"TEST-AWARE LOW POWER CLOCK GATING IN DSP BLOCKS"

ECE -242 DIGITAL SYSTEM TESTING AND TESTABLE DESIGN

Advisor: Dr. Reza Raeisi

Presented By

Sravani Yalaganamoni (301851603)

Contents

1. Abstract

This project presents the complete design, synthesis, and validation of a low-power Finite Impulse Response (FIR) filter with integrated scan chain insertion for enhanced testability. FIR filters are fundamental components in digital signal processing (DSP) applications, requiring both high-speed computation and hardware efficiency. The design was implemented in Verilog HDL and verified at the Register Transfer Level (RTL) using ModelSim for functional correctness. Following simulation, the design was synthesized using Synopsys Design Compiler with the Nangate 45nm Open Cell Library, producing detailed area, power, and timing reports.

To address power efficiency, clock gating was incorporated to reduce dynamic power by disabling the clock to inactive portions of the design. Simultaneously, scan insertion was applied using Synopsys tools to ensure full testability, even in the presence of gated clock paths. The scan-enabled design was validated through post-synthesis simulation, demonstrating functional equivalence with the RTL design. ATPG (Automatic Test Pattern Generation) was performed using Synopsys TetraMAX, achieving 100% fault coverage for modeled stuck-at and transition faults, with minimal pattern count and low computational overhead.

Keywords:
FIR Filter, RTL Design, ASIC Implementation, Synopsys Design Compiler, Scan Chain Insertion, Design for Testability (DFT), TetraMAX, ATPG, Fault Coverage, Post-Synthesis Simulation, Area and Power Analysis, Timing Optimization, Digital Signal Processing (DSP)

2. Introduction

Finite Impulse Response (FIR) filters are integral to a wide range of applications in digital signal processing, including audio enhancement, image processing, and communications. FIR filters are preferred for their inherent stability and ability to achieve exact linear phase responses, which are vital in systems where signal integrity is paramount. However, as digital circuits grow in complexity, the design and implementation of FIR filters need to be optimized not only for performance but also for hardware efficiency and testability [1].

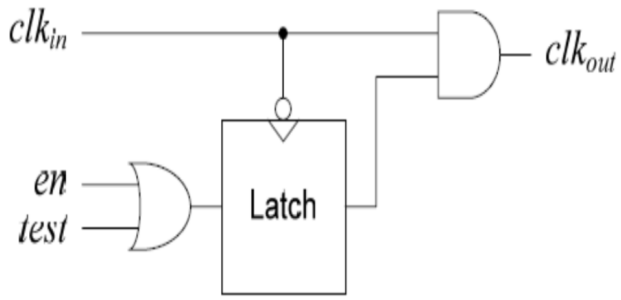


Figure 1: Clock Gating Cell [1]

The objective of this project is to design a 7-tap FIR filter capable of processing 8-bit input data and generating filtered outputs based on fixed tap coefficients. The filter is first described using Verilog Hardware Description Language (HDL) at the RTL level, where modularity, clarity, and simulation flexibility are emphasized. A comprehensive testbench is developed to apply stimulus vectors and observe the output response using ModelSim, enabling waveform-based validation of the filter's correctness [2].

Following functional verification, the FIR filter is synthesized using Synopsys Design Compiler. This process translates the behavioral RTL code into a gate-level representation that adheres to the constraints of a 45nm standard cell library. Synthesis outputs such as area utilization, cell counts, power breakdown (dynamic and leakage), and timing reports (slack, setup/hold margins) are analyzed to assess the hardware efficiency of the design.

To facilitate post-silicon testing and improve defect detectability in mass manufacturing, the design undergoes scan chain insertion using Design Compiler and is subsequently verified using Synopsys TetraMAX. The ATPG process is executed to evaluate the fault coverage of the inserted scan chains and generate test patterns. The design achieved full test coverage (100%) for all modeled faults, including transition faults, indicating high testability. Additionally, schematic visualization and waveform analysis post-synthesis confirm the logical and structural integrity of the scan-inserted design [3].

This project not only demonstrates a complete digital design methodology from RTL to a testable netlist but also highlights the importance of integrating DFT techniques into high-performance DSP designs using industry-grade EDA tools.

2.1 OBJECTIVE

The primary objective of this project is to design and implement a low-power Digital Signal Processing (DSP) block—specifically, a Finite Impulse Response (FIR) filter or a Multiply-Accumulate (MAC) unit—with integrated test-aware clock gating. The goal is to reduce dynamic power consumption by disabling the clock signal to idle functional units while preserving full testability during scan-based testing [3]. The project also aims to evaluate the impact of clock gating on testability, particularly Automatic Test Pattern Generation (ATPG) fault coverage, and to develop a scan-compatible clock gating strategy that bypasses clock suppression in test mode. To validate the proposed methodology, the design is synthesized using Synopsys Design Compiler and evaluated with Synopsys TetraMAX for fault coverage, power analysis, and area efficiency [1][2].

2.2 MOTIVATION

As modern digital systems increasingly demand low power consumption—particularly in areas like wireless communication, biomedical devices, and portable electronics—techniques such as clock gating have become essential. Clock gating significantly reduces dynamic power by cutting off the clock to inactive modules, thus minimizing unnecessary switching activity. However, this optimization often introduces critical challenges in scan-based testing, where full observability and controllability of sequential elements are necessary to achieve high fault coverage [5]. In digital signal processing (DSP) blocks, which inherently exhibit high switching activity and are vital to performance-intensive applications, the integration of power-saving mechanisms must be carefully managed to avoid compromising testability. This project is motivated by the need to address this trade-off, ensuring that low-power design techniques can coexist with high fault coverage and reliable testing standards in ASIC development.

2.3 PROBLEM STATEMENT

Clock gating is widely used to reduce power consumption in digital circuits by disabling the clock signal to inactive sections of the design. However, this technique poses significant challenges when applied to designs that require high testability through scan-based methods. Specifically, during ATPG-based scan testing, gated clock domains may inhibit the propagation and capture of fault effects, leading to a reduction in fault coverage. This limitation becomes critical in DSP blocks such as FIR filters or MAC units, where maintaining both performance and testability is essential. The core problem addressed in this project is how to integrate clock gating into a DSP block without undermining scan-based testing effectiveness [2][3]. The challenge lies in modifying the clock gating logic to selectively override gating during scan mode, enabling fault propagation and detection while preserving low-power benefits during normal operation [6]. The project aims to develop a solution that achieves this balance and validates its effectiveness using industry-standard EDA tools.

3. Literature Review

The design of low-power and testable digital signal processing (DSP) architectures has garnered significant attention in recent years, particularly for applications demanding energy efficiency and reliability. Among various optimization techniques, clock gating has emerged as a potent solution to reduce dynamic power consumption [6]. However, incorporating such techniques into scan-testable designs poses unique challenges.

In the work by Heřmánek et al., the authors demonstrated the application of clock gating to a reconfigurable embedded DSP platform, showing notable dynamic power reduction by gating idle functional units like multipliers and ALUs [2]. This study validated clock gating's applicability to both ASIC and FPGA-based systems, offering up to 30% power savings during idle states without significantly impacting performance[1].

To further improve energy efficiency, Reddy and Choi proposed an optimized clock gating cell tailored for nanoscale CMOS technologies. Their cell design addressed both switching and leakage power,

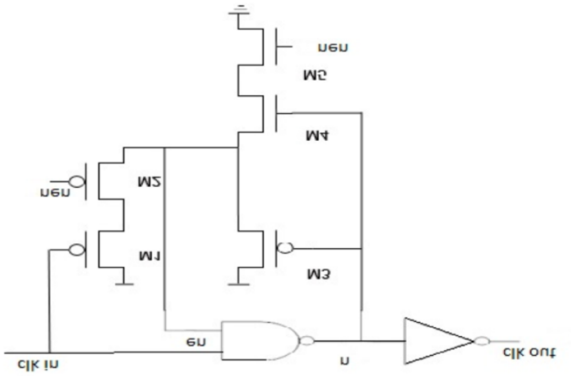


Figure 2: Low Power Clock Gating [2]

reinforcing clock gating's relevance in high-density DSP architectures.

An innovative contribution by Vaidya et al. introduced the use of Vedic Mathematics in conjunction with clock gating to enhance both computational speed and power savings in DSP co-processors. Their Verilog-based implementation showed that Vedic multipliers could significantly outperform traditional multipliers in terms of delay, while also being area-competitive.

From a testability standpoint, Tsai and Rajski addressed the limitations of scan-based testing in clock-gated and multi-clock domain designs by introducing a clock-domain-aware test methodology. Their approach utilized pulsed interactive clocks (PIC) and selective flop holding to reduce pattern count by up to 39% without compromising fault coverage, highlighting the need for adaptive scan-insertion strategies in power-optimized circuits[3].

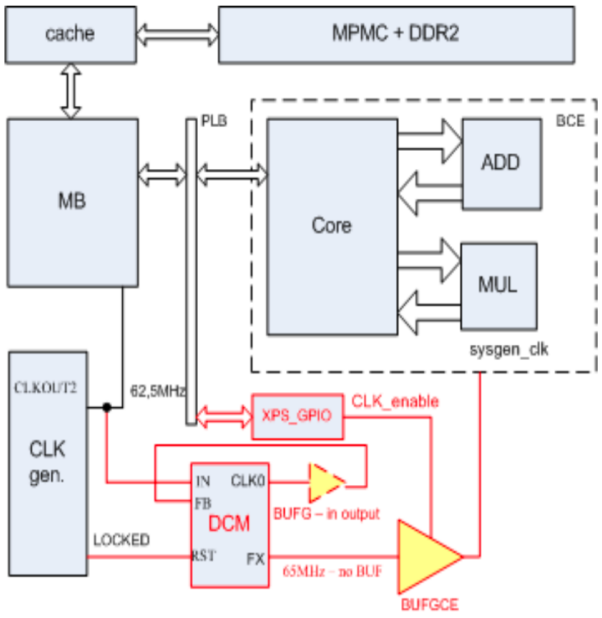


Figure 3: System with Clock Gating [1]

Complementing these findings, your project aligns well with the proposed methodologies in integrating test-aware clock gating for FIR filters. The methodology used in your design—such as introducing a test mode override during scan insertion—follows practices discussed in several papers, notably the draft proposal that emphasizes overriding gated clock logic via OR-gates during scan testing to preserve ATPG fault coverage [4][5].

4. METHODOLOGY

The methodology adopted for this project follows a complete ASIC design and test flow, starting from RTL design and ending with scan-based test validation. The process is divided into sequential stages, each contributing to the structural and functional validation of the FIR filter design.

4.1. RTL Design and Testbench Development

The design of a 7-tap Finite Impulse Response (FIR) filter was implemented using Verilog HDL. The filter accepts an 8-bit input (x_in) and performs convolution with 7 fixed coefficients, producing a wider

output (y_out) due to accumulated bit growth. The design includes synchronous reset and a test mode signal to support later scan insertion. A comprehensive testbench (fir_filter_tb.v) was created to generate a 100 MHz clock, apply various input sequences, and activate test mode selectively. The testbench simulates both normal operation and constant-value testing under scan-enabled conditions.

4.2. RTL Simulation

Functional verification was conducted using ModelSim. The waveform output was examined to ensure correct propagation of filtered values through the tape delay line and to observe the output delay corresponding to the number of filter stages. This simulation confirmed that the FIR filter design operates correctly under various input conditions [1].

4.3. RTL Synthesis using Synopsys Design Compiler

Post-verification, the RTL was synthesized using Synopsys Design Compiler targeting the Nangate 45nm Open Cell Library. A synthesis constraint file (.sdc) was provided to define the clock period and timing requirements. The output netlist (fir_filter_7tap.vg) was analyzed for area, power, and timing using report_area, report_power, and report_timing commands. The synthesized design demonstrated low area usage (~552.48 μm²), acceptable power consumption (~247.8 μW), and met timing with positive slack (~2.59 ns).

4.4. Post-Synthesis Simulation

The gate-level netlist generated from synthesis was simulated again in ModelSim to verify functional equivalence with the RTL design. The waveform output confirmed that synthesis preserved the intended behavior. Signal delays were observed, and the output latency remained consistent with the FIR pipeline architecture.

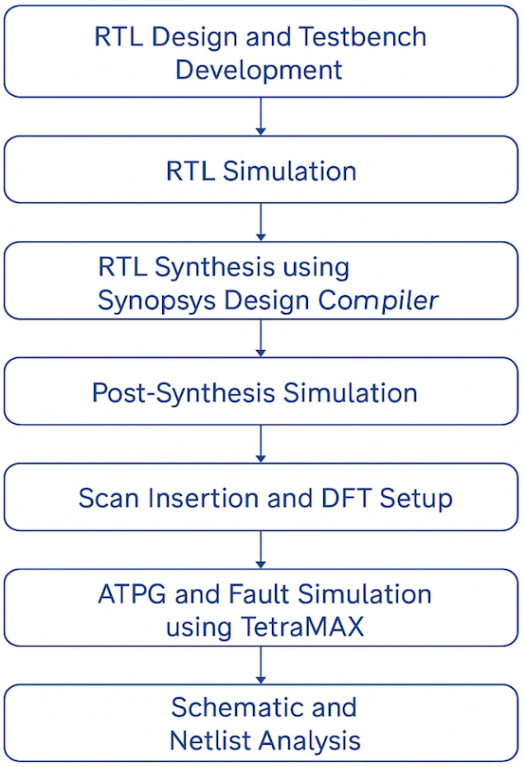


Figure 4: ASIC FLOW

4.5. Scan Insertion and DFT Setup

Design-for-Testability (DFT) was implemented by inserting scan chains into the synthesized netlist using Synopsys Design Compiler. Using appropriate DFT commands (set_scan_configuration, insert_dft), flip-flops were replaced with scan-enabled variants and chained serially to form a scan path. A new gate-level netlist (fir_filter_7tap_scan.vg) was generated, along with updated reports to validate scan logic insertion. The scan-ready design maintained original timing and functionality while being enhanced for testability.

4.6. ATPG and Fault Simulation using TetraMAX

The scan-inserted netlist was imported into Synopsys TetraMAX for ATPG (Automatic Test Pattern Generation). The tool generated scan vectors to detect stuck-at and transition faults. A fault coverage report showed that 904 out of 922 faults were detected, and the remaining 18 were marked as untestable due to redundancy. This yielded a fault coverage of 100.00%, indicating successful scan insertion and complete controllability/observability of the design.

4.7. Schematic and Netlist Analysis

The scan-inserted netlist was also visualized in Design Vision and TetraMAX. The schematic view confirmed correct structural representation of the scan chain and datapath. Signal transitions, flip-flop connections, and logic block structures were verified to ensure correctness.

5. RESULTS AND VERIFICATION

5.1 Pre-synthesis Verification

Initially, it's crucial for anyone working with digital designs to know how to use Hardware Description Languages. This knowledge allows us to simulate, verify, synthesize, and test our designs properly. In my case, I compiled and simulated the design using Verilog in the ModelSim - INTEL FPGA STARTER EDITION 10.5b (Quartus Prime 18.1) simulation environment. This tool is accessible through the Fresno State Apporto Virtual Lab.

To begin working with ModelSim for simulating my design, here's what I did:

1. First, access the Fresno State virtual lab by navigating to https://fresnostate.apporto.com/.

2. After logging in with Fresno State credentials, search for ModelSim and launch it as shown in Figure below.

Figure 5: Accessing ModelSim from Apporto Virtual Lab

3. I have ModelSim already installed in my desktop,so I skipped the above steps.

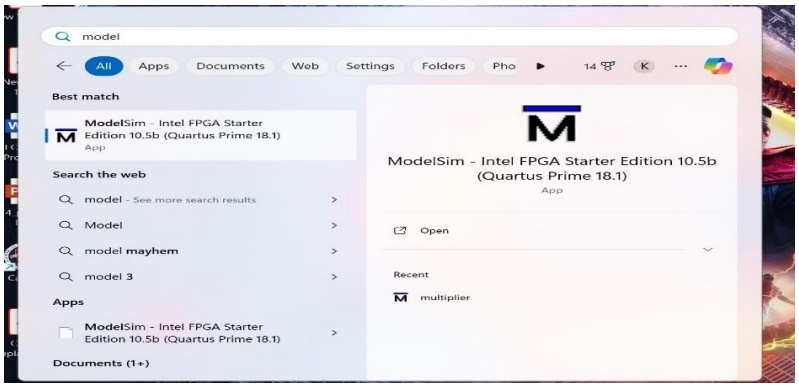


Figure 6: Accessing ModelSim by installing

4. Once in ModelSim, Open File > New > Project to open the project creation dialog box as shown in Figure. Here, browse to select the folder where you want to save project and then clicked 'OK' to create it.

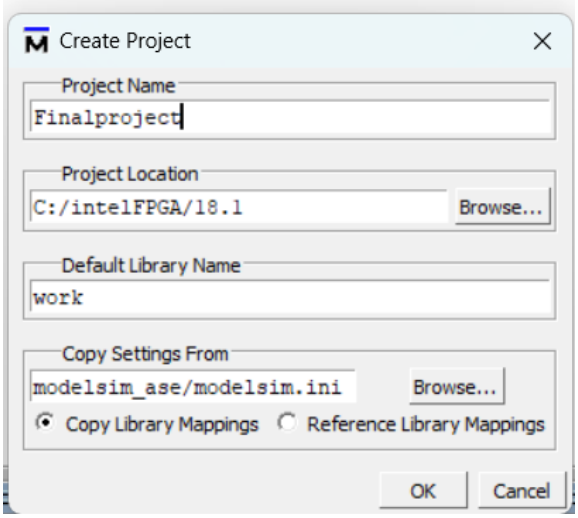


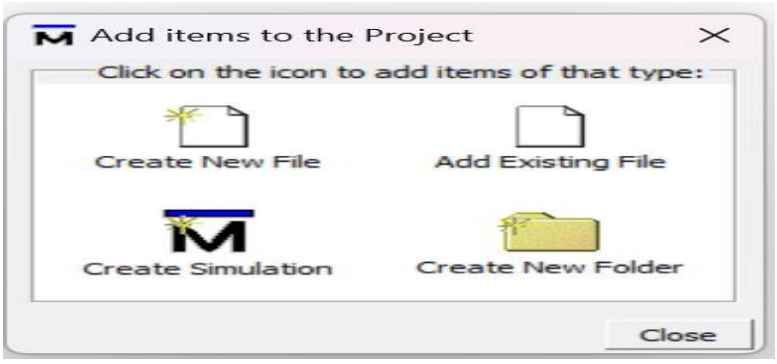
Figure 7: Creating Project

5. Here's how I proceeded with the design and testing of the fir filter using Verilog in ModelSim:

5.1 Writing the Verilog Code: start by writing the Verilog code for the fir filter encoder in the newly created project file.

5.2 Preparing the Test Bench: To ensure the design worked as intended, I prepared a test bench. This test bench was crucial for validating the functionality of the priority encoder.

5.3 Adding Files to the Project: For managing the project files, right click in the project area, selected 'Create New File and then created the Top Module and testbench files. This process is illustrated in Figures 5,6 .



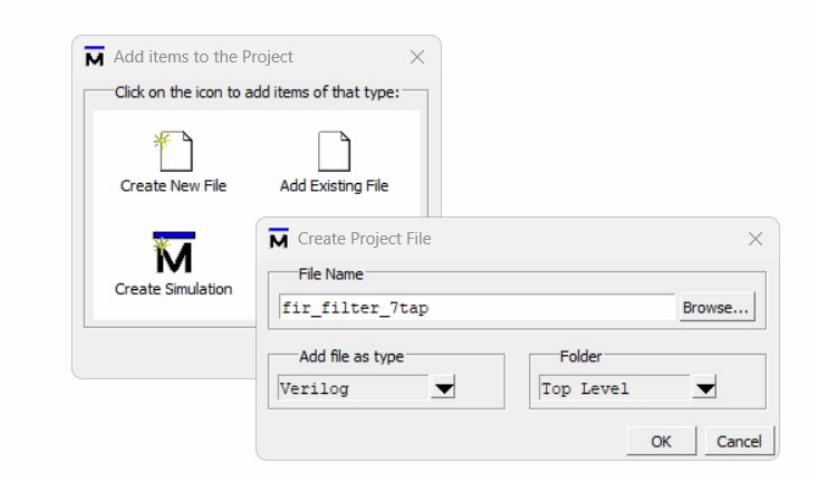


Figure 8: Creating file for code

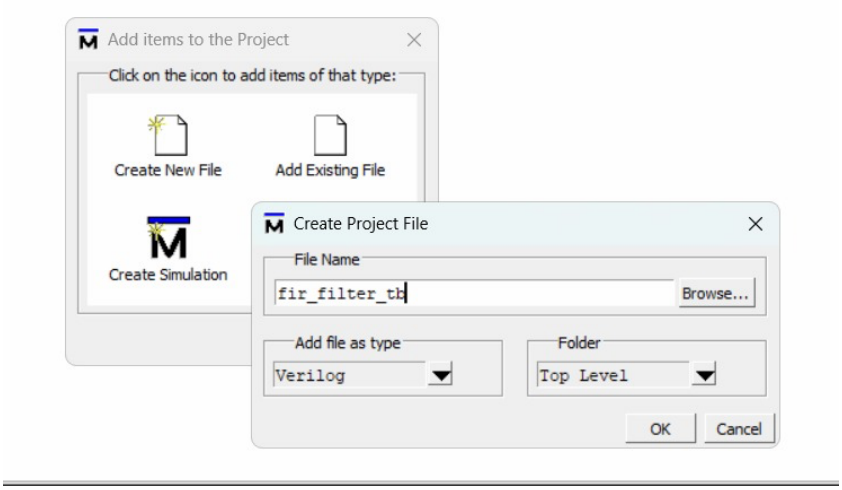


Figure 9: Creating File for Testbench

5.4 Compiling the Design and Test Bench: After setting up the test bench and design code, compile both. The successful compilations were confirmed in the transcript window, as depicted in Figures 10 and 11.

