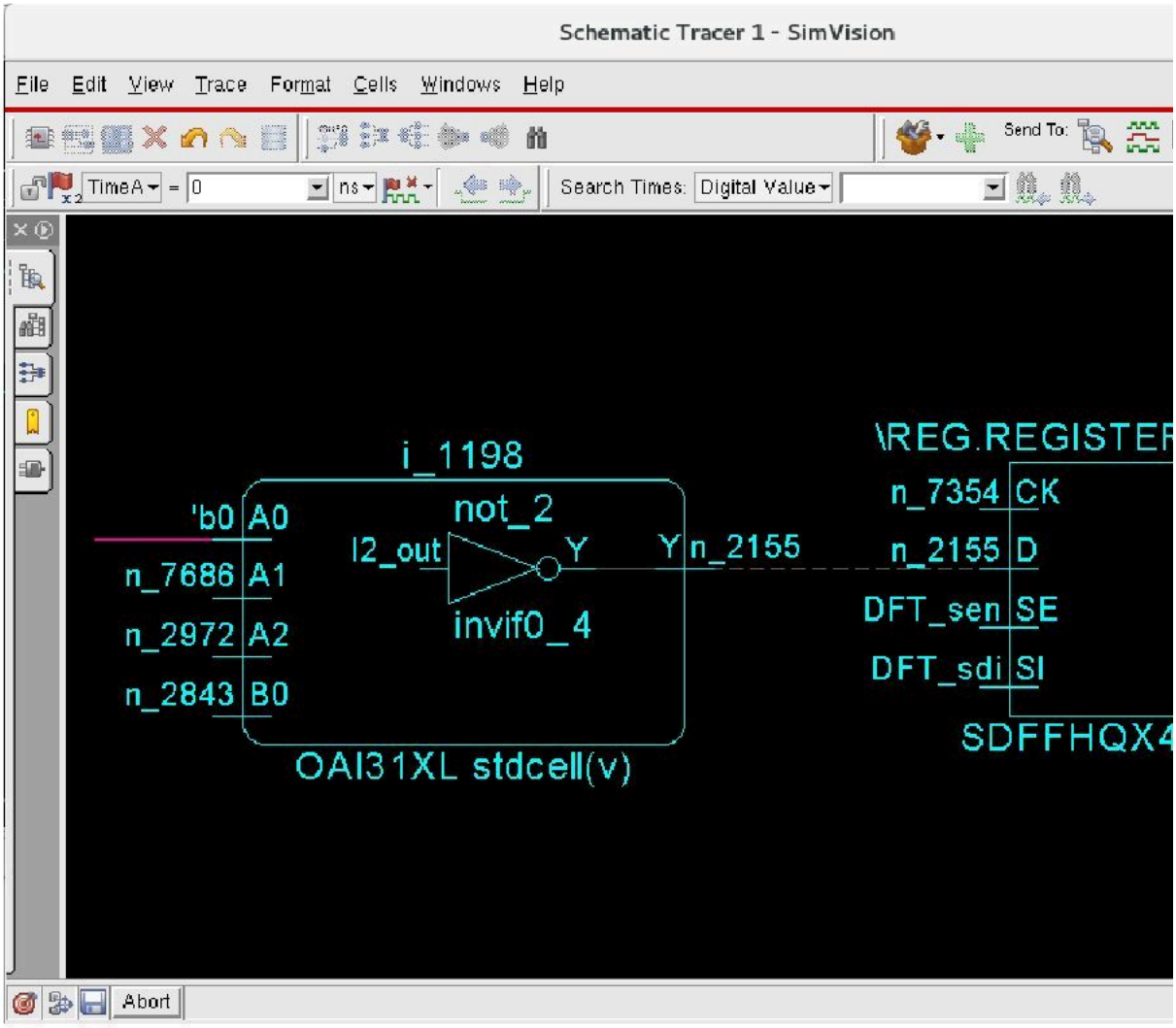
```
xrun \
    +access+rwc \
    +gui \
    +xmstatus \
    +xm64bit \
    +TESTFILE1=./testresults/verilog/VER.FULLSCAN.logic
    +TESTFILE2=./testresults/verilog/VER.FULLSCAN.logic
    +HEARTBEAT \
    +FAILSET \
    +xmtimescale+1ns/1ps \
    +xmoverride_timescale \
    +xmseq_udp_delay+2ps \
    +libext+.v+.V+.z+.Z+.gz \
    +xmlibdirname+$WORKDIR/Inca_libs_10_13_21 \
```