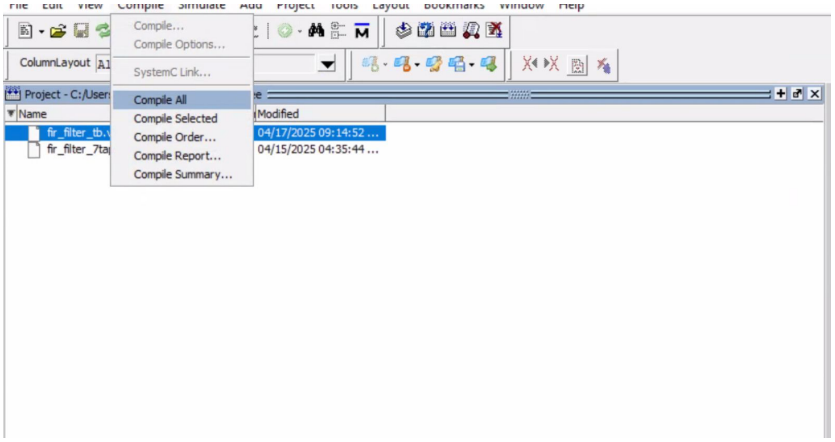


Figure 10: Compilation of the codes

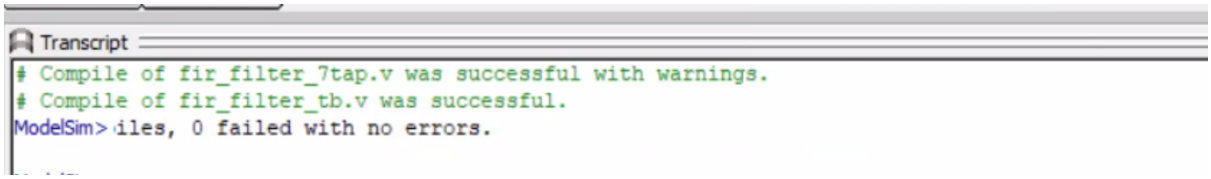


Figure 11: Successfully Compiled

5.5 Simulating the Test Bench: Next, run the simulation using the test vectors written in the test bench. This step was vital for verifying the fir filter functionality, shown in Figure 12 and Figure 13.

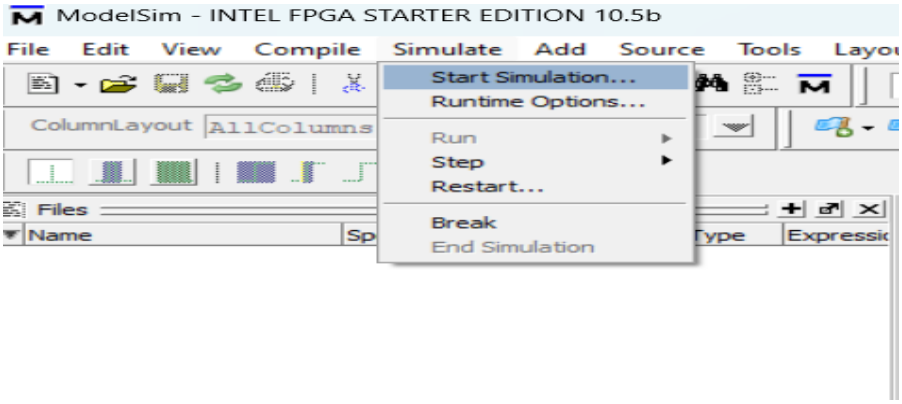


Figure 12: Simulation

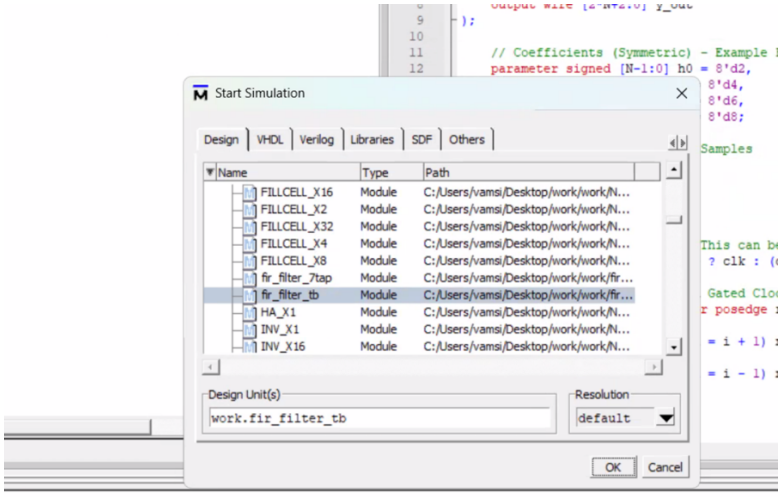


Figure 13: search for the main module name

5.6 Adding Waveforms: To visualize the operation, select the necessary objects and clicked 'Add Wave', as shown in Figure 11. Then, execute 'run -all' in the Transcript window to populate the Wave window, shown in Figure 14.

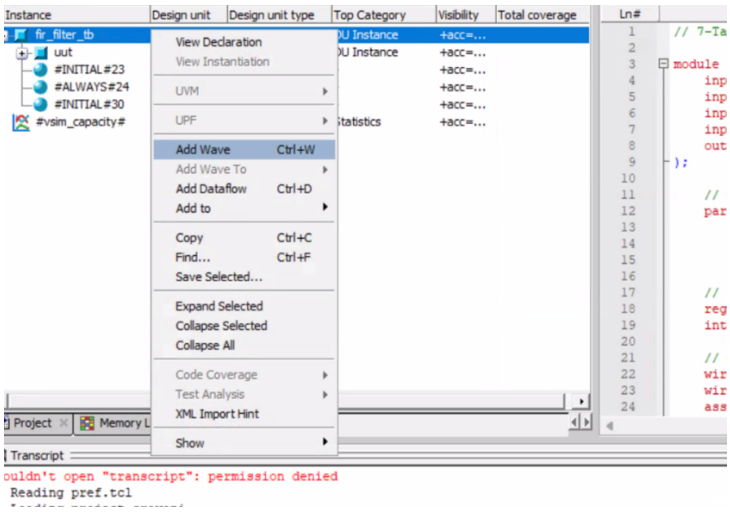


Figure 14: Adding wave forms

5.7 Reviewing the Simulation Results: When prompted by a pop-up window to finish, choose "No". Select the Radix i.e, Binary. The simulation waveform, along with the test vectors from the test bench, was displayed in Figure, confirming the proper operation.

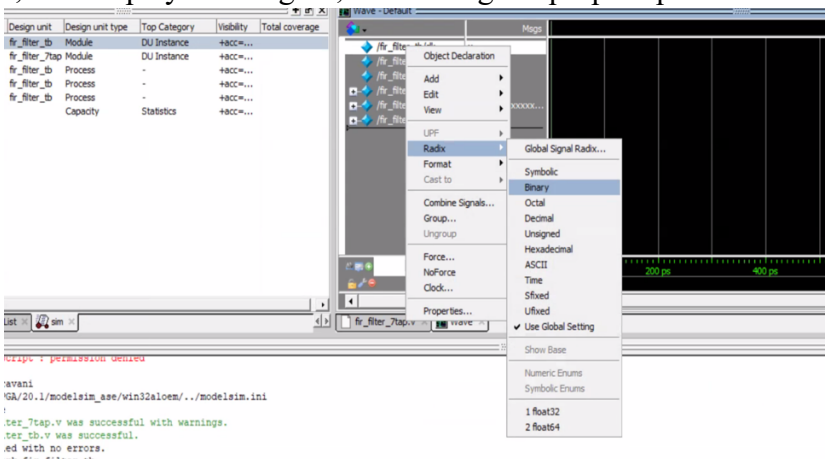


Figure 15: Selecting the Binary format.

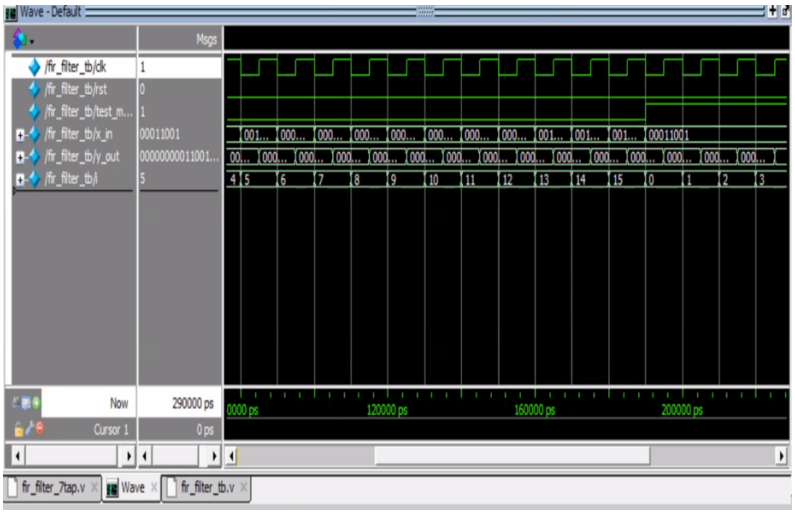


Figure 16: Simulated Waveforms

Closing the Simulation: To exit the simulation, I entered **'quit -sim'** in the command tab.

This entire process, termed "Pre-synthesis Verification," ensured that the design functioned correctly before moving on to the synthesis stage, which we will discuss in the next section.

5.2 SYNTHESIS OF FIR FILTER USING SYNOPSYS DESIGN COMPILER

After the successful verification of the fir filter design through simulation, the next step was to synthesize the functionally validated design into actual hardware logic. This process was performed using Synopsys hosted on an apporto virtual lab, which is particularly suited for such tasks due to its robust processing capabilities and compatibility with synthesis tools.

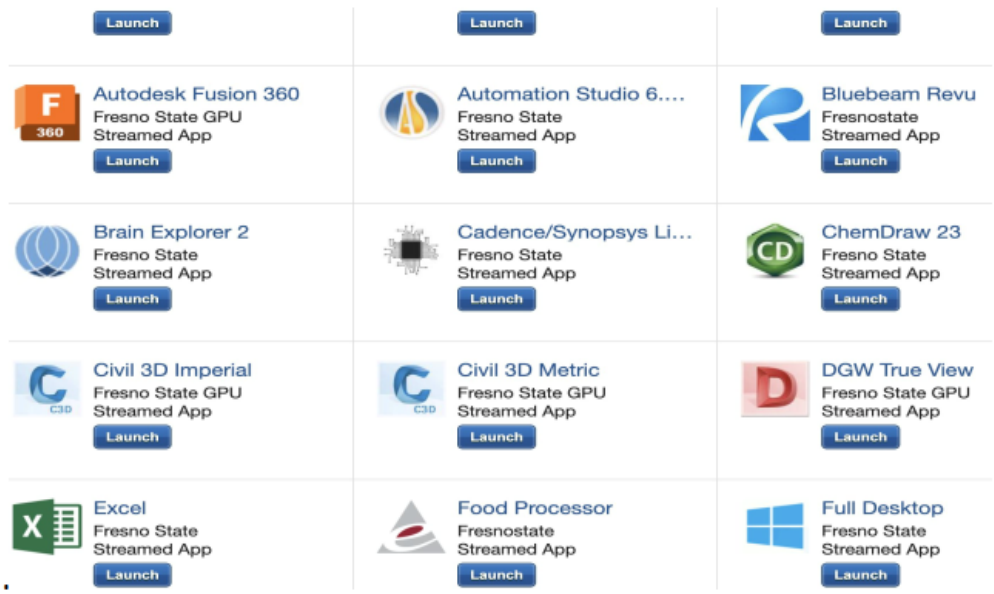


Figure 17: Launching the cadence/synopsys application

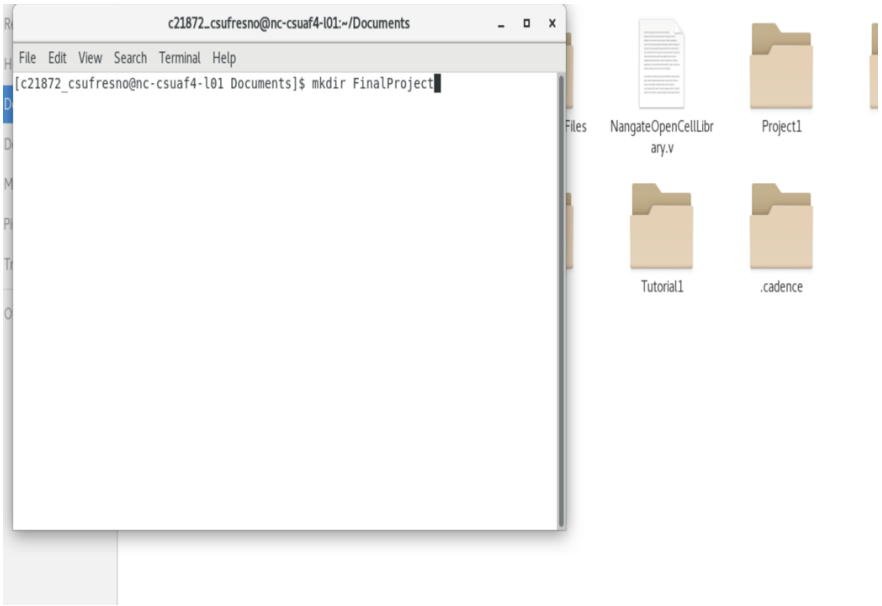


Figure 18: create new folder using above command

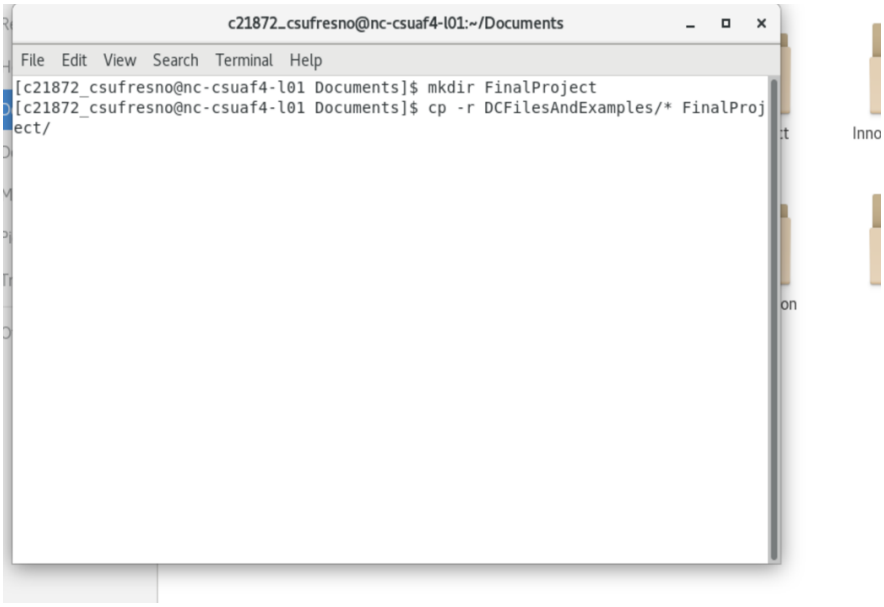


Figure 19: copied synopsys dc compiler files

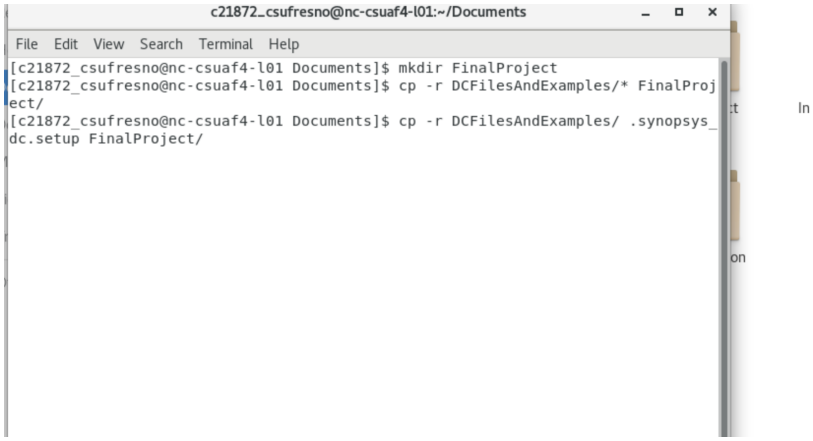


Figure 20: setting up the synopsys dc compiler environment by above command

1. **File Preparation and Initial Setup:** Update the abc.vfs and abc.vfv files to reflect only the top module, which is the main. v file for the project. Using the 'Save As' option, rename these files to match the top module name, ensuring that all references were consistent and accurate. The original files were substituted with our custom files, as demonstrated in Figures 21 to figure 25.

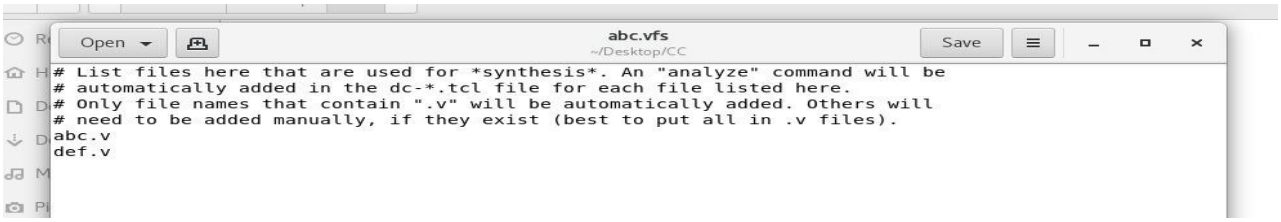


Figure 21: original vfs File

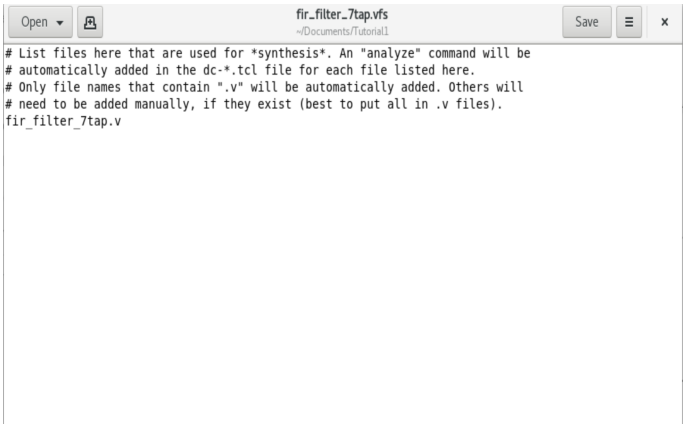


Figure 22: Modified vfs file

Figure 23: Original vfv file

Figure 24: Modified vfv file

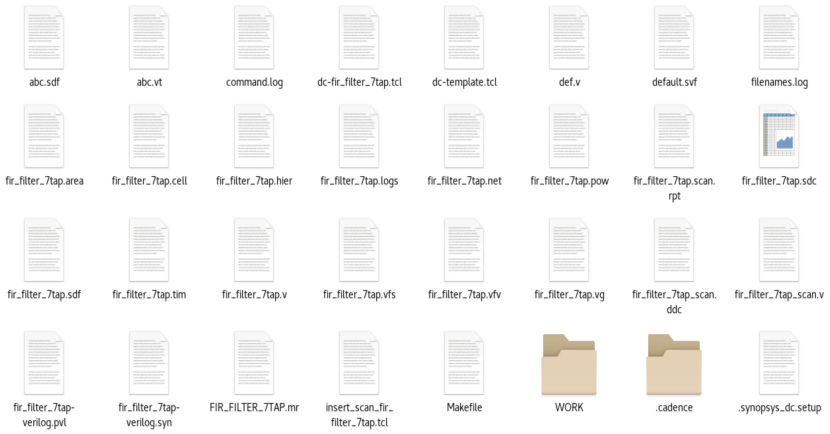


Figure 25: All files in folder

With the necessary files set and saved under the correct names, the setup for the synthesis was complete. This preparation was crucial for a smooth transition into the actual synthesis process.