3. Once the flop is loaded in the schematic, you can back trace the
clicking on the pins. You can add any pins in the schematic to y
selecting the pin and then clicking the waveform Icon in the bar



4. Back trace the pin D of the flop. You should reach the instance i
this cell to waveform window by clicking on the cell and then ri

Debugging the Test Pattern

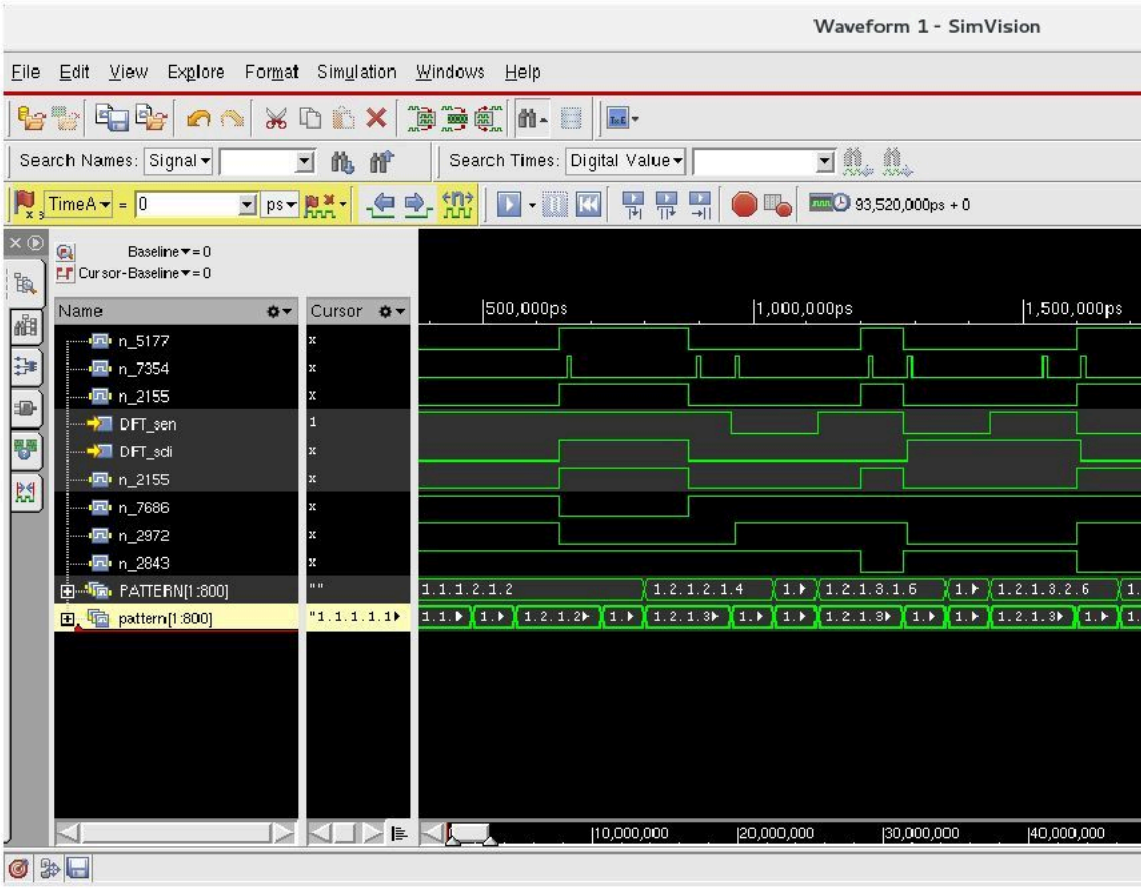


5. Go back to design browser and select the testbench name "**LAB5_F** can see the nets on the right pane. Search for "**pattern**" and send it