2. Conducting the Synthesis:

The synthesis of the top module was carried out in the following steps:

Step 1: Accessing the Terminal To begin the synthesis process, I accessed the terminal window within the virtual environment. This was done by right-clicking in the designated area and selecting the appropriate option.

Step 2: Executing the Synthesis Command In the terminal window, I entered the specific command necessary to initiate the synthesis. This command was crucial as it triggers the conversion of the Verilog code into a synthesized gate netlist. It was important to type this command accurately to avoid any syntax errors, as illustrated in Figure 26.

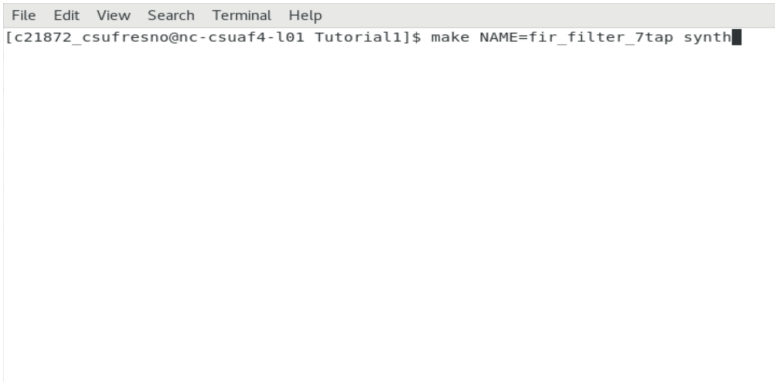


Figure 26: giving synth RUN

Step 3: Starting the Synthesis After entering the command, I pressed the enter key to start the synthesis process, as detailed in Figure. This action commenced the translation of the Verilog code into a gate netlist, an essential step for analyzing the design further.

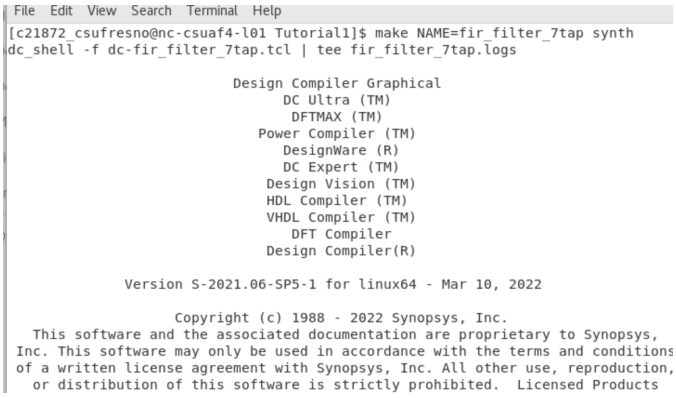


Figure 27: Dc shell run started

6. **Synthesis Completion and Analysis:** Once the synthesis was initiated, the duration of the process depended on the complexity of the design and the processing power of the computer.

Upon completion, the synthesis reports were automatically saved in the current directory. These reports provided valuable data on area and power consumption, crucial for evaluating the efficiency of the design. The results of the synthesis, including the synthesized gate netlist and analysis reports, were stored as follows:

- (your top module name).vg: This file contains the gate netlist. Example: fir_filter_7tap.vg
- (your top module name).area: This file holds the results of the area analysis. Example: fir_filter_7tap.area
- (your top module name).pow: This file details the power analysis outcomes. Example: fir_filter_7tap.pow

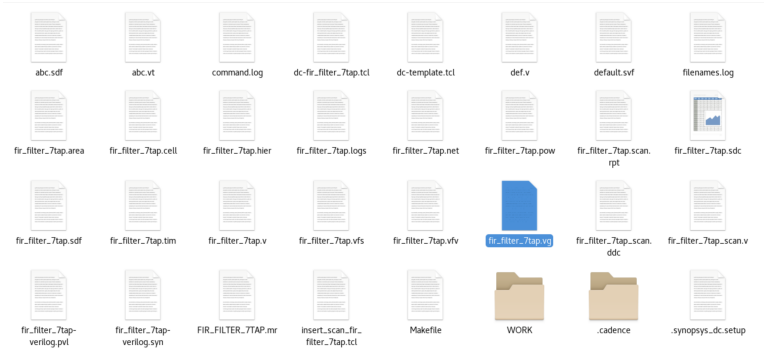


Figure 28: vg file generated

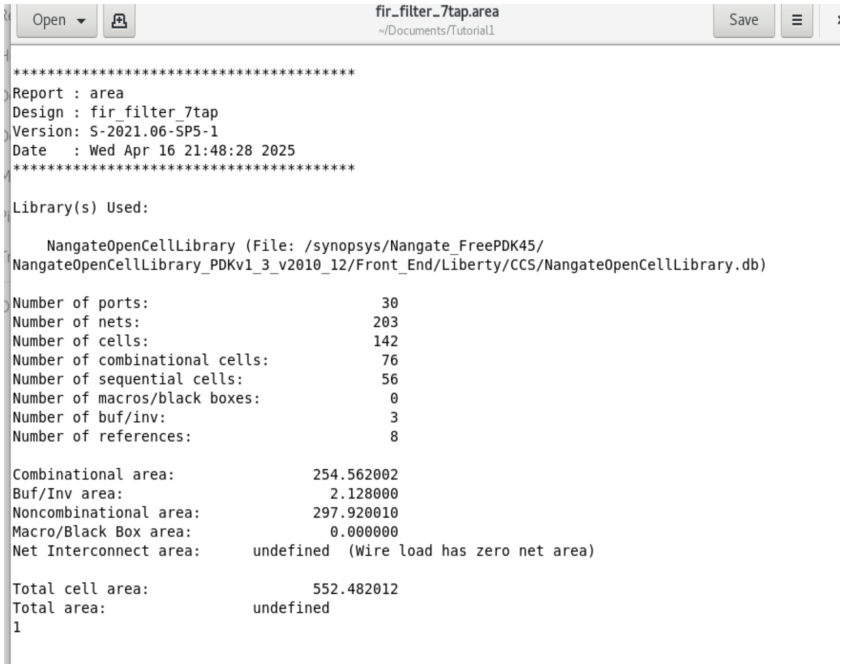
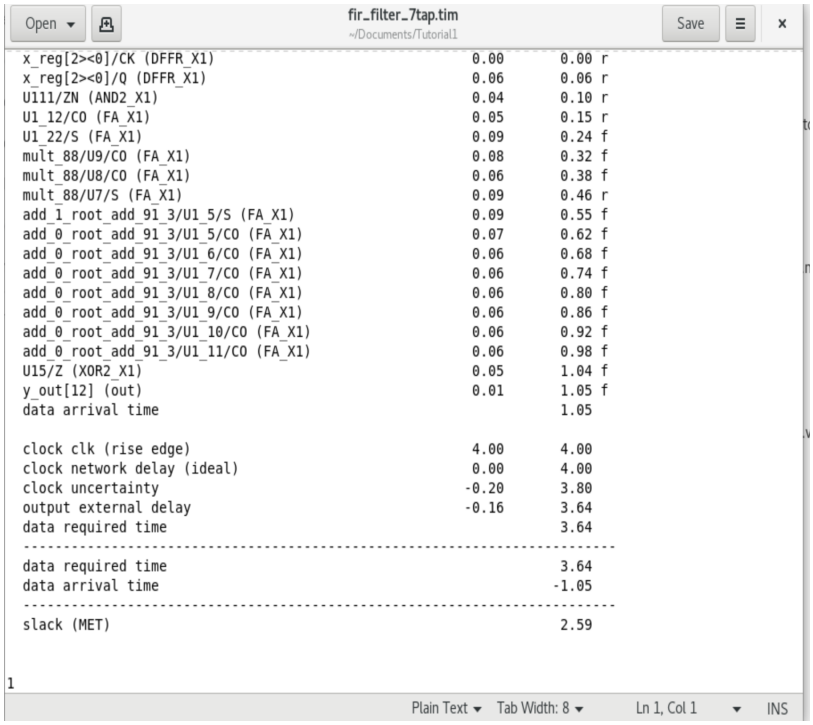


Figure 29: area File



fir_filter_7tap.tim
~/Documents/Tutorial1

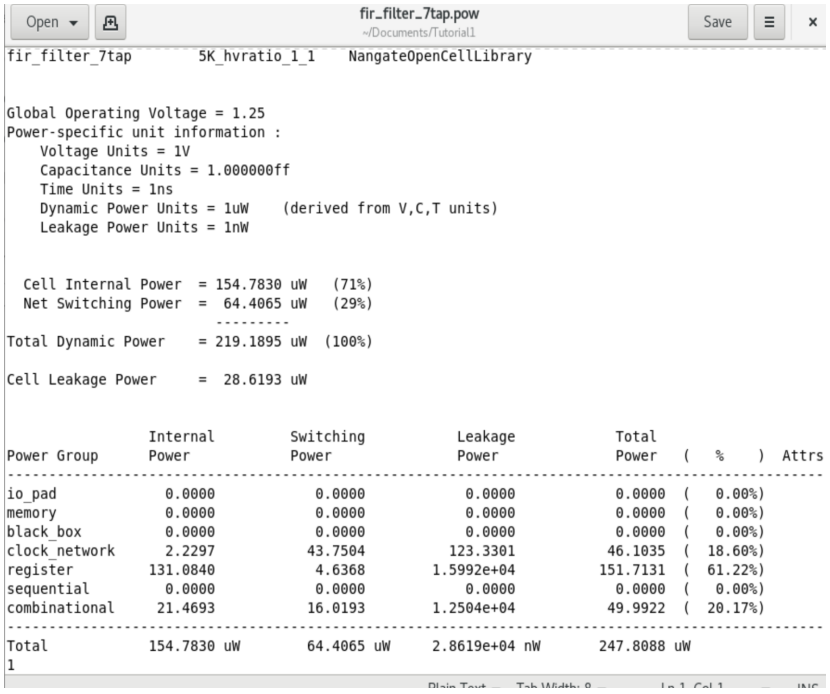
```
  x_reg[2><0]/CK (DFFR_X1)                                0.00       0.00 r
  x_reg[2><0]/Q (DFFR_X1)                                 0.06       0.06 r
  U111/ZN (AND2_X1)                                       0.04       0.10 r
  U1_12/CO (FA_X1)                                        0.05       0.15 r
  U1_22/S (FA_X1)                                         0.09       0.24 f
  mult_88/U9/CO (FA_X1)                                   0.08       0.32 f
  mult_88/U8/CO (FA_X1)                                   0.06       0.38 f
  mult_88/U7/S (FA_X1)                                    0.09       0.46 r
  add_1_root_add_91_3/U1_5/S (FA_X1)                      0.09       0.55 f
  add_0_root_add_91_3/U1_5/CO (FA_X1)                     0.07       0.62 f
  add_0_root_add_91_3/U1_6/CO (FA_X1)                     0.06       0.68 f
  add_0_root_add_91_3/U1_7/CO (FA_X1)                     0.06       0.74 f
  add_0_root_add_91_3/U1_8/CO (FA_X1)                     0.06       0.80 f
  add_0_root_add_91_3/U1_9/CO (FA_X1)                     0.06       0.86 f
  add_0_root_add_91_3/U1_10/CO (FA_X1)                    0.06       0.92 f
  add_0_root_add_91_3/U1_11/CO (FA_X1)                    0.06       0.98 f
  U15/Z (XOR2_X1)                                         0.05       1.04 f
  y_out[12] (out)                                         0.01       1.05 f
  data arrival time                                                  1.05

  clock clk (rise edge)                                   4.00       4.00
  clock network delay (ideal)                             0.00       4.00
  clock uncertainty                                      -0.20       3.80
  output external delay                                  -0.16       3.64
  data required time                                                 3.64
  --------------------------------------------------------------------------
  data required time                                                 3.64
  data arrival time                                                 -1.05
  --------------------------------------------------------------------------
  slack (MET)                                                        2.59


1
```

Plain Text Tab Width: 8 Ln 1, Col 1 INS

Figure 30: Timing File



fir_filter_7tap.pow
~/Documents/Tutorial1

```
fir_filter_7tap        5K_hvratio_1_1    NangateOpenCellLibrary


Global Operating Voltage = 1.25
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  = 154.7830 uW   (71%)
  Net Switching Power  =  64.4065 uW   (29%)
                         ---------
Total Dynamic Power    = 219.1895 uW  (100%)

Cell Leakage Power     =  28.6193 uW


                Internal         Switching           Leakage            Total
Power Group     Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000            0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000            0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000            0.0000  (   0.00%)
clock_network      2.2297          43.7504          123.3301           46.1035  (  18.60%)
register         131.0840           4.6368        1.5992e+04          151.7131  (  61.22%)
sequential         0.0000           0.0000            0.0000            0.0000  (   0.00%)
combinational     21.4693          16.0193        1.2504e+04           49.9922  (  20.17%)
--------------------------------------------------------------------------------------------------
Total            154.7830 uW       64.4065 uW     2.8619e+04 nW       247.8088 uW
1
```

Figure 31: Power File

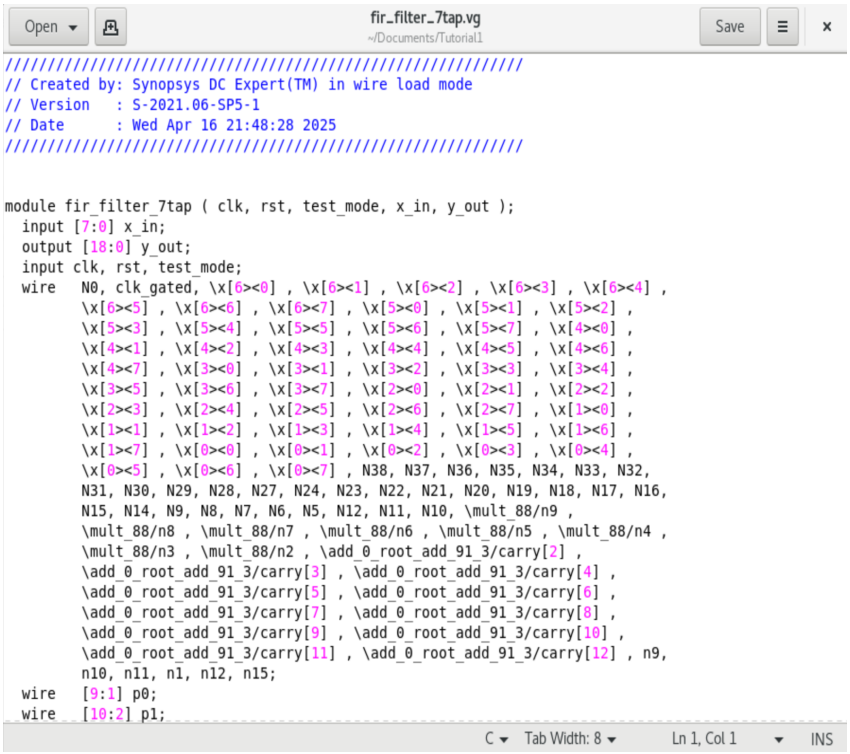


Figure 32: Netlist vg file

The synthesis phase was critical for confirming that the design met all technical requirements and was ready for further testing and eventual implementation.

5.3 SCAN INSERTION:

STEP: Create a new folder for scan insertion and copy all files of pre-synthesis

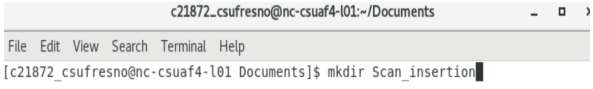


Figure 33: Creating new folder

STEP 2: Create a new tcl file and write code for scan insertion

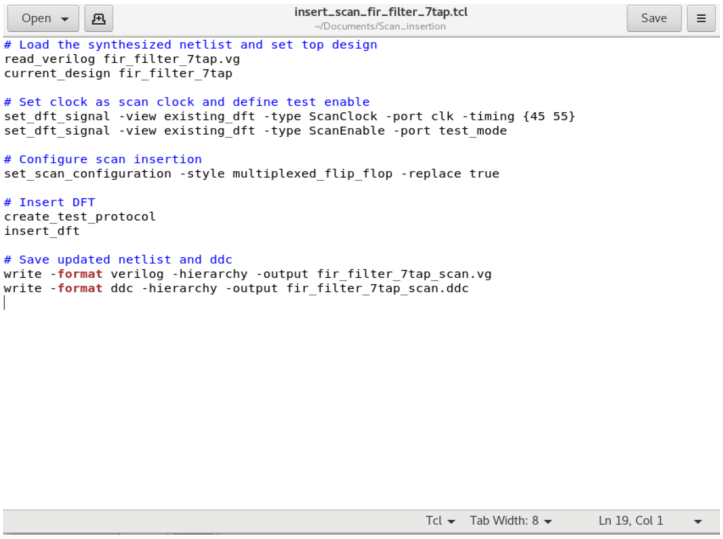


Figure 34: TCL file script

STEP 3: from the folder give the terminal and give the command dc_shell

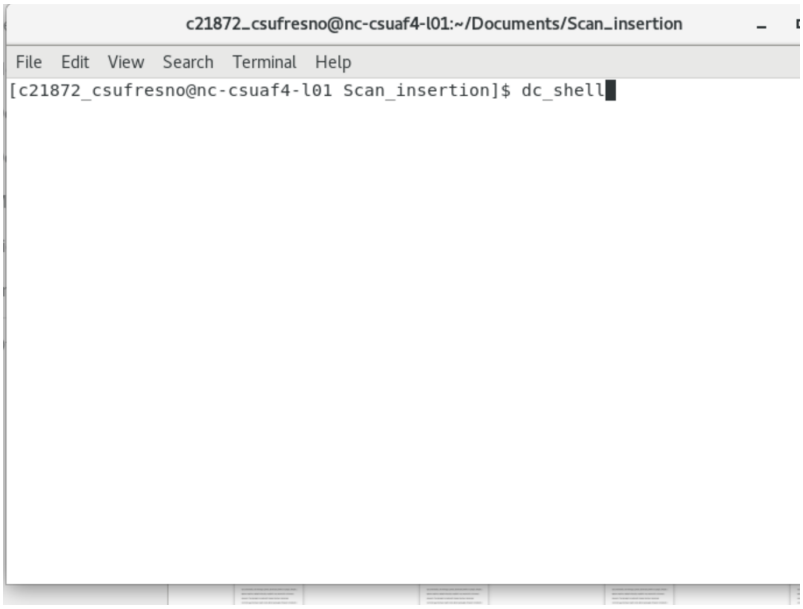


Figure 35: command dc_shell

STEP 4: Next give command source scan_insertion_fir_filter_7tap.tcl

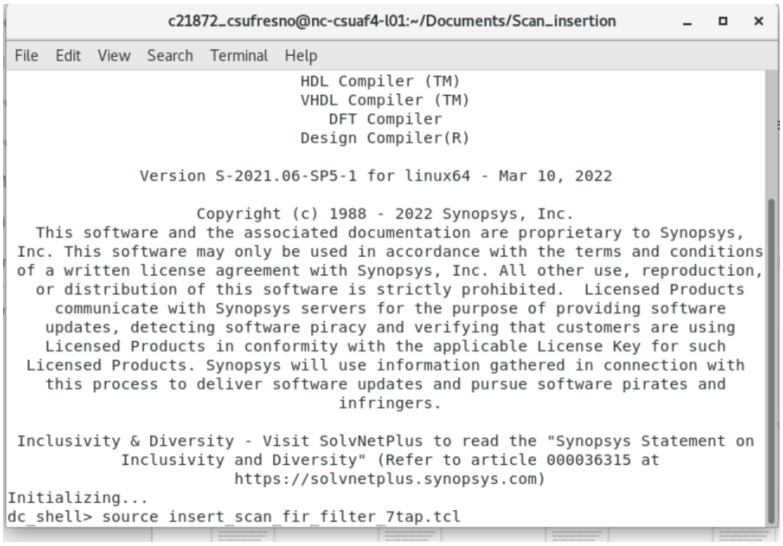


Figure 36: dc_shell environment

STEP 5: We can observe the scan_insertion_fir_filter_7tap.vg file

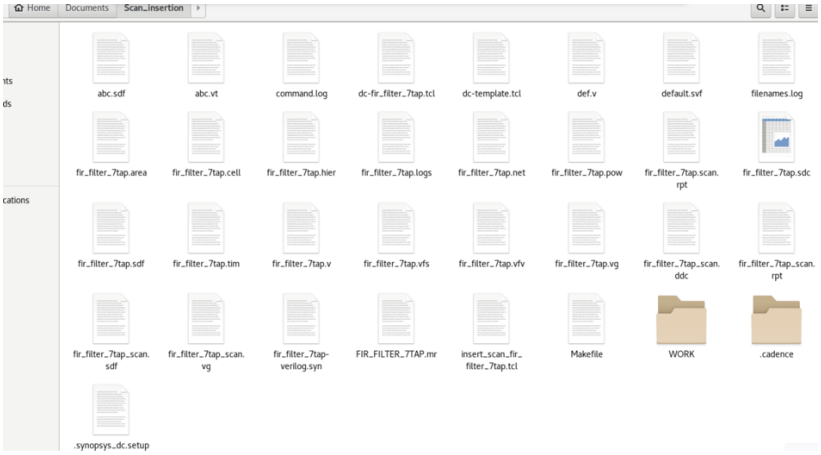


Figure 37: Scan insertion

STEP 6: This is the required scan insertion netlist file

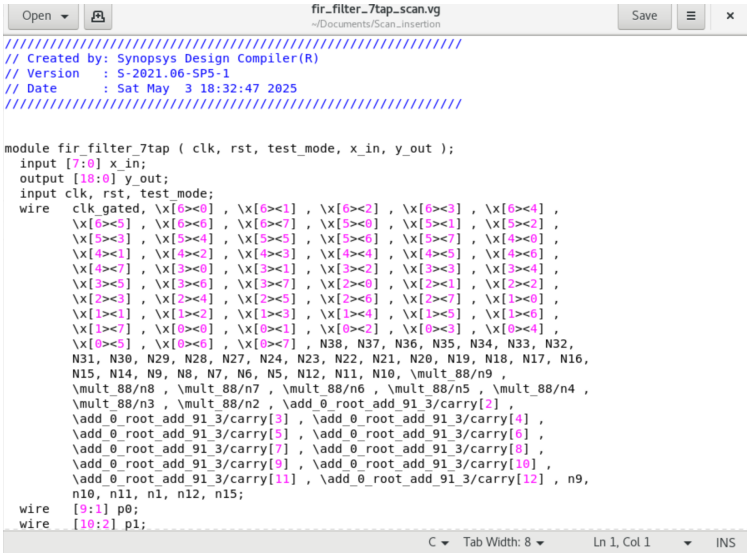


Figure 38: Scan insertion netlist file

5.4 Post-synthesis Verification

1. **Integrating the Library File**: The method for incorporating the library file is outlined in the steps that follow, ensuring that all necessary components are in place for successful test pattern generation.To integrate the library file into the project, I followed these steps:

- **Step 1:** Navigate to 'Places' and then 'Computer' **(Figure 39)**

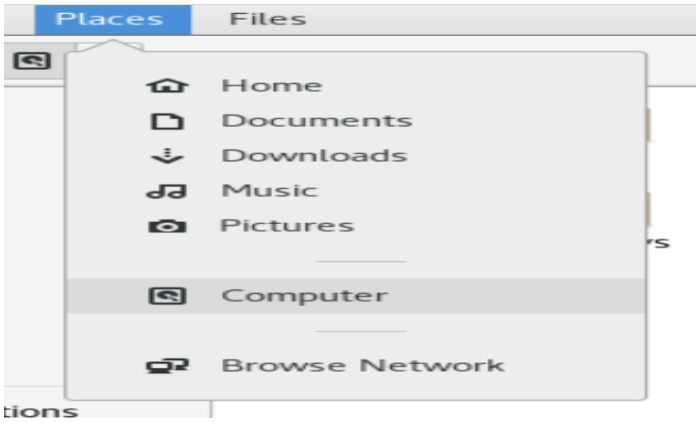


Figure 39: integrate the library file into the project

- **Step 2:** Locate the Synopsys Directory **(Figure 40)**

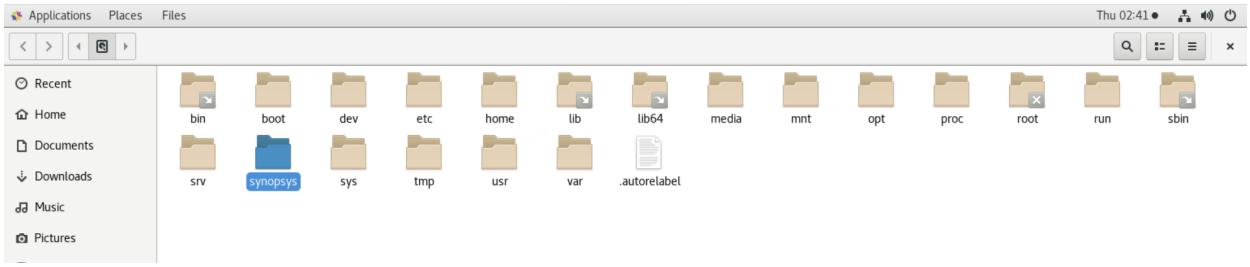


Figure 40: select synopsys

- **Step 3:** Search for Nangate_FreePDK45 **(Figure 41)**

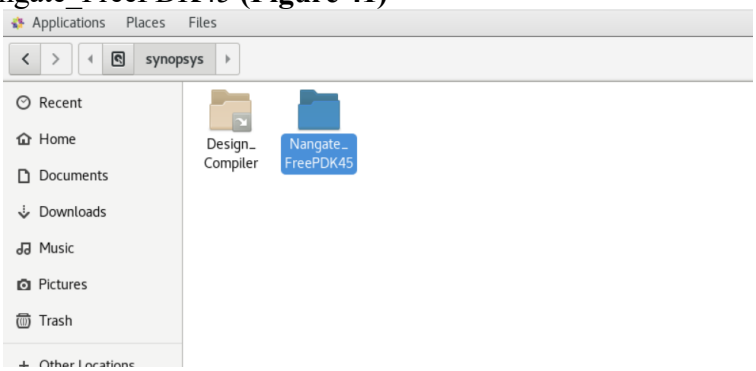


Figure 41: Nangate_FreePDK45

- **Step 4 :** Access the Front_End Directory **(Figure 42)**

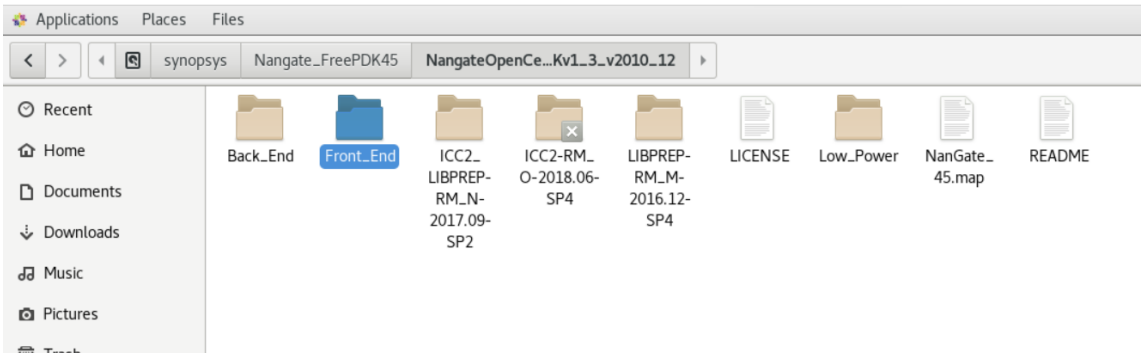


Figure 42: select front end

- **Step 5 :** Navigate to the Verilog Folder **(Figure 43)**

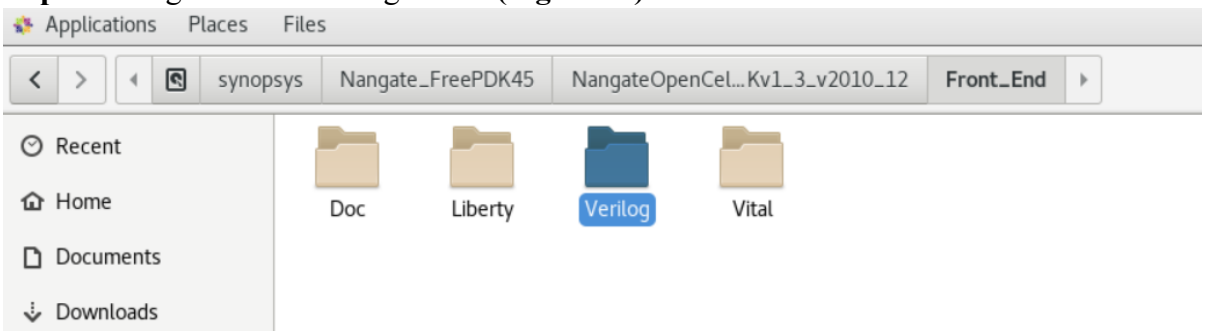


Figure 43: select verilog

- **Step 6 :** Identify and select the library file **(Figure 44)**

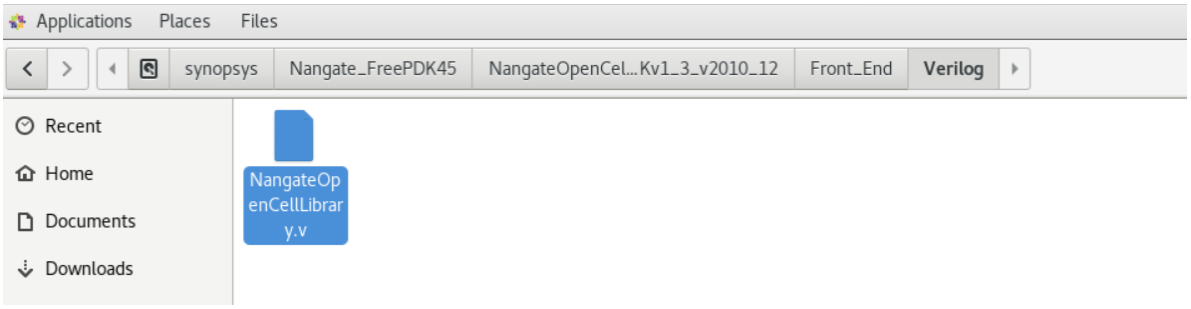


Figure 44: library file

After successfully synthesizing the FIR filter and acquiring the gate netlist, it is imperative to conduct a post-synthesis verification. This verification ensures that the synthesized design functions correctly within the specified technology parameters. This stage, referred to as "post-synthesis verification," is important for confirming that the physical implementation to the design specifications.

2. Post synthesis verification

Step 1: The first step involves repeating the procedures from the pre-synthesis simulation but now integrating the synthesized gate netlist. This involves creating a new Verilog file, saved under the name fir_filter_postsynth.v, specifically for simulating the gate netlist, as shown in Figure 45.

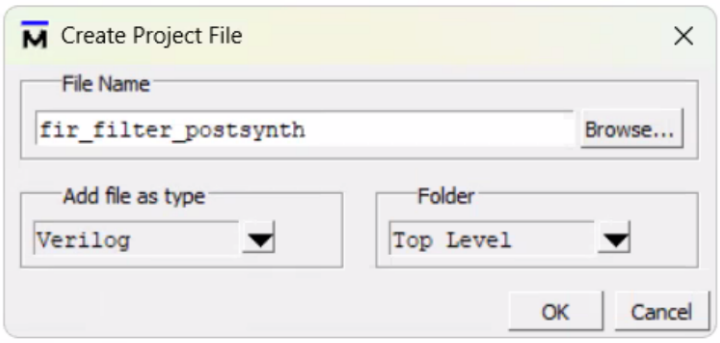


Figure 45: creating new file for post synthesis

Step 2: Incorporate the gate netlist obtained from the synthesis process into the newly created Verilog file. This file will serve as the primary input for post-synthesis simulation.

Step 3: Modify the original pre-synthesis test bench to make it compatible with the post-synthesis design.

Step 4: Compile both the post-synthesized gate netlist Verilog file and the test bench file. It is common to encounter compilation errors due to the inclusion of gates synthesized from libraries not recognized by ModelSim. To resolve these errors, integrate the NangateOpenCellLibrary.v into your project by adding the line include "NangateOpenCellLibrary.v" at the beginning of your top module file. Recompile all files within the project directory following this adjustment.

Step 5: With the files compiled, run a simulation using the test bench file to validate the design's functionality. If the simulator reports an "Error loading design", check the library file for the absence of a time unit or time precision. Correct this by adding the line **timescale 1ns/1 ps** to the library file settings, ensuring the simulator accurately interprets the timing of the design.

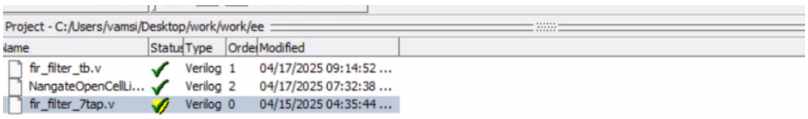


Figure 46: successfully compiled.

Step 6: After resolving any initial errors, run the simulation again to ensure that the design operates as expected. Reintroduce waveforms for post-synthesis verification, to the pre-synthesis phase. The resulting waveform, depicted in Figure 47, is crucial for inspecting the functionality under simulated gate delays.

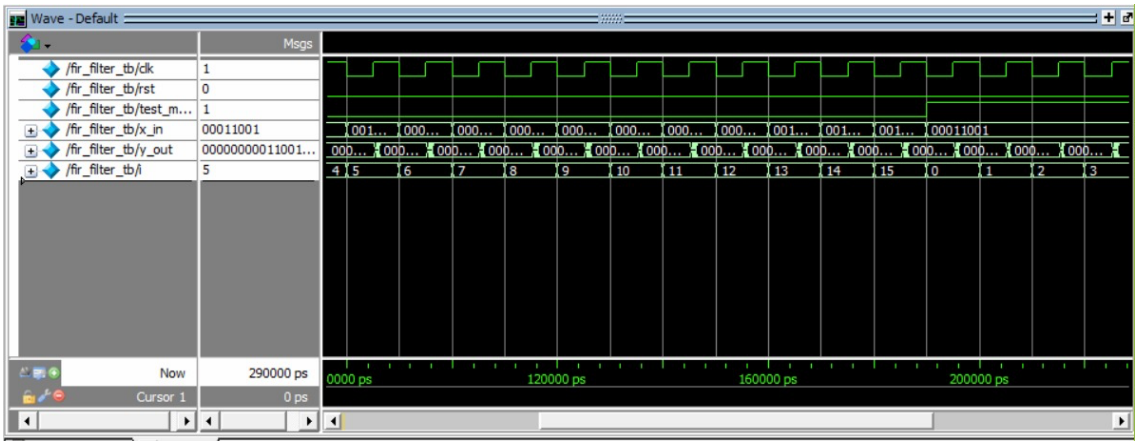


Figure 47: Functional Verification of the Synthesized Gate Netlist

Observations confirm that the design remains functional even with the inclusion of actual gate delays. If the test coverage from the ATPG tool Synopsys TetraMax does not reach 100%, the design must be revised, starting again from the RTL Design (Pre-synthesis stage).

3.5. GENERATING OF TEST PATTERNS WITH TETRAMAX

3. **Test File Preparation:** To begin the test pattern generation process, I first prepared the necessary files. I created a new folder and placed both the netlist and library files within it, ensuring an organized workspace for the subsequent steps.

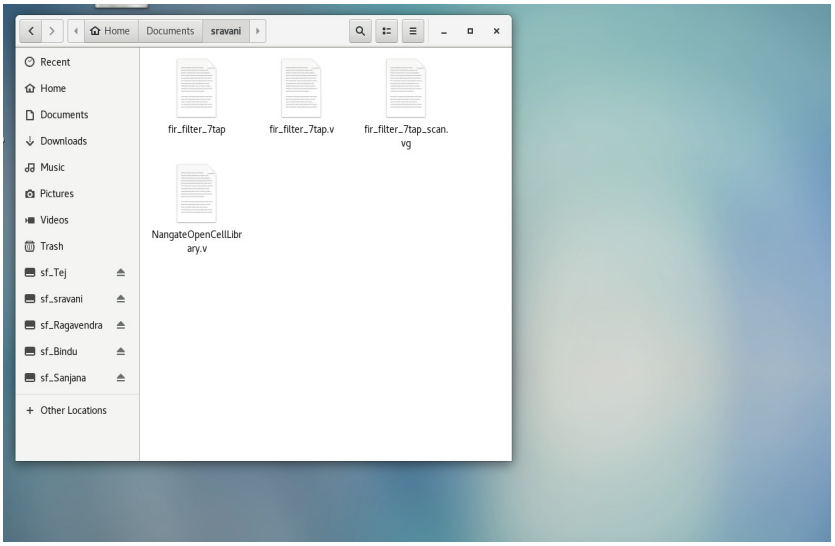


Figure 48: file In folder for post-synthesis process

4. **Development of Test Patterns**:The test pattern generation with TetraMAX involved several critical steps:

4.1 Launching TetraMAX :To begin, I opened the terminal and entered the command to launch TetraMAX as shown in Figure.

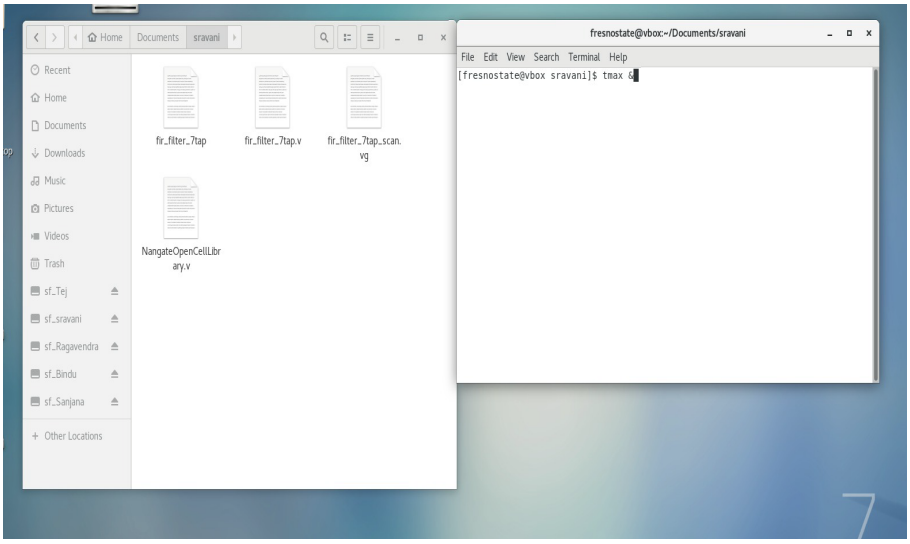


Figure 49: Executing TetraMax Command (tmax &)

4.2 Netlist Configuration: With TetraMax launched, I started by configuring the netlist, which plays a crucial role in generating test patterns. This step is shown in Figure 50, where I loaded both the netlist and corresponding library files into TetraMAX.

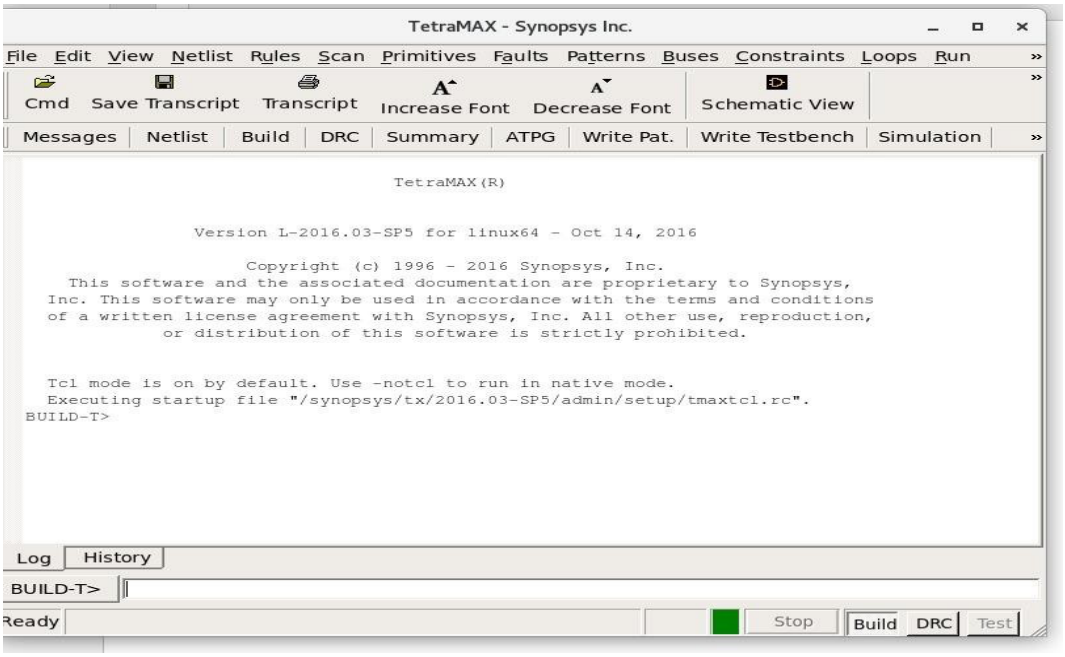


Figure 50: Main Screen of TetraMAX

I loaded the netlist through the command toolbar as depicted in Figure 51, ensuring that library modules were appropriately selected during the library file opening process.

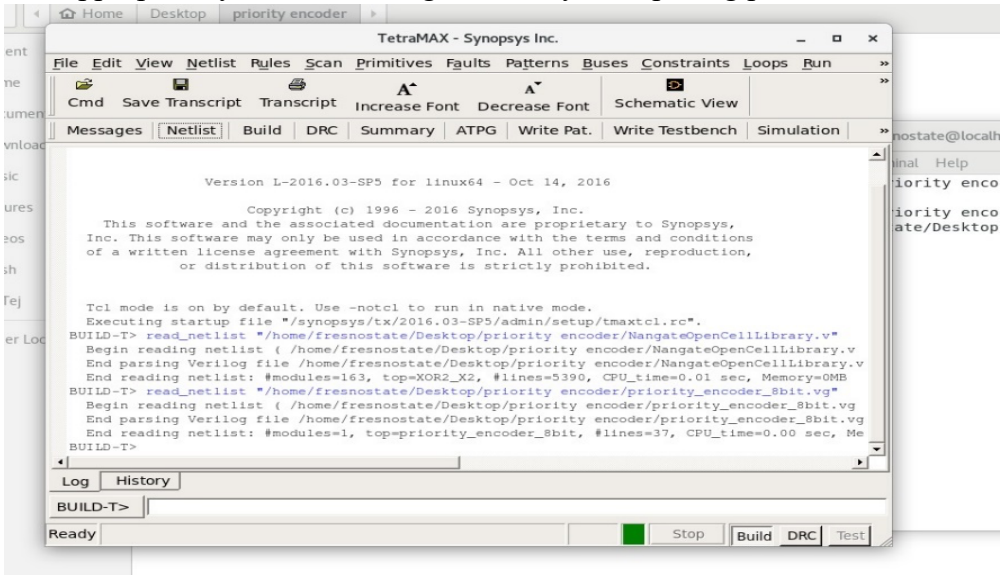


Figure 51: Loading the Netlist

Subsequent steps involved ensuring that the "library modules" option was toggled correctly for each file, as demonstrated in Figure 52.

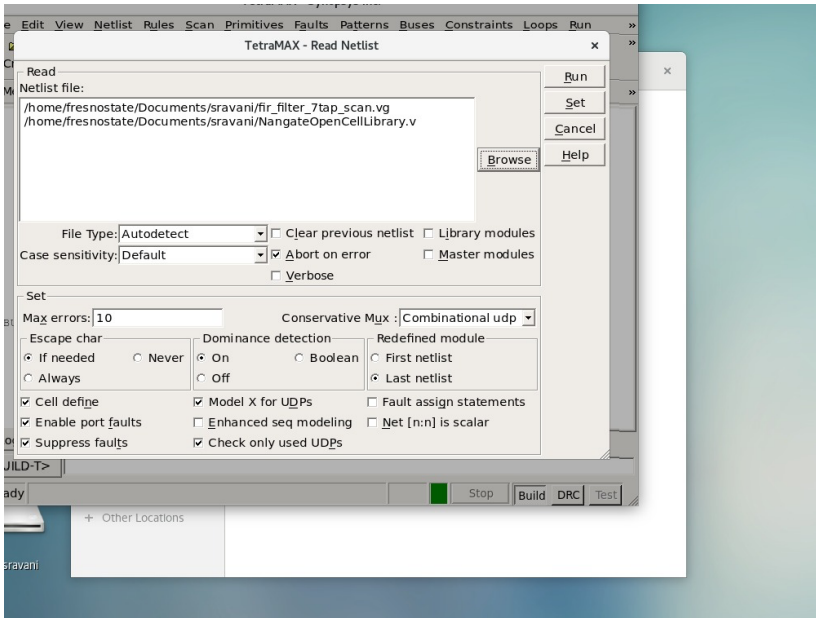


Figure 52: Ensuring 'Library Modules' Are Selected During Library File Opening

4.3 Building Setup: Once the netlist setup was completed, the next step was to build the patterns. This required ensuring that the top-level module name matched the one used in the design. The build process and results are shown in Figure 53.

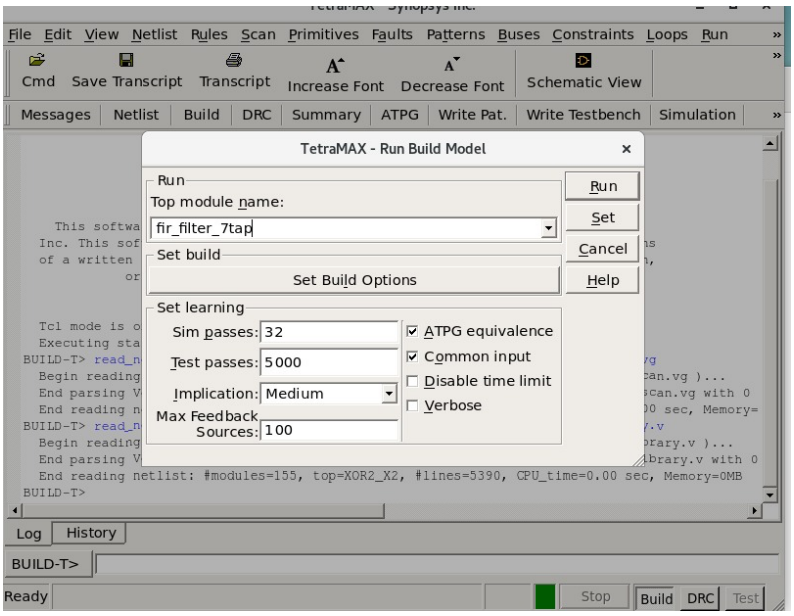


Figure 53: Inputting Top Module Name for Execution Figure

4.4 Design Rule Checking (DRC) Setup: Proceed with the Design Rule Check (DRC) in TetraMax by navigating to the "DRC" option in the command toolbar and initiating the run. The outcome of this step is essential for ensuring compliance with design standards, as shown in Figure 54.

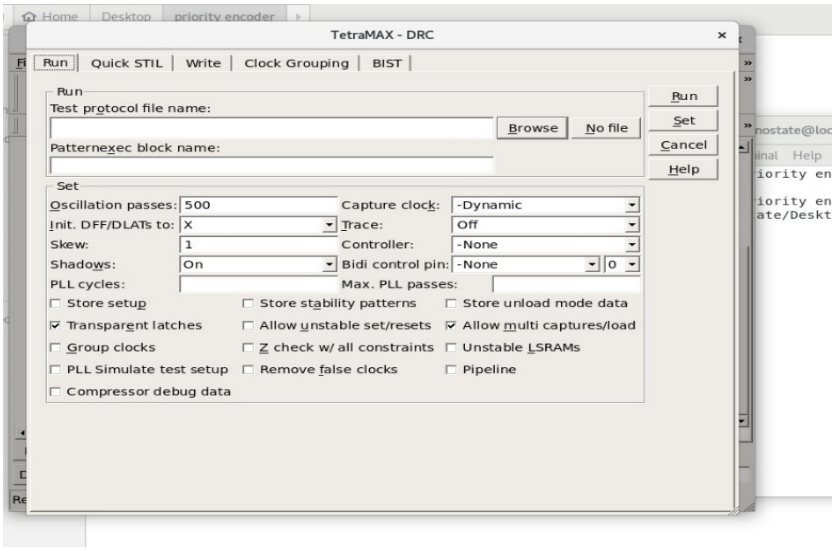
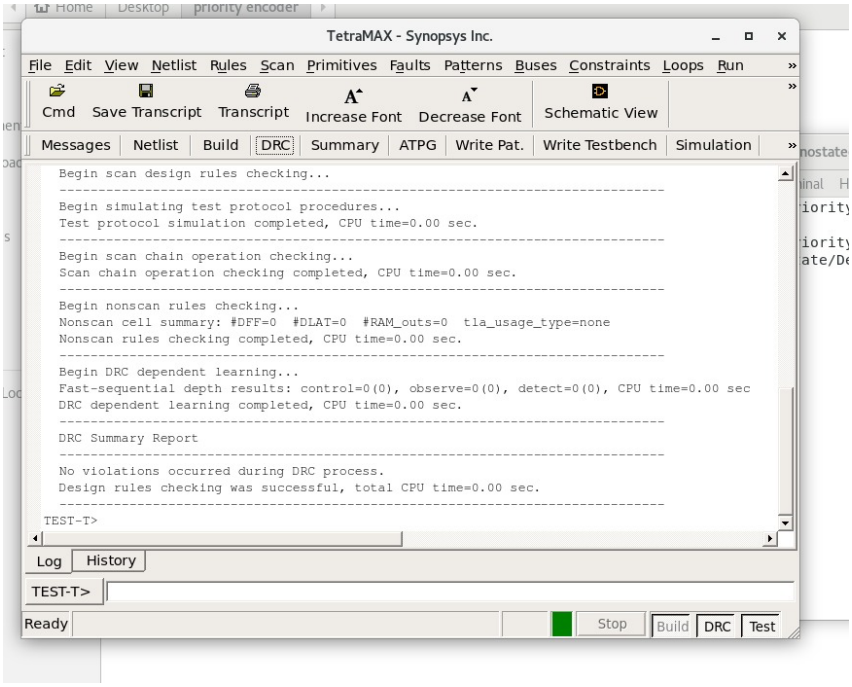


Figure 54: Initiate Run Without Specifying File Names



4.5 Automatic Test Pattern Generation (ATPG): The ATPG process is critical for ensuring the reliability of the design by identifying potential faults. To begin, the user must initialize the fault list in TetraMax, selecting the appropriate model on the 'Run ATPG' window. This includes setting various parameters such as the abort limit, coverage percentage, fault source, fault modeling, and the type of ATPG (Basic scan, Fast sequential, or Full sequential). Once these settings are configured, the user can execute the test pattern generation by selecting the "AUTO" option in ATPG and monitoring the output in the transcript window.

For this tutorial, you would:

1. Click on ATPG, and in the pop-up window, set the coverage to 100%.
2. Click on "Add all faults" to set the fault source.
3. Choose the Stuck fault model.
4. Click on "Auto" to start generating the test patterns. This process is depicted in Figure 55.

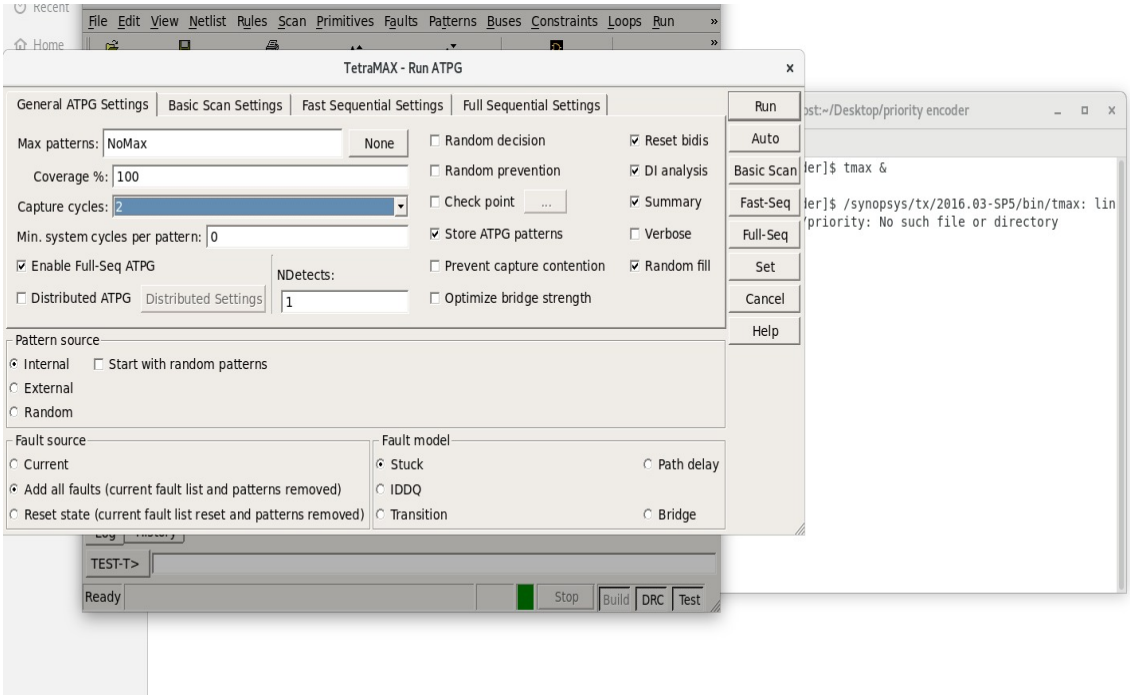


Figure 55: Configuring Fault Settings and Initiating ATPG

Following the ATPG test mode, the transcript window displays the number of faults detected, the number of test patterns generated, and CPU usage data, as shown in Figure 56.

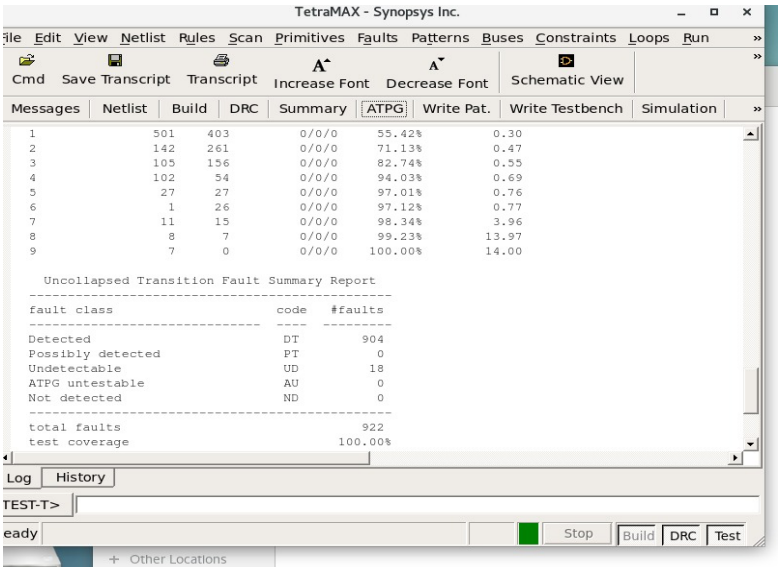


Figure 56: Transcript Window Showing Fault Coverage and Test Patterns

4.6 Writing Test Patterns: To write the test patterns:

1. Navigate to the "Write Patterns" option in the command toolbar in TetraMax's main window.
2. Specify the pattern file name and the format in which the patterns are to be written.
3. Click "OK" to create the file and write all the test patterns into it. This action is illustrated in Figure 57.

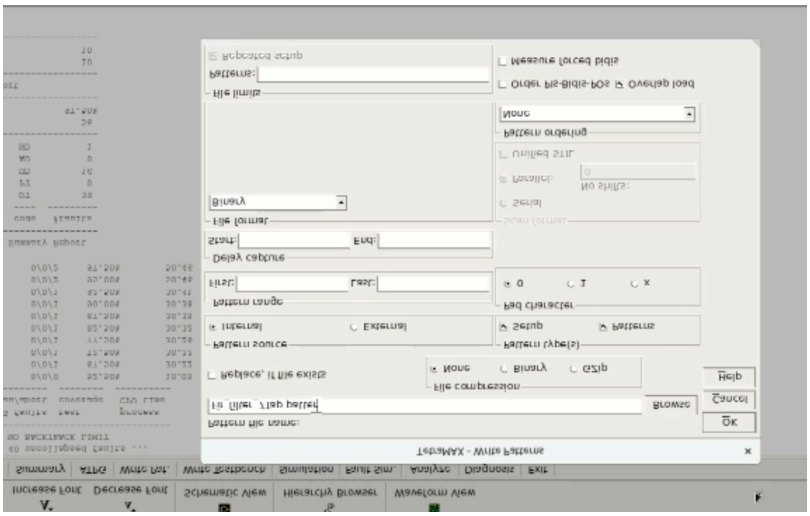


Figure 57: Writing Test Patterns to File

Once the patterns are written, they are stored in the same folder containing the library and netlist files, ensuring easy access and organization, shows the details of the generated patterns, including detected faults and fault coverage.

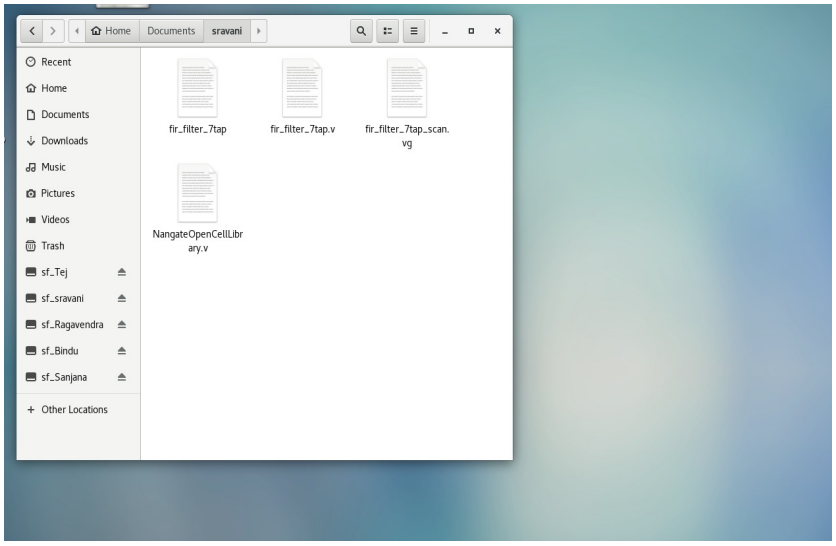


Figure 58: Generated test patterns

```
STIL 1.0 { Design 2005; }
Header {
   Title "  TetraMAX(R)  L-2016.03-SP5-i161014_180153 STIL output";
   Date "Wed Apr 30 15:54:03 2025";
   History {
      Ann {*   Uncollapsed Transition Fault Summary Report *}
      Ann {* ----------------------------------------------- *}
      Ann {* fault class                    code   #faults *}
      Ann {* ------------------------------  ----  --------- *}
      Ann {* Detected                        DT        904 *}
      Ann {* Possibly detected               PT          0 *}
      Ann {* Undetectable                    UD         18 *}
      Ann {* ATPG untestable                 AU          0 *}
      Ann {* Not detected                    ND          0 *}
      Ann {* ----------------------------------------------- *}
      Ann {* total faults                             922 *}
      Ann {* test coverage                         100.00% *}
      Ann {* ----------------------------------------------- *}
      Ann {*  *}
      Ann {*            Pattern Summary Report *}
      Ann {* ----------------------------------------------- *}
      Ann {* #internal patterns                       10 *}
      Ann {*     #full_sequential patterns            10 *}
      Ann {* ----------------------------------------------- *}
      Ann {*  *}
      Ann {* rule  severity  #fails  description *}
      Ann {* ----  --------  ------  ---------------------------------- *}
      Ann {* N21   warning        1  unsupported UDP entry *}
      Ann {* B9    warning       19  undriven module internal net *}
      Ann {* B10   warning       21  unconnected module internal net *}
      Ann {* C2    warning       56  unstable nonscan DFF when clocks off *}
      Ann {* C25   warning       56  unstable cell clock input connected from multiple sources  (nomask) *}
      Ann {*  *}
      Ann {* There are no clocks *}
      Ann {* There are no constraint ports *}
      Ann {* There are no equivalent pins *}
      Ann {* There are no net connections *}
      Ann {* top_module_name = fir_filter_7tap *}
      Ann {* Unified STIL Flow *}
      Ann {* min_n_shifts = 1 *}
      Ann {* serial_flag = 1 *}
   }
}
```

Figure 59: Detailed View of Generated Test Patterns

3.7. Schematic View

For visual verification and analysis:

1. Click on the "SCHEMATIC VIEW" in TetraMax, as shown in Figure.
2. Click on "SETUP" to select the hierarchy; choose between Design or Primitive.
3. After selecting the hierarchy, click "SHOW" and then "ALL" to display the schematic design of the 8-bit Priority encoder, as shown in Figure 60 to figure 62.