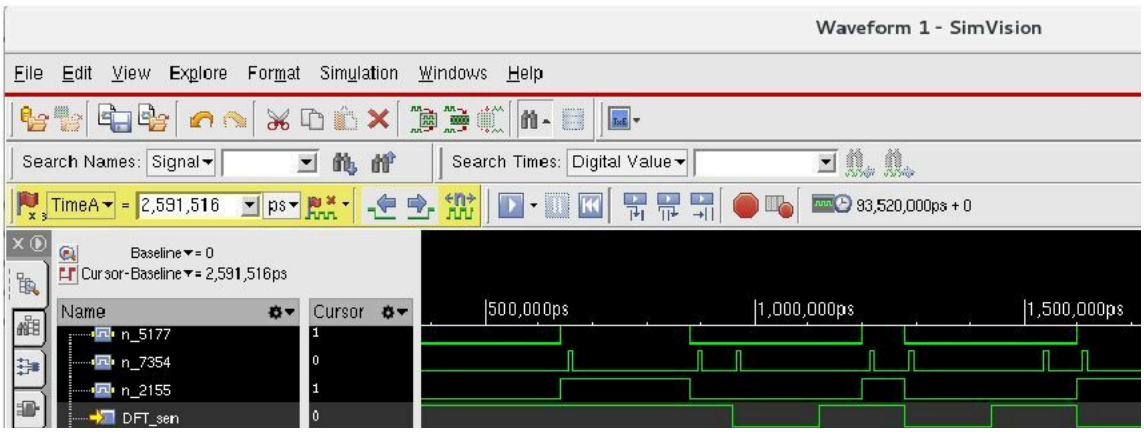
6. On the waveform window change the radix of the waveform "patter waveform element under "cursor" click on Radix/Mnemonic and ch ASCII. This should show the pattern odometer on x axis and will he odometer. **1.2.1.3.1**.

7. Go back to the SimVision Console and type run at the SimVision pr will simulate the testbench.

8. Go to the required odometer now you have the Good Machine a
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 th
that is not the case. Back trace the pin further and you can see th



Alternate way of Simulation Debug

There is one more way of bringing up **SimVision** on the **BAD** machine sim
tool to dump the values of the nodes/ports in one directory (simvision.shm)
SimVision(UI) and can reach the desired flop.

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

```
xrun \
    +access+rwc \
    +simvision \
    +xmstatus \
    +xm64bit \
    +TESTFILE1=./testresults/verilog/VER.FULLSCAN.logic
    +TESTFILE2=./testresults/verilog/VER.FULLSCAN.logic
    +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
```