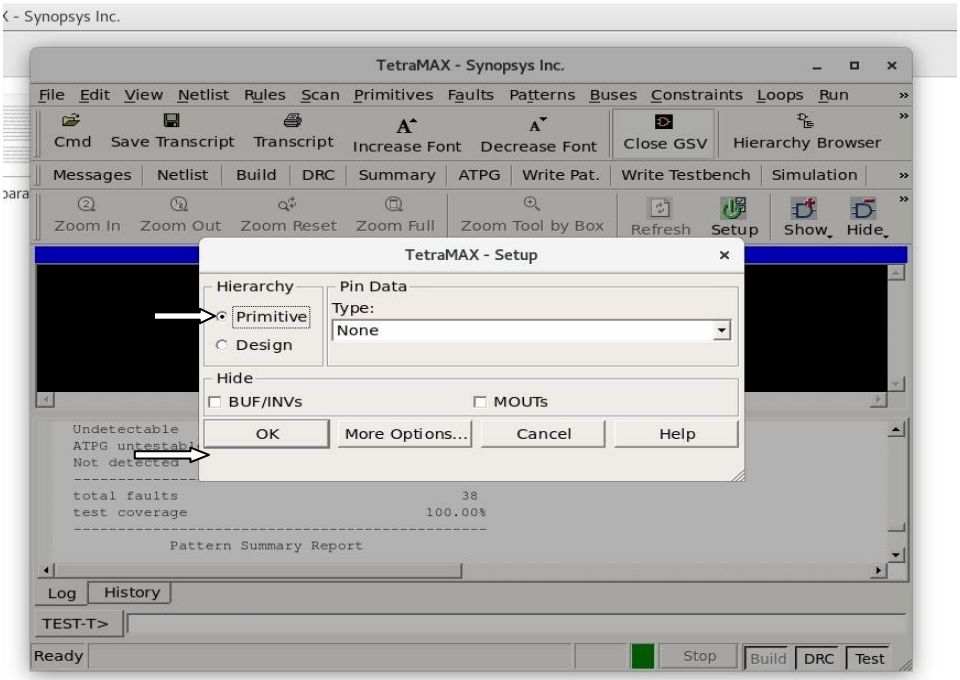


Figure 60: Setting Up Hierarchy Options

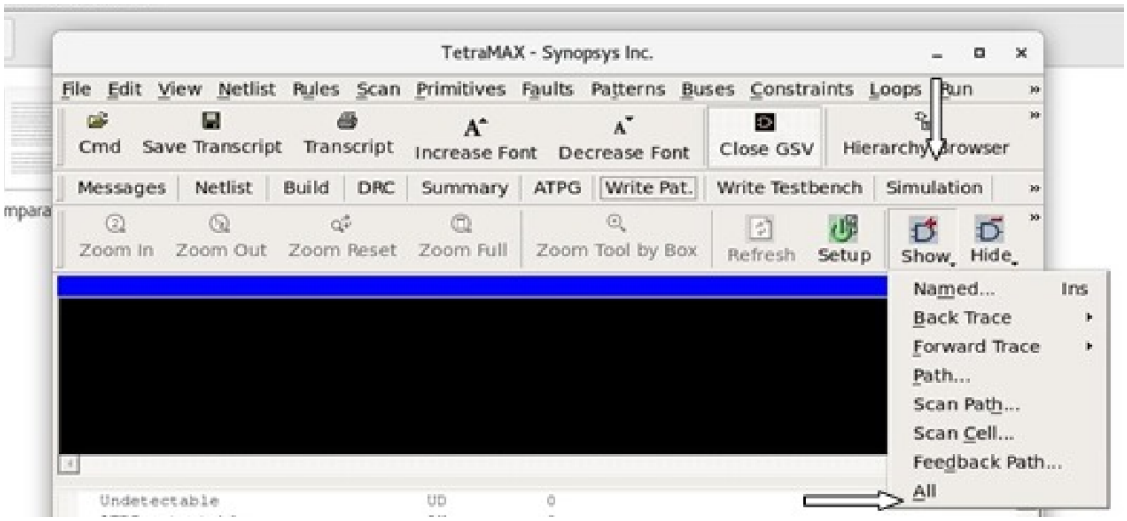


Figure 61: click "SHOW" and then "ALL"

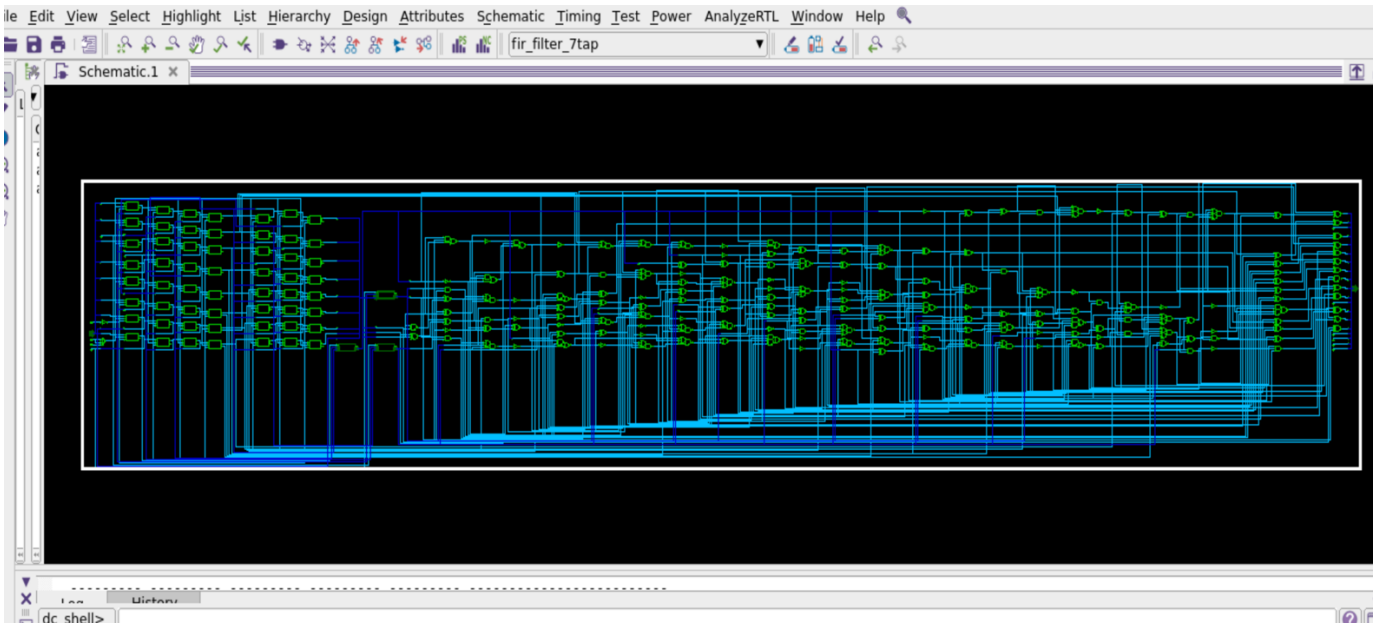


Figure 62: Final Schematic of the Design

6. CONCLUSION

This project successfully demonstrates the complete ASIC design and verification flow for a 7-tap Finite Impulse Response (FIR) filter, incorporating both power optimization and testability enhancements. The FIR filter was designed using Verilog HDL and validated through RTL simulation in ModelSim, confirming functional correctness across varied input sequences. RTL synthesis using Synopsys Design Compiler produced an efficient gate-level implementation with an area utilization of approximately 552.48 μm² and a total power consumption of 247.8 μW, including 28.62 μW of leakage power. The design achieved positive slack (2.59 ns) under a 4 ns clock period, verifying that the timing requirements were comfortably met.

To address the need for post-silicon testability, scan chains were inserted using Synopsys DFT Compiler. The scan-inserted gate-level netlist was re-verified via post-synthesis simulation, ensuring functional equivalence and proper integration of test logic. The test mode override mechanism effectively maintained scan chain operation without disrupting normal filter behavior. Automatic Test Pattern Generation (ATPG) was performed using TetraMAX, resulting in 100% fault coverage for 922 faults, with only 9 patterns required, and negligible computational delay (14.04 CPU seconds). These results confirm that the scan-inserted FIR filter is not only functionally correct and power-efficient but also highly testable and production-ready.

In conclusion, this project validates a robust and industry-aligned methodology for designing low-power DSP blocks with full scan-based test coverage. It proves that integrating clock gating and DFT techniques can be effectively achieved without compromising timing or functional correctness. The outcomes support further exploration of power-aware test strategies and the extension of this design approach to more complex signal processing systems.

REFERENCES

[1] P. Heřmánek, D. Bálek, and V. Žalud, “Reducing Power Consumption of an Embedded DSP Platform through the Clock-Gating Technique,” in *2021 44th International Conference on Telecommunications and Signal Processing (TSP)*, pp. 160–164, IEEE, 2021.
[2] M. V. Reddy and M. K. Choi, “Optimized Clock Gating Cell for Low Power Design in Nanoscale CMOS Technology,” in *2015 International SoC Design Conference (ISOCC)*, pp. 165–168, IEEE, 2015.
[3] Y.-T. Chao, C.-H. Lin, and C.-H. Lin, “Low-power Implementations of DSP through Operand Isolation and Clock Gating,” in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 3, pp. 689–697, March 2015.
[4] S. Vaidya, H. P. Patil, and A. D. Patil, “Implementation of a High-Speed Low Power DSP Co-Processor Based on Clock Gating and Vedic Mathematics,” in *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, vol. 3, no. 2, pp. 7304–7309, 2014.
[5] R. Srivatsava, R. K. Dwivedi, and M. Sarvagya, “Dynamic Power Reduction through Clock Gating Technique for Low Power Memory Applications,” in *International Journal of Engineering Research and Applications*, vol. 7, no. 7, pp. 22–27, July 2017.
[6] K.-H. Tsai and J. Rajski, “Clock-Domain-Aware Test for Improving Pattern Compression,” in *2015 IEEE International Test Conference (ITC)*, pp. 1–10, IEEE, 2015.

7. APPENDIX

Verilog Code (for Pre-Synthesis simulation):

```
`timescale 10ps/1ps
`celldefine
// 7-Tap Low-Pass FIR Filter with Test-Aware Clock Gating

module fir_filter_7tap #(parameter N = 8) (
   input wire clk,
   input wire rst,
   input wire test_mode, // For test-aware gating
   input wire [N-1:0] x_in,
   output wire [2*N+2:0] y_out
);

   // Coefficients (Symmetric) - Example Fixed-Point Q4.4 Format
   parameter signed [N-1:0] h0 = 8'd2,
                            h1 = 8'd4,
                            h2 = 8'd6,
                            h3 = 8'd8;

   // Shift Register for Input Samples
   reg [N-1:0] x [0:6];
   integer i;

   // Clock Gating Control
   wire clk_gated;
   wire enable_data = 1'b1; // This can be a dynamic signal in real apps
   assign clk_gated = test_mode ? clk : (clk & enable_data);

   // Shift Register Block with Gated Clock
   always @(posedge clk_gated or posedge rst) begin
      if (rst) begin
         for (i = 0; i < 7; i = i + 1) x[i] <= 0;
      end else begin
         for (i = 6; i > 0; i = i - 1) x[i] <= x[i-1];
         x[0] <= x_in;
      end
   end

   // Symmetric Multiply-Accumulate
   wire signed [2*N-1:0] p0 = h0 * (x[0] + x[6]);
   wire signed [2*N-1:0] p1 = h1 * (x[1] + x[5]);
   wire signed [2*N-1:0] p2 = h2 * (x[2] + x[4]);
   wire signed [2*N-1:0] p3 = h3 * x[3];
```

```
    assign y_out = p0 + p1 + p2 + p3;

endmodule


    // 7-Tap Low-Pass FIR Filter with Test-Aware Clock Gating

module fir_filter_7tap #(parameter N = 8) (
    input wire clk,
    input wire rst,
    input wire test_mode, // For test-aware gating
    input wire [N-1:0] x_in,
    output wire [2*N+2:0] y_out
);

    // Coefficients (Symmetric) - Example Fixed-Point Q4.4 Format
    parameter signed [N-1:0] h0 = 8'd2,
                             h1 = 8'd4,
                             h2 = 8'd6,
                             h3 = 8'd8;

    // Shift Register for Input Samples
    reg [N-1:0] x [0:6];
    integer i;

    // Clock Gating Control
    wire clk_gated;
    wire enable_data = 1'b1; // This can be a dynamic signal in real apps
    assign clk_gated = test_mode ? clk : (clk & enable_data);

    // Shift Register Block with Gated Clock
    always @(posedge clk_gated or posedge rst) begin
        if (rst) begin
            for (i = 0; i < 7; i = i + 1) x[i] <= 0;
        end else begin
            for (i = 6; i > 0; i = i - 1) x[i] <= x[i-1];
            x[0] <= x_in;
        end
    end

    // Symmetric Multiply-Accumulate
    wire signed [2*N-1:0] p0 = h0 * (x[0] + x[6]);
    wire signed [2*N-1:0] p1 = h1 * (x[1] + x[5]);
    wire signed [2*N-1:0] p2 = h2 * (x[2] + x[4]);
    wire signed [2*N-1:0] p3 = h3 * x[3];
```

```
    assign y_out = p0 + p1 + p2 + p3;

endmodule
```

Test Bench (for Pre-Synthesis simulation):

```
// Testbench for fir_filter_7tap

`timescale 1ns/1ps

module fir_filter_tb;

    parameter N = 8;
    reg clk;
    reg rst;
    reg test_mode;
    reg [N-1:0] x_in;
    wire [2*N+2:0] y_out;

    // Instantiate the DUT (Device Under Test)
    fir_filter_7tap #(N) uut (
        .clk(clk),
        .rst(rst),
        .test_mode(test_mode),
        .x_in(x_in),
        .y_out(y_out)
    );

    // Clock Generation
    initial clk = 0;
    always #5 clk = ~clk;  // 100 MHz clock

    // Test Vector Input
    reg [N-1:0] input_sequence [0:15];
    integer i;

    initial begin
        // Initialize input sequence (example values)
        input_sequence[0] = 8'd10;
        input_sequence[1] = 8'd20;
        input_sequence[2] = 8'd30;
        input_sequence[3] = 8'd40;
        input_sequence[4] = 8'd50;
```

```
        input_sequence[5] = 8'd40;
        input_sequence[6] = 8'd30;
        input_sequence[7] = 8'd20;
        input_sequence[8] = 8'd10;
        input_sequence[9] = 8'd0;
        input_sequence[10] = 8'd10;
        input_sequence[11] = 8'd20;
        input_sequence[12] = 8'd30;
        input_sequence[13] = 8'd40;
        input_sequence[14] = 8'd50;
        input_sequence[15] = 8'd60;

        // Initialize
        rst = 1;
        test_mode = 0;
        x_in = 0;
        #20;

        rst = 0;

        // Apply Inputs
        for (i = 0; i < 16; i = i + 1) begin
            x_in = input_sequence[i];
            #10;
        end

        // Enable test mode and repeat some inputs
        test_mode = 1;
        for (i = 0; i < 5; i = i + 1) begin
            x_in = 8'd25;
            #10;
        end

        $stop;
    end

endmodule
```

Synthesized GATE NETLIST (for Post-Synthesis simulation):

```
////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : S-2021.06-SP5-1
// Date      : Sun Apr 13 00:22:46 2025
////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : S-2021.06-SP5-1
// Date      : Sat May  3 18:29:20 2025
////////////////////////////////////////////////////////////


module fir_filter_7tap ( clk, rst, test_mode, x_in, y_out );
  input [7:0] x_in;
  output [18:0] y_out;
  input clk, rst, test_mode;
  wire   N0, clk_gated, \x[6><0] , \x[6><1] , \x[6><2] , \x[6><3] , \x[6><4] ,
        \x[6><5] , \x[6><6] , \x[6><7] , \x[5><0] , \x[5><1] , \x[5><2] ,
        \x[5><3] , \x[5><4] , \x[5><5] , \x[5><6] , \x[5><7] , \x[4><0] ,
        \x[4><1] , \x[4><2] , \x[4><3] , \x[4><4] , \x[4><5] , \x[4><6] ,
        \x[4><7] , \x[3><0] , \x[3><1] , \x[3><2] , \x[3><3] , \x[3><4] ,
        \x[3><5] , \x[3><6] , \x[3><7] , \x[2><0] , \x[2><1] , \x[2><2] ,
        \x[2><3] , \x[2><4] , \x[2><5] , \x[2><6] , \x[2><7] , \x[1><0] ,
        \x[1><1] , \x[1><2] , \x[1><3] , \x[1><4] , \x[1><5] , \x[1><6] ,
        \x[1><7] , \x[0><0] , \x[0><1] , \x[0><2] , \x[0><3] , \x[0><4] ,
        \x[0><5] , \x[0><6] , \x[0><7] , N38, N37, N36, N35, N34, N33, N32,
        N31, N30, N29, N28, N27, N24, N23, N22, N21, N20, N19, N18, N17, N16,
        N15, N14, N9, N8, N7, N6, N5, N12, N11, N10, \mult_88/n9 ,
        \mult_88/n8 , \mult_88/n7 , \mult_88/n6 , \mult_88/n5 , \mult_88/n4 ,
        \mult_88/n3 , \mult_88/n2 , \add_0_root_add_91_3/carry[2] ,
        \add_0_root_add_91_3/carry[3] , \add_0_root_add_91_3/carry[4] ,
        \add_0_root_add_91_3/carry[5] , \add_0_root_add_91_3/carry[6] ,
        \add_0_root_add_91_3/carry[7] , \add_0_root_add_91_3/carry[8] ,
        \add_0_root_add_91_3/carry[9] , \add_0_root_add_91_3/carry[10] ,
        \add_0_root_add_91_3/carry[11] , \add_0_root_add_91_3/carry[12] , n9,
        n10, n11, n1, n12, n15;
  wire   [9:1] p0;
  wire   [10:2] p1;
  wire   [11:0] p2;
  wire   [13:1] \add_1_root_add_91_3/carry ;
  wire   [12:1] \add_2_root_add_91_3/carry ;
  wire   [8:1] carry;
  wire   [8:1] n1carry;
  wire   [8:1] n2carry;
  assign N0 = test_mode;
  assign y_out[14] = 1'b0;
  assign y_out[15] = 1'b0;
  assign y_out[16] = 1'b0;
  assign y_out[17] = 1'b0;
  assign y_out[18] = 1'b0;
  assign y_out[0] = 1'b0;

  DFFR_X1 \x_reg[0><7]  ( .D(x_in[7]), .CK(clk_gated), .RN(n10), .Q(\x[6><0] )
        );
  DFFR_X1 \x_reg[0><6]  ( .D(x_in[6]), .CK(clk_gated), .RN(n10), .Q(\x[6><1] )
        );
  DFFR_X1 \x_reg[0><5]  ( .D(x_in[5]), .CK(clk_gated), .RN(n10), .Q(\x[6><2] )
```

```
      );
DFFR_X1 \x_reg[0><4]  ( .D(x_in[4]), .CK(clk_gated), .RN(n10), .Q(\x[6><3] )
      );
DFFR_X1 \x_reg[0><3]  ( .D(x_in[3]), .CK(clk_gated), .RN(n9), .Q(\x[6><4] )
      );
DFFR_X1 \x_reg[0><2]  ( .D(x_in[2]), .CK(clk_gated), .RN(n11), .Q(\x[6><5] )
      );
DFFR_X1 \x_reg[0><1]  ( .D(x_in[1]), .CK(clk_gated), .RN(n11), .Q(\x[6><6] )
      );
DFFR_X1 \x_reg[0><0]  ( .D(x_in[0]), .CK(clk_gated), .RN(n11), .Q(\x[6><7] )
      );
DFFR_X1 \x_reg[1><7]  ( .D(\x[6><0] ), .CK(clk_gated), .RN(n9), .Q(\x[5><0] ) );
DFFR_X1 \x_reg[1><6]  ( .D(\x[6><1] ), .CK(clk_gated), .RN(n11), .Q(
    \x[5><1] ) );
DFFR_X1 \x_reg[1><5]  ( .D(\x[6><2] ), .CK(clk_gated), .RN(n11), .Q(
    \x[5><2] ) );
DFFR_X1 \x_reg[1><4]  ( .D(\x[6><3] ), .CK(clk_gated), .RN(n11), .Q(
    \x[5><3] ) );
DFFR_X1 \x_reg[1><3]  ( .D(\x[6><4] ), .CK(clk_gated), .RN(n11), .Q(
    \x[5><4] ) );
DFFR_X1 \x_reg[1><2]  ( .D(\x[6><5] ), .CK(clk_gated), .RN(n11), .Q(
    \x[5><5] ) );
DFFR_X1 \x_reg[1><1]  ( .D(\x[6><6] ), .CK(clk_gated), .RN(n11), .Q(
    \x[5><6] ) );
DFFR_X1 \x_reg[1><0]  ( .D(\x[6><7] ), .CK(clk_gated), .RN(n10), .Q(
    \x[5><7] ) );
DFFR_X1 \x_reg[2><7]  ( .D(\x[5><0] ), .CK(clk_gated), .RN(n9), .Q(\x[4><0] ) );
DFFR_X1 \x_reg[2><6]  ( .D(\x[5><1] ), .CK(clk_gated), .RN(n11), .Q(
    \x[4><1] ) );
DFFR_X1 \x_reg[2><5]  ( .D(\x[5><2] ), .CK(clk_gated), .RN(n11), .Q(
    \x[4><2] ) );
DFFR_X1 \x_reg[2><4]  ( .D(\x[5><3] ), .CK(clk_gated), .RN(n11), .Q(
    \x[4><3] ) );
DFFR_X1 \x_reg[2><3]  ( .D(\x[5><4] ), .CK(clk_gated), .RN(n9), .Q(\x[4><4] ) );
DFFR_X1 \x_reg[2><2]  ( .D(\x[5><5] ), .CK(clk_gated), .RN(n9), .Q(\x[4><5] ) );
DFFR_X1 \x_reg[2><1]  ( .D(\x[5><6] ), .CK(clk_gated), .RN(n9), .Q(\x[4><6] ) );
DFFR_X1 \x_reg[2><0]  ( .D(\x[5><7] ), .CK(clk_gated), .RN(n9), .Q(\x[4><7] ) );
DFFR_X1 \x_reg[3><7]  ( .D(\x[4><0] ), .CK(clk_gated), .RN(n9), .Q(\x[3><0] ) );
DFFR_X1 \x_reg[3><6]  ( .D(\x[4><1] ), .CK(clk_gated), .RN(n9), .Q(\x[3><1] ) );
DFFR_X1 \x_reg[3><5]  ( .D(\x[4><2] ), .CK(clk_gated), .RN(n9), .Q(\x[3><2] ) );
DFFR_X1 \x_reg[3><4]  ( .D(\x[4><3] ), .CK(clk_gated), .RN(n9), .Q(\x[3><3] ) );
DFFR_X1 \x_reg[3><3]  ( .D(\x[4><4] ), .CK(clk_gated), .RN(n9), .Q(\x[3><4] ) );
DFFR_X1 \x_reg[3><2]  ( .D(\x[4><5] ), .CK(clk_gated), .RN(n9), .Q(\x[3><5] ) );
DFFR_X1 \x_reg[3><1]  ( .D(\x[4><6] ), .CK(clk_gated), .RN(n9), .Q(\x[3><6] ) );
DFFR_X1 \x_reg[3><0]  ( .D(\x[4><7] ), .CK(clk_gated), .RN(n9), .Q(\x[3><7] ) );
DFFR_X1 \x_reg[4><7]  ( .D(\x[3><0] ), .CK(clk_gated), .RN(n11), .Q(
    \x[2><0] ) );
DFFR_X1 \x_reg[4><6]  ( .D(\x[3><1] ), .CK(clk_gated), .RN(n10), .Q(
    \x[2><1] ) );
DFFR_X1 \x_reg[4><5]  ( .D(\x[3><2] ), .CK(clk_gated), .RN(n10), .Q(
    \x[2><2] ) );
DFFR_X1 \x_reg[4><4]  ( .D(\x[3><3] ), .CK(clk_gated), .RN(n11), .Q(
    \x[2><3] ) );
DFFR_X1 \x_reg[4><3]  ( .D(\x[3><4] ), .CK(clk_gated), .RN(n11), .Q(
```

```
    \x[2><4] ) );
DFFR_X1 \x_reg[4><2]  ( .D(\x[3><5] ), .CK(clk_gated), .RN(n11), .Q(
    \x[2><5] ) );
DFFR_X1 \x_reg[4><1]  ( .D(\x[3><6] ), .CK(clk_gated), .RN(n11), .Q(
    \x[2><6] ) );
DFFR_X1 \x_reg[4><0]  ( .D(\x[3><7] ), .CK(clk_gated), .RN(n11), .Q(
    \x[2><7] ) );
DFFR_X1 \x_reg[5><7]  ( .D(\x[2><0] ), .CK(clk_gated), .RN(n11), .Q(
    \x[1><0] ) );
DFFR_X1 \x_reg[5><6]  ( .D(\x[2><1] ), .CK(clk_gated), .RN(n11), .Q(
    \x[1><1] ) );
DFFR_X1 \x_reg[5><5]  ( .D(\x[2><2] ), .CK(clk_gated), .RN(n11), .Q(
    \x[1><2] ) );
DFFR_X1 \x_reg[5><4]  ( .D(\x[2><3] ), .CK(clk_gated), .RN(n9), .Q(\x[1><3] ) );
DFFR_X1 \x_reg[5><3]  ( .D(\x[2><4] ), .CK(clk_gated), .RN(n10), .Q(
    \x[1><4] ) );
DFFR_X1 \x_reg[5><2]  ( .D(\x[2><5] ), .CK(clk_gated), .RN(n10), .Q(
    \x[1><5] ) );
DFFR_X1 \x_reg[5><1]  ( .D(\x[2><6] ), .CK(clk_gated), .RN(n10), .Q(
    \x[1><6] ) );
DFFR_X1 \x_reg[5><0]  ( .D(\x[2><7] ), .CK(clk_gated), .RN(n10), .Q(
    \x[1><7] ) );
DFFR_X1 \x_reg[6><7]  ( .D(\x[1><0] ), .CK(clk_gated), .RN(n10), .Q(
    \x[0><0] ) );
DFFR_X1 \x_reg[6><6]  ( .D(\x[1><1] ), .CK(clk_gated), .RN(n10), .Q(
    \x[0><1] ) );
DFFR_X1 \x_reg[6><5]  ( .D(\x[1><2] ), .CK(clk_gated), .RN(n10), .Q(
    \x[0><2] ) );
DFFR_X1 \x_reg[6><4]  ( .D(\x[1><3] ), .CK(clk_gated), .RN(n10), .Q(
    \x[0><3] ) );
DFFR_X1 \x_reg[6><3]  ( .D(\x[1><4] ), .CK(clk_gated), .RN(n10), .Q(
    \x[0><4] ) );
DFFR_X1 \x_reg[6><2]  ( .D(\x[1><5] ), .CK(clk_gated), .RN(n10), .Q(
    \x[0><5] ) );
DFFR_X1 \x_reg[6><1]  ( .D(\x[1><6] ), .CK(clk_gated), .RN(n10), .Q(
    \x[0><6] ) );
DFFR_X1 \x_reg[6><0]  ( .D(\x[1><7] ), .CK(clk_gated), .RN(n10), .Q(
    \x[0><7] ) );
HA_X1 \mult_88/U10  ( .A(N5), .B(N27), .CO(\mult_88/n9 ), .S(N28) );
FA_X1 \mult_88/U9  ( .A(N6), .B(N5), .CI(\mult_88/n9 ), .CO(\mult_88/n8 ),
    .S(p2[3]) );
FA_X1 \mult_88/U8  ( .A(N7), .B(N6), .CI(\mult_88/n8 ), .CO(\mult_88/n7 ),
    .S(p2[4]) );
FA_X1 \mult_88/U7  ( .A(N8), .B(N7), .CI(\mult_88/n7 ), .CO(\mult_88/n6 ),
    .S(p2[5]) );
FA_X1 \mult_88/U6  ( .A(N9), .B(N8), .CI(\mult_88/n6 ), .CO(\mult_88/n5 ),
    .S(p2[6]) );
FA_X1 \mult_88/U5  ( .A(N10), .B(N9), .CI(\mult_88/n5 ), .CO(\mult_88/n4 ),
    .S(p2[7]) );
FA_X1 \mult_88/U4  ( .A(N11), .B(N10), .CI(\mult_88/n4 ), .CO(\mult_88/n3 ),
    .S(p2[8]) );
FA_X1 \mult_88/U3  ( .A(N12), .B(N11), .CI(\mult_88/n3 ), .CO(\mult_88/n2 ),
    .S(p2[9]) );
HA_X1 \mult_88/U2  ( .A(\mult_88/n2 ), .B(N12), .CO(p2[11]), .S(p2[10]) );
```

```
FA_X1 \add_0_root_add_91_3/U1_2  ( .A(N15), .B(N28), .CI(
    \add_0_root_add_91_3/carry[2] ), .CO(\add_0_root_add_91_3/carry[3] ),
    .S(y_out[2]) );
FA_X1 \add_0_root_add_91_3/U1_3  ( .A(N16), .B(N29), .CI(
    \add_0_root_add_91_3/carry[3] ), .CO(\add_0_root_add_91_3/carry[4] ),
    .S(y_out[3]) );
FA_X1 \add_0_root_add_91_3/U1_4  ( .A(N17), .B(N30), .CI(
    \add_0_root_add_91_3/carry[4] ), .CO(\add_0_root_add_91_3/carry[5] ),
    .S(y_out[4]) );
FA_X1 \add_0_root_add_91_3/U1_5  ( .A(N18), .B(N31), .CI(
    \add_0_root_add_91_3/carry[5] ), .CO(\add_0_root_add_91_3/carry[6] ),
    .S(y_out[5]) );
FA_X1 \add_0_root_add_91_3/U1_6  ( .A(N19), .B(N32), .CI(
    \add_0_root_add_91_3/carry[6] ), .CO(\add_0_root_add_91_3/carry[7] ),
    .S(y_out[6]) );
FA_X1 \add_0_root_add_91_3/U1_7  ( .A(N20), .B(N33), .CI(
    \add_0_root_add_91_3/carry[7] ), .CO(\add_0_root_add_91_3/carry[8] ),
    .S(y_out[7]) );
FA_X1 \add_0_root_add_91_3/U1_8  ( .A(N21), .B(N34), .CI(
    \add_0_root_add_91_3/carry[8] ), .CO(\add_0_root_add_91_3/carry[9] ),
    .S(y_out[8]) );
FA_X1 \add_0_root_add_91_3/U1_9  ( .A(N22), .B(N35), .CI(
    \add_0_root_add_91_3/carry[9] ), .CO(\add_0_root_add_91_3/carry[10] ),
    .S(y_out[9]) );
FA_X1 \add_0_root_add_91_3/U1_10  ( .A(N23), .B(N36), .CI(
    \add_0_root_add_91_3/carry[10] ), .CO(\add_0_root_add_91_3/carry[11] ),
    .S(y_out[10]) );
FA_X1 \add_0_root_add_91_3/U1_11  ( .A(N24), .B(N37), .CI(
    \add_0_root_add_91_3/carry[11] ), .CO(\add_0_root_add_91_3/carry[12] ),
    .S(y_out[11]) );
FA_X1 \add_1_root_add_91_3/U1_4  ( .A(p2[4]), .B(\x[3><6] ), .CI(
    \add_1_root_add_91_3/carry [4]), .CO(\add_1_root_add_91_3/carry [5]),
    .S(N30) );
FA_X1 \add_1_root_add_91_3/U1_5  ( .A(p2[5]), .B(\x[3><5] ), .CI(
    \add_1_root_add_91_3/carry [5]), .CO(\add_1_root_add_91_3/carry [6]),
    .S(N31) );
FA_X1 \add_1_root_add_91_3/U1_6  ( .A(p2[6]), .B(\x[3><4] ), .CI(
    \add_1_root_add_91_3/carry [6]), .CO(\add_1_root_add_91_3/carry [7]),
    .S(N32) );
FA_X1 \add_1_root_add_91_3/U1_7  ( .A(p2[7]), .B(\x[3><3] ), .CI(
    \add_1_root_add_91_3/carry [7]), .CO(\add_1_root_add_91_3/carry [8]),
    .S(N33) );
FA_X1 \add_1_root_add_91_3/U1_8  ( .A(p2[8]), .B(\x[3><2] ), .CI(
    \add_1_root_add_91_3/carry [8]), .CO(\add_1_root_add_91_3/carry [9]),
    .S(N34) );
FA_X1 \add_1_root_add_91_3/U1_9  ( .A(p2[9]), .B(\x[3><1] ), .CI(
    \add_1_root_add_91_3/carry [9]), .CO(\add_1_root_add_91_3/carry [10]),
    .S(N35) );
FA_X1 \add_1_root_add_91_3/U1_10  ( .A(p2[10]), .B(\x[3><0] ), .CI(
    \add_1_root_add_91_3/carry [10]), .CO(\add_1_root_add_91_3/carry [11]),
    .S(N36) );
FA_X1 \add_2_root_add_91_3/U1_3  ( .A(p0[3]), .B(p1[3]), .CI(
    \add_2_root_add_91_3/carry [3]), .CO(\add_2_root_add_91_3/carry [4]),
    .S(N16) );
```

```
FA_X1 \add_2_root_add_91_3/U1_4  ( .A(p0[4]), .B(p1[4]), .CI(
    \add_2_root_add_91_3/carry [4]), .CO(\add_2_root_add_91_3/carry [5]),
    .S(N17) );
FA_X1 \add_2_root_add_91_3/U1_5  ( .A(p0[5]), .B(p1[5]), .CI(
    \add_2_root_add_91_3/carry [5]), .CO(\add_2_root_add_91_3/carry [6]),
    .S(N18) );
FA_X1 \add_2_root_add_91_3/U1_6  ( .A(p0[6]), .B(p1[6]), .CI(
    \add_2_root_add_91_3/carry [6]), .CO(\add_2_root_add_91_3/carry [7]),
    .S(N19) );
FA_X1 \add_2_root_add_91_3/U1_7  ( .A(p0[7]), .B(p1[7]), .CI(
    \add_2_root_add_91_3/carry [7]), .CO(\add_2_root_add_91_3/carry [8]),
    .S(N20) );
FA_X1 \add_2_root_add_91_3/U1_8  ( .A(p0[8]), .B(p1[8]), .CI(
    \add_2_root_add_91_3/carry [8]), .CO(\add_2_root_add_91_3/carry [9]),
    .S(N21) );
FA_X1 \add_2_root_add_91_3/U1_9  ( .A(p0[9]), .B(p1[9]), .CI(
    \add_2_root_add_91_3/carry [9]), .CO(\add_2_root_add_91_3/carry [10]),
    .S(N22) );
BUF_X1 U12 ( .A(n11), .Z(n9) );
BUF_X1 U13 ( .A(n11), .Z(n10) );
AND2_X1 U14 ( .A1(\add_0_root_add_91_3/carry[12] ), .A2(N38), .ZN(y_out[13])
     );
XOR2_X1 U15 ( .A(N38), .B(\add_0_root_add_91_3/carry[12] ), .Z(y_out[12]) );
AND2_X1 U16 ( .A1(N27), .A2(N14), .ZN(\add_0_root_add_91_3/carry[2] ) );
XOR2_X1 U17 ( .A(N27), .B(N14), .Z(y_out[1]) );
AND2_X1 U18 ( .A1(\add_2_root_add_91_3/carry [10]), .A2(p1[10]), .ZN(N24) );
XOR2_X1 U19 ( .A(p1[10]), .B(\add_2_root_add_91_3/carry [10]), .Z(N23) );
AND2_X1 U20 ( .A1(p0[2]), .A2(p1[2]), .ZN(\add_2_root_add_91_3/carry [3]) );
XOR2_X1 U21 ( .A(p1[2]), .B(p0[2]), .Z(N15) );
AND2_X1 U22 ( .A1(\add_1_root_add_91_3/carry [11]), .A2(p2[11]), .ZN(N38) );
XOR2_X1 U23 ( .A(p2[11]), .B(\add_1_root_add_91_3/carry [11]), .Z(N37) );
AND2_X1 U24 ( .A1(p2[3]), .A2(\x[3><7] ), .ZN(\add_1_root_add_91_3/carry [4]) );
XOR2_X1 U25 ( .A(\x[3><7] ), .B(p2[3]), .Z(N29) );
MUX2_X1 U26 ( .A(clk), .B(clk), .S(N0), .Z(clk_gated) );
INV_X1 U27 ( .A(rst), .ZN(n11) );
XOR2_X1 U2 ( .A(\x[1><7] ), .B(\x[5><7] ), .Z(p1[2]) );
AND2_X1 U1 ( .A1(\x[1><7] ), .A2(\x[5><7] ), .ZN(n1) );
FA_X1 U1_1 ( .A(\x[5><6] ), .B(\x[1><6] ), .CI(n1), .CO(carry[2]), .S(p1[3])
     );
FA_X1 U1_2 ( .A(\x[5><5] ), .B(\x[1><5] ), .CI(carry[2]), .CO(carry[3]), .S(
    p1[4]) );
FA_X1 U1_3 ( .A(\x[5><4] ), .B(\x[1><4] ), .CI(carry[3]), .CO(carry[4]), .S(
    p1[5]) );
FA_X1 U1_4 ( .A(\x[5><3] ), .B(\x[1><3] ), .CI(carry[4]), .CO(carry[5]), .S(
    p1[6]) );
FA_X1 U1_5 ( .A(\x[5><2] ), .B(\x[1><2] ), .CI(carry[5]), .CO(carry[6]), .S(
    p1[7]) );
FA_X1 U1_6 ( .A(\x[5><1] ), .B(\x[1><1] ), .CI(carry[6]), .CO(carry[7]), .S(
    p1[8]) );
FA_X1 U1_7 ( .A(\x[5><0] ), .B(\x[1><0] ), .CI(carry[7]), .CO(p1[10]), .S(
    p1[9]) );
XOR2_X1 U210 ( .A(\x[0><7] ), .B(\x[6><7] ), .Z(N14) );
AND2_X1 U110 ( .A1(\x[0><7] ), .A2(\x[6><7] ), .ZN(n12) );
FA_X1 U1_11 ( .A(\x[6><6] ), .B(\x[0><6] ), .CI(n12), .CO(n1carry[2]), .S(
```

```
      p0[2]) );
  FA_X1 U1_21 ( .A(\x[6><5] ), .B(\x[0><5] ), .CI(n1carry[2]), .CO(n1carry[3]),
      .S(p0[3]) );
  FA_X1 U1_31 ( .A(\x[6><4] ), .B(\x[0><4] ), .CI(n1carry[3]), .CO(n1carry[4]),
      .S(p0[4]) );
  FA_X1 U1_41 ( .A(\x[6><3] ), .B(\x[0><3] ), .CI(n1carry[4]), .CO(n1carry[5]),
      .S(p0[5]) );
  FA_X1 U1_51 ( .A(\x[6><2] ), .B(\x[0><2] ), .CI(n1carry[5]), .CO(n1carry[6]),
      .S(p0[6]) );
  FA_X1 U1_61 ( .A(\x[6><1] ), .B(\x[0><1] ), .CI(n1carry[6]), .CO(n1carry[7]),
      .S(p0[7]) );
  FA_X1 U1_71 ( .A(\x[6><0] ), .B(\x[0><0] ), .CI(n1carry[7]), .CO(p0[9]), .S(
      p0[8]) );
  XOR2_X1 U211 ( .A(\x[2><7] ), .B(\x[4><7] ), .Z(N27) );
  AND2_X1 U111 ( .A1(\x[2><7] ), .A2(\x[4><7] ), .ZN(n15) );
  FA_X1 U1_12 ( .A(\x[4><6] ), .B(\x[2><6] ), .CI(n15), .CO(n2carry[2]), .S(N5) );
  FA_X1 U1_22 ( .A(\x[4><5] ), .B(\x[2><5] ), .CI(n2carry[2]), .CO(n2carry[3]),
      .S(N6) );
  FA_X1 U1_32 ( .A(\x[4><4] ), .B(\x[2><4] ), .CI(n2carry[3]), .CO(n2carry[4]),
      .S(N7) );
  FA_X1 U1_42 ( .A(\x[4><3] ), .B(\x[2><3] ), .CI(n2carry[4]), .CO(n2carry[5]),
      .S(N8) );
  FA_X1 U1_52 ( .A(\x[4><2] ), .B(\x[2><2] ), .CI(n2carry[5]), .CO(n2carry[6]),
      .S(N9) );
  FA_X1 U1_62 ( .A(\x[4><1] ), .B(\x[2><1] ), .CI(n2carry[6]), .CO(n2carry[7]),
      .S(N10) );
  FA_X1 U1_72 ( .A(\x[4><0] ), .B(\x[2><0] ), .CI(n2carry[7]), .CO(N12), .S(
      N11) );
endmodule
```