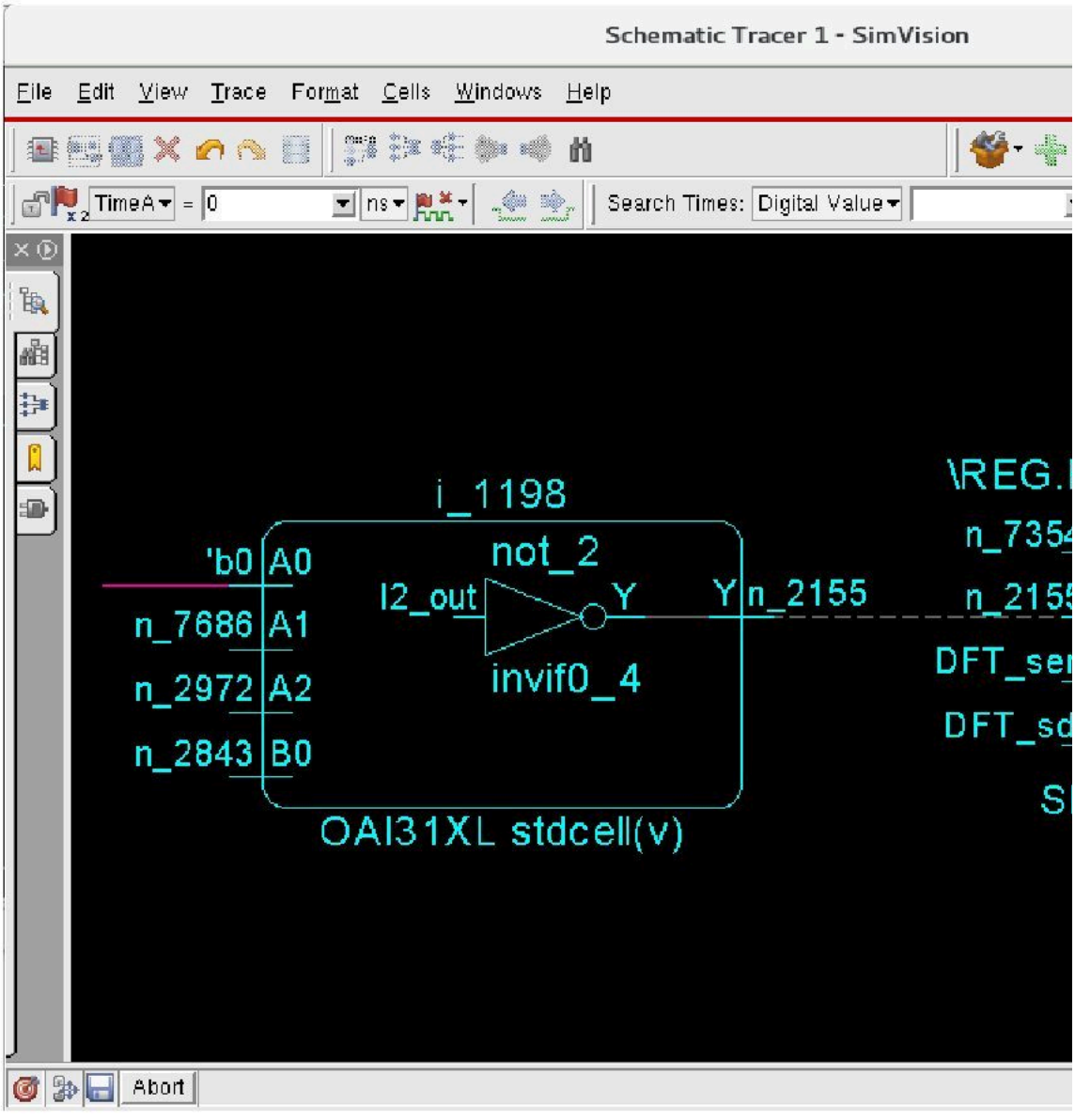
   Or you may use the already created script ./verilogsim/run_sim_bad

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

   Note: Once the simulation is over you can see a directory "simvisi
   database.

6. Once the flop is loaded in the schematic, you can back trace the
clicking on the pins. You can add any pins in the schematic to y
selecting the pin and then clicking the waveform Icon in the bar

7. Back trace the pin D of the flop. You should reach the instance i
this cell to waveform window by clicking on the cell and then **ri**
Waveform Window".



8. You should be able to see that the value on A0 of instance

Appendix: Lab Answers

Lab 3-1 Perform the Modus ATPG (Automatic Test Patter

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 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 mo

Answer: 3663

Lab 3-2 ATPG Vector Generation

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.REGI
- Unconnected clock at net DLX_CORE.THE_REG_FILE.__

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_

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

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_re data from Observable scan bit 6.
- Tracing backward identifies a loop in the scan chain data tha

Answer:

- The driving Flip/Flop DLX_CORE.THE_REG_FILE.\REG.REGISTER_FILE controllable clock.

- Tracing the clock on the above FF finds an AND gate DLX_CORE.THE_REG_FILE.i65129. The clock is not AND gate. The B pin needs to be a 1 during the scan sta 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

Analyzing Sequences

Where is the Blue Mark arrow?

Answer: 1.7.1

Is the proper transition highlighted in the Watch window

Answer: Yes