Test Patterns Written by TetraMAX:

```
STIL 1.0 { Design 2005; }
Header {
  Title "  TetraMAX(R)  L-2016.03-SP5-i161014_180153 STIL output";
  Date "Wed Apr 30 17:48:16 2025";
  History {
    Ann {*   Uncollapsed Transition Fault Summary Report *}
    Ann {* ----------------------------------------------- *}
    Ann {* fault class                  code   #faults *}
    Ann {* ------------------------------  ----  --------- *}
    Ann {* Detected                     DT       904 *}
    Ann {* Possibly detected             PT         0 *}
    Ann {* Undetectable                 UD        18 *}
    Ann {* ATPG untestable               AU         0 *}
    Ann {* Not detected                  ND         0 *}
    Ann {* ----------------------------------------------- *}
    Ann {* total faults                          922 *}
    Ann {* test coverage                        100.00% *}
    Ann {* ----------------------------------------------- *}
    Ann {*  *}
    Ann {*           Pattern Summary Report *}
    Ann {* ----------------------------------------------- *}
    Ann {* #internal patterns                       9 *}
```

```
    Ann {*     #full_sequential patterns               9 *}
    Ann {* ---------------------------------------------- *}
    Ann {*  *}
    Ann {* rule  severity  #fails  description *}
    Ann {* ----  --------  ------  ---------------------------------- *}
    Ann {* N21   warning       1  unsupported UDP entry *}
    Ann {* B9    warning      19  undriven module internal net *}
    Ann {* B10   warning      21  unconnected module internal net *}
    Ann {* C2    warning      56  unstable nonscan DFF when clocks off *}
    Ann {* C25   warning      56  unstable cell clock input connected from multiple sources  (nomask) *}
    Ann {*  *}
    Ann {* There are no clocks *}
    Ann {* There are no constraint ports *}
    Ann {* There are no equivalent pins *}
    Ann {* There are no net connections *}
    Ann {* top_module_name = fir_filter_7tap *}
    Ann {* Unified STIL Flow *}
    Ann {* min_n_shifts = 1 *}
    Ann {* serial_flag = 1 *}
  }
}
Signals {
  "clk" In; "rst" In; "test_mode" In; "x_in[7]" In; "x_in[6]" In; "x_in[5]" In; "x_in[4]" In;
  "x_in[3]" In; "x_in[2]" In; "x_in[1]" In; "x_in[0]" In; "y_out[18]" Out; "y_out[17]" Out;
  "y_out[16]" Out; "y_out[15]" Out; "y_out[14]" Out; "y_out[13]" Out; "y_out[12]" Out;
  "y_out[11]" Out; "y_out[10]" Out; "y_out[9]" Out; "y_out[8]" Out; "y_out[7]" Out;
  "y_out[6]" Out; "y_out[5]" Out; "y_out[4]" Out; "y_out[3]" Out; "y_out[2]" Out;
  "y_out[1]" Out; "y_out[0]" Out;
}
SignalGroups {
  "_default_In_Timing_" = '"clk" + "rst" + "test_mode" + "x_in[7]" + "x_in[6]" +
  "x_in[5]" + "x_in[4]" + "x_in[3]" + "x_in[2]" + "x_in[1]" + "x_in[0]"'; // #signals=11
  "_pi" = '"clk" + "rst" + "test_mode" + "x_in[7]" + "x_in[6]" + "x_in[5]" +
  "x_in[4]" + "x_in[3]" + "x_in[2]" + "x_in[1]" + "x_in[0]"'; // #signals=11
  "_in" = '"clk" + "rst" + "test_mode" + "x_in[7]" + "x_in[6]" + "x_in[5]" +
  "x_in[4]" + "x_in[3]" + "x_in[2]" + "x_in[1]" + "x_in[0]"'; // #signals=11
  "_default_Out_Timing_" = '"y_out[18]" + "y_out[17]" + "y_out[16]" +
  "y_out[15]" + "y_out[14]" + "y_out[13]" + "y_out[12]" + "y_out[11]" +
  "y_out[10]" + "y_out[9]" + "y_out[8]" + "y_out[7]" + "y_out[6]" + "y_out[5]" +
  "y_out[4]" + "y_out[3]" + "y_out[2]" + "y_out[1]" + "y_out[0]"'; // #signals=19
  "_po" = '"y_out[18]" + "y_out[17]" + "y_out[16]" + "y_out[15]" + "y_out[14]" +
  "y_out[13]" + "y_out[12]" + "y_out[11]" + "y_out[10]" + "y_out[9]" +
  "y_out[8]" + "y_out[7]" + "y_out[6]" + "y_out[5]" + "y_out[4]" + "y_out[3]" +
  "y_out[2]" + "y_out[1]" + "y_out[0]"'; // #signals=19
  "_out" = '"y_out[18]" + "y_out[17]" + "y_out[16]" + "y_out[15]" + "y_out[14]" +
  "y_out[13]" + "y_out[12]" + "y_out[11]" + "y_out[10]" + "y_out[9]" +
  "y_out[8]" + "y_out[7]" + "y_out[6]" + "y_out[5]" + "y_out[4]" + "y_out[3]" +
  "y_out[2]" + "y_out[1]" + "y_out[0]"'; // #signals=19
}
Timing {
  WaveformTable "_default_WFT_" {
    Period '100ns';
    Waveforms {
      "_default_In_Timing_" { 0 { '0ns' D; } }
```

```
        "_default_In_Timing_" { 1 { '0ns' U; } }
        "_default_In_Timing_" { Z { '0ns' Z; } }
        "_default_In_Timing_" { N { '0ns' N; } }
        "_default_Out_Timing_" { X { '0ns' X; } }
        "_default_Out_Timing_" { H { '0ns' X; '40ns' H; } }
        "_default_Out_Timing_" { T { '0ns' X; '40ns' T; } }
        "_default_Out_Timing_" { L { '0ns' X; '40ns' L; } }
      }
    }
}
ScanStructures {
  // Uncomment and modify the following to suit your design
  // ScanChain "chain_name" { ScanIn "chain_input_name"; ScanOut "chain_output_name"; }
}
PatternBurst "_burst_" {
  PatList { "_pattern_" {
  }
}}
PatternExec {
  PatternBurst "_burst_";
}
Procedures {
  "capture" {
    W "_default_WFT_";
    C { "_po"=\r19 X ; }
    "forcePI": V { "_pi"=\r11 # ; }
    "measurePO": V { "_po"=\r19 # ; }
  }
  // Uncomment and modify the following to suit your design
  // load_unload {
    // V { } // force clocks off and scan enable pins active
    // Shift { V { _si=#; _so=#; }} // pulse shift clocks
  // }
}
MacroDefs {
}
Pattern "_pattern_" {
  W "_default_WFT_";
  "precondition all Signals": C { "_pi"=\r11 0 ; "_po"=\r19 X ; }
  Ann {* full_sequential *}
  "pattern 0": V { "_pi"=11000000111; }
  V {}
  V { "_pi"=00100110110; }
  V { "_po"=LLLLLLLLLLLLLLLLLL; }
  C { "_po"= \r19 X ; }
  V { "_pi"=10101111000; }
  V { "_po"=LLLLLLLLLLLHHLHHLL; }
  C { "_po"= \r19 X ; }
  V { "_pi"=10100101000; }
  V {}
  V { "_pi"=10101010101; }
  V {}
  V { "_pi"=00110100011; }
  V {}
```

```
V { "_pi"=00011111011; }
V {}
V { "_pi"=00001000100; }
V {}
V { "_pi"=10101010010; }
V { "_po"=LLLLLLLLLLHLHHLLLLL; }
C { "_po"= \r19 X ; }
V { "_pi"=00100110011; }
V {}
V { "_pi"=00011010001; }
V { "_po"=LLLLLLLLLLHLHHLLLLL; }
C { "_po"= \r19 X ; }
V { "_pi"=10111011011; }
V { "_po"=LLLLLLLLLHHHHHHLHHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00000101100; }
V {}
V { "_pi"=00100000000; }
V {}
V { "_pi"=10001010010; }
V { "_po"=LLLLLLLLHHLHLLLHHLL; }
C { "_po"= \r19 X ; }
V { "_pi"=00100000000; }
V {}
V { "_pi"=10011111100; }
V { "_po"=LLLLLLLHLLLLHLLHLHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00001110111; }
V {}
V { "_pi"=00011111010; }
V { "_po"=LLLLLLLHLLLLHLLHLHL; }
C { "_po"= \r19 X ; }
V { "_pi"=11010100111; }
"end 0 measurePO": V { "_po"=LLLLLLLLLLLLLLLLLLL; }
C { "_po"= \r19 X ; }
Ann {* full_sequential *}
"pattern 1": V { "_pi"=00011101101; }
V {}
V { "_pi"=10000111100; }
V {}
V { "_pi"=00110001011; }
V {}
V { "_pi"=10101011011; }
V {}
V { "_pi"=01110110100; }
V { "_po"=LLLLLLLLLLLLLLLLLLL; }
C { "_po"= \r19 X ; }
V { "_pi"=11111101010; }
V {}
V { "_pi"=00000000010; }
V { "_po"=LLLLLLLLLLLLLLLLLLL; }
C { "_po"= \r19 X ; }
V { "_pi"=10111111001; }
V { "_po"=LLLLLLLLLLLLLLLLHLL; }
```

```
C { "_po"= \r19 X ; }
V { "_pi"=10101001000; }
V {}
V { "_pi"=00110101001; }
V {}
V { "_pi"=00010001000; }
V {}
V { "_pi"=10110000001; }
V { "_po"=LLLLLLLLLLHLLLHHLLL; }
C { "_po"= \r19 X ; }
V { "_pi"=00101101111; }
V {}
V { "_pi"=00110000000; }
V {}
V { "_pi"=10010011101; }
V { "_po"=LLLLLLLLLHHLLHLHHLL; }
C { "_po"= \r19 X ; }
V { "_pi"=10110111001; }
V {}
V { "_pi"=00011001101; }
V {}
V { "_pi"=10001111000; }
V { "_po"=LLLLLLLLHHLHHLHHLHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00111000000; }
V {}
V { "_pi"=10000100001; }
V { "_po"=LLLLLLLHHLLLLLLLLLL; }
C { "_po"= \r19 X ; }
V { "_pi"=00110000000; }
V {}
V { "_pi"=00001111000; }
V {}
V { "_pi"=10011011111; }
V { "_po"=LLLLLLLHHHHHHHHLHHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00101011001; }
V {}
V { "_pi"=00010111000; }
V {}
V { "_pi"=00000000101; }
V { "_po"=LLLLLLLHHHHHHHHLHHL; }
C { "_po"= \r19 X ; }
V { "_pi"=10010110011; }
"end 1 measurePO": V { "_po"=LLLLLLHLLLHHHHHLHHL; }
C { "_po"= \r19 X ; }
Ann {* full_sequential *}
"pattern 2": V { "_pi"=01110001100; }
V { "_po"=LLLLLLLLLLLLLLLLLLL; }
C { "_po"= \r19 X ; }
V { "_pi"=00111010010; }
V {}
V { "_pi"=10101110100; }
V { "_po"=LLLLLLLLLLHHLHLLHLL; }
```

```
C { "_po"= \r19 X ; }
V { "_pi"=00100100100; }
V {}
V { "_pi"=00110111100; }
V {}
V { "_pi"=00001011101; }
V {}
V { "_pi"=00111111101; }
V {}
V { "_pi"=10101100011; }
V { "_po"=LLLLLLLLHLHLHLLLLHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00001111111; }
V {}
V { "_pi"=00010001000; }
V {}
V { "_pi"=10001011010; }
V { "_po"=LLLLLLLHLLHHHHHLLLL; }
C { "_po"= \r19 X ; }
V { "_pi"=00011001000; }
V {}
V { "_pi"=00101001000; }
V {}
V { "_pi"=10101010011; }
V { "_po"=LLLLLLLHHHHLLHLHHHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00000000000; }
V {}
V { "_pi"=10110011000; }
V { "_po"=LLLLLLHLLLHLLHLLHLL; }
C { "_po"= \r19 X ; }
V { "_pi"=10001111000; }
V {}
V { "_pi"=00011111100; }
V { "_po"=LLLLLLHLLLHLLHLLHLL; }
C { "_po"= \r19 X ; }
V { "_pi"=10110010001; }
V { "_po"=LLLLLLHLLLHLLLHHHHL; }
C { "_po"= \r19 X ; }
V { "_pi"=01111111010; }
"end 2 measurePO": V { "_po"=LLLLLLLLLLLLLLLLLLL; }
C { "_po"= \r19 X ; }
Ann {* full_sequential *}
"pattern 3": V { "_pi"=11110000100; }
V {}
V { "_pi"=00100011101; }
V { "_po"=LLLLLLLLLLLLLLLLLLL; }
C { "_po"= \r19 X ; }
V { "_pi"=10111001010; }
V { "_po"=LLLLLLLLLLLLLHHHLHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00011001100; }
V {}
V { "_pi"=10000101010; }
```

V { "_po"=LLLLLLLLLHLLLLLHHLL; }
C { "_po"= \r19 X ; }
V { "_pi"=00011100000; }
V {}
V { "_pi"=10000011101; }
V {}
V { "_pi"=00010101110; }
V {}
V { "_pi"=10000010011; }
V {}
V { "_pi"=00010111000; }
V {}
V { "_pi"=00011100000; }
V {}
V { "_pi"=10001001011; }
V { "_po"=LLLLLLHLLLLHHLLLHHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00100110010; }
V {}
V { "_pi"=00111111111; }
V {}
V { "_pi"=10011011111; }
V { "_po"=LLLLLLHLHLHHHLLHHHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00001110001; }
V {}
V { "_pi"=00111100010; }
V { "_po"=LLLLLLHLHLHHHLLHHHL; }
C { "_po"= \r19 X ; }
V { "_pi"=10010100011; }
V { "_po"=LLLLLLHHLLHLLLHHLHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00000000000; }
V {}
V { "_pi"=10001110100; }
V { "_po"=LLLLLLHHLLHHLHLHHHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00000011101; }
V {}
V { "_pi"=10110000111; }
V { "_po"=LLLLLLHLHHHLLHHLHHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00001010101; }
V {}
V { "_pi"=10101010110; }
"end 3 measurePO": V { "_po"=LLLLLLHLLHHLLLLLHLL; }
C { "_po"= \r19 X ; }
Ann {* full_sequential *}
"pattern 4": V { "_pi"=01000111110; }
V { "_po"=LLLLLLLLLLLLLLLLLLL; }
C { "_po"= \r19 X ; }
V { "_pi"=00101111100; }
V {}
V { "_pi"=10110011111; }

```
V {}
V { "_pi"=00111111100; }
V {}
V { "_pi"=10110000000; }
V { "_po"=LLLLLLLLLHHHHHLHLLL; }
C { "_po"= \r19 X ; }
V { "_pi"=00010101110; }
V {}
V { "_pi"=00000001001; }
V {}
V { "_pi"=00100010101; }
V {}
V { "_pi"=00000011100; }
V {}
V { "_pi"=00111111110; }
V {}
V { "_pi"=10001010111; }
V {}
V { "_pi"=00101101010; }
V {}
V { "_pi"=00111100111; }
V {}
V { "_pi"=00011011001; }
V {}
V { "_pi"=00010101100; }
V {}
V { "_pi"=10000100011; }
V {}
V { "_pi"=00111111101; }
V {}
V { "_pi"=00110000000; }
V {}
V { "_pi"=10100010111; }
V {}
V { "_pi"=00011100111; }
V {}
V { "_pi"=00000000001; }
V {}
V { "_pi"=10111100000; }
V {}
V { "_pi"=00010000011; }
V {}
V { "_pi"=10100100011; }
V {}
V { "_pi"=00100000011; }
V {}
V { "_pi"=10100011110; }
V { "_po"=LLLLLLHLLLLLLLHLLLL; }
C { "_po"= \r19 X ; }
V { "_pi"=01010000110; }
"end 4 measurePO": V { "_po"=LLLLLLLLLLLLLLLLLLL; }
C { "_po"= \r19 X ; }
Ann {* full_sequential *}
"pattern 5": V { "_pi"=01101001110; }
```

```
V {}
V { "_pi"=00110001000; }
V {}
V { "_pi"=10101001110; }
V {}
V { "_pi"=00110010000; }
V {}
V { "_pi"=00110100100; }
V {}
V { "_pi"=00100000010; }
V {}
V { "_pi"=10011010000; }
V {}
V { "_pi"=00011010110; }
V {}
V { "_pi"=00011000000; }
V {}
V { "_pi"=10001100101; }
V {}
V { "_pi"=00010111001; }
V {}
V { "_pi"=00110000000; }
V {}
V { "_pi"=10000110001; }
V {}
V { "_pi"=00011111000; }
V {}
V { "_pi"=10001111011; }
V {}
V { "_pi"=00111111100; }
V {}
V { "_pi"=10010010111; }
"end 5 measurePO": V { "_po"=LLLLLLHLLLHLLLLLHLL; }
C { "_po"= \r19 X ; }
Ann {* full_sequential *}
"pattern 6": V { "_pi"=00010000001; }
V {}
V { "_pi"=10111111100; }
V {}
V { "_pi"=00110000000; }
V {}
V { "_pi"=10101111100; }
V {}
V { "_pi"=00000000010; }
V {}
V { "_pi"=10101010010; }
V {}
V { "_pi"=00010000010; }
V {}
V { "_pi"=00000000000; }
V {}
V { "_pi"=10011110000; }
V {}
V { "_pi"=00001101111; }
```

```
V {}
V { "_pi"=00000000000; }
V {}
V { "_pi"=10000011011; }
V {}
V { "_pi"=00011000000; }
V {}
V { "_pi"=10000110111; }
V {}
V { "_pi"=00111111111; }
V {}
V { "_pi"=10010011001; }
V { "_po"=LLLLLLLHLLLLLLLHHLL; }
C { "_po"= \r19 X ; }
V { "_pi"=01101000111; }
"end 6 measurePO": V { "_po"=LLLLLLLLLLLLLLLLLLL; }
C { "_po"= \r19 X ; }
Ann {* full_sequential *}
"pattern 7": V { "_pi"=01101111000; }
V {}
V { "_pi"=00100000001; }
V { "_po"=LLLLLLLLLLLLLLLLLLL; }
C { "_po"= \r19 X ; }
V { "_pi"=10001011000; }
V {}
V { "_pi"=00100000001; }
V {}
V { "_pi"=10111111110; }
V {}
V { "_pi"=00110001000; }
V {}
V { "_pi"=10010100111; }
V { "_po"=LLLLLLLLLLHLLLHHLHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00110000000; }
V {}
V { "_pi"=10110011000; }
V {}
V { "_pi"=00011001101; }
V {}
V { "_pi"=10111111110; }
V {}
V { "_pi"=00001111110; }
V {}
V { "_pi"=10101101100; }
V { "_po"=LLLLLLLHLHHLHHHHLHL; }
C { "_po"= \r19 X ; }
V { "_pi"=00111111110; }
V {}
V { "_pi"=10110010000; }
"end 7 measurePO": V { "_po"=LLLLLLLHHHHHHHHHLLL; }
C { "_po"= \r19 X ; }
Ann {* full_sequential *}
"pattern 8": V { "_pi"=00111110000; }
```

```
  V {}
  V { "_pi"=10000101000; }
  V {}
  V { "_pi"=00110101001; }
  V {}
  V { "_pi"=00110000100; }
  V {}
  V { "_pi"=10111000111; }
  V {}
  V { "_pi"=00100000001; }
  V {}
  V { "_pi"=10000000001; }
  V {}
  V { "_pi"=00011000010; }
  V {}
  V { "_pi"=10100100110; }
  V {}
  V { "_pi"=00101010011; }
  V {}
  V { "_pi"=10000111100; }
  V {}
  V { "_pi"=00110000100; }
  V {}
  V { "_pi"=10000001001; }
  V {}
  V { "_pi"=00111110110; }
  V {}
  V { "_pi"=10000111100; }
  V { "_po"=LLLLLLLHHHHHHHHLHLL; }
  C { "_po"= \r19 X ; }
  V { "_pi"=01000100100; }
  "end 8 measurePO": V { "_po"=LLLLLLLLLLLLLLLLLL; }
}

// Patterns reference 366 V statements, generating 366 test cycle
```