ABSTRACT

SCAN-BASED DESIGN-FOR-TESTABILITY AND ATPG IMPLEMENTATION FOR A UART TRANSMITTER

This project presents the complete design and testability enhancement of a UART (Universal Asynchronous Receiver Transmitter) transmitter using Verilog HDL and Synopsys EDA tools. The functional verification of the RTL design was conducted through simulation in ModelSim. To enable scan-based Design-for-Testability (DFT), all sequential elements were replaced with scan-compatible flip-flops, and a complete scan chain was integrated at the RTL level.

Both the original and scan-inserted designs were synthesized using Synopsys Design Compiler. Post-synthesis simulation confirmed functional equivalence. For Automatic Test Pattern Generation (ATPG), the synthesized scan-inserted netlist and the associated constraint files were processed using Synopsys TetraMAX in GUI mode. The presence and connectivity of scan chains were verified, and fault coverage was evaluated using the ATPG tool's built-in analysis features. Schematic views were obtained directly from TetraMAX for structural understanding.

Comparative analysis of synthesis reports demonstrated that the scan-enabled design significantly improved testability with minimal overhead in area, timing, and power. This project successfully demonstrates a full RTL-to-DFT implementation, offering hands-on experience with structured testing, scan-based verification, and ATPG methodologies in digital systems.

Keywords: UART, Design-for-Testability (DFT), Scan Chain, Verilog HDL, Scan Flip-Flop, ATPG, Synopsys Design Compiler, TetraMAX, Fault Coverage, RTL Simulation, Gate-Level Synthesis

BinduSree Chandu

May 2025

TABLE OF CONTENTS **Page No.**

LIST OF FIGURES---------- ii

1. INTRODUCTION---------- 1

2. OBJECTIVE---------- 3

3. BACKGROUND---------- 4

4. ARCHITECTURE OF UART TRANSMITTER---------- 7

4.1. DESIGN OF THE UART TRANSMITTER WITHOUT SCAN----------7

4.2. DESIGN OF THE UART TRANSMITTER WITH SCAN----------7

5. SOFTWARE REQUIRED---------- 9

6. METHODOLOGY----------9

7. EXPERIMENTAL PROCEDURE---------- 10

7.1. PHASE 1: BEHAVIORAL AND RTL DESIGN----------10

7.2. PHASE 2: FUNCTIONAL SIMULATION---------- 11

7.3. PHASE 3: LOGIC SYNTHESIS USING SYNOPSYS DC COMPILER------- 12

7.3.1 UART TRANSMITTER DESIGN WITHOUT SCAN CHAIN----------13

7.3.2 UART TRANSMITTER DESIGN WITH SCAN CHAIN---------- 20

7.4. PHASE 4: GATE-LEVEL SIMULATION----------23

7.5. PHASE 5: SCAN INSERTION AND DFT VALIDATION---------- 24

7.6. PHASE 6: TEST COVERAGE ANALYSIS----------24

8. RESULTS AND COMPARISONS OF THE DATA OBTAINED---------- 29

9. CHALLENGES AND SOLUTIONS---------- 29

10. CONCLUSION ----------

11. REFERENCES.....37

LIST OF FIGURES **PG. NO**

Figure 1 Scan Flip-Flop..... 6

Figure 2 A Typical Scan Chain..... 8

Figure 3 Traditional UART Transmitter design..... 11

Figure 4 UART Transmitter Without Scan..... 7

Figure 5 UART Transmitter With Scan Enable..... 8

Figure 6 Project Flow Block Diagram..... 10

Figure 7 Original UART without Scan Compilation and Instantiation..... 12

Figure 8 Original with no scan UART Transmission o/p & Signal Monitoring during Simulation..... 12

Figure 9 Scan UART Transmission Output & Signal Monitoring during Simulation..... 13

Figure 10 Apporto login page..... 14

Figure 11 Launch Cadence..... 14

Figure 12 Home Folder..... 14

Figure 13 Opening Terminal..... 15

Figure 14 New Folder created (242project_originalUART)..... 15

Figure 15 Synopsys DC Compiler Files..... 15

Figure 16 updating .v file and saving with top module name..... 16

Figure 17 updating .vfs file..... 16

Figure 18 updating .vfv file..... 17

Figure 19 synthesis..... 17

Figure 20 After successful synthesis..... 17

Figure 21 Area, power, Time, logs reports and netlist file generated..... 18

Figure 22 Area report for original UART design without scan..... 18

Figure 23 Power report for original UART design without scan..... 18

Figure 24 Timing report for original UART design without scan..... 19

Figure 25 Netlist file also generated for original UART design without scan.....19

Figure 26 Generated Netlist file for UART design without scan..... 19

Figure 27 New Folder created (242project_UARTwithscan) and directory changedL..... 20

Figure 28 Generated report files after synthesis..... 20

Figure 29 Area report for UART design with scan..... 21

Figure 30 Power report for UART design with scan.....21

Figure 31 Timings report for UART design with scan.....22

Figure 32 Netlist file generated for UART design with scan..... 22

Figure 33 Netlist file for UART design with scan.....22

Figure 34 Gate level simulation in modelsim.....23

Figure 35 Mount the folder and copying into Centos VM..... 24

Figure 36 Opening tetramax by command in terminal.....24

Figure 37 Adding netlist file and library files..... 25

Figure 38 After Running Netlist.....25

Figure 39 Build model, loads top module name.....25

Figure 40 After running Build model..... 25

Figure 41 Open DRC.....26

Figure 42 After running DRC check..... 26

Figure 43 Report Summary, keep default.....26

Figure 44 Run ATPG by this settings..... 26

Figure 45 After running ATPG,takes time..... 27

Figure 46 Test coverage for design..... 27

Figure 47 Writing patterns..... 27

Figure 48 Pattern file created for design..... 27

Figure 49 Setup for schematic design..... 28

Figure 50 Selecting ALL to view entire design.....28

Figure 51 Schematic design zoom in.....28

Figure 52 Entire Schematic View.....28

Figure 53 For saving the schematics..... 29

Figure 54 UART Transmitter with no scan design output waveform..... 29

Figure 55 UART Transmitter with scan design output waveform..... 30

Figure 56 Synthesis Comparison – UART Without Scan vs UART With Scan..... 31

Figure 57 Gate-Level Simulation Output of UART Transmitter Using ModelSim..... 31

Figure 58 Test coverage in tetraMax.....32

Figure 59 Write patterns.....32

Figure 60 Schematic view of design.....32

1. INTRODUCTION

As integrated circuits grow increasingly complex, ensuring testability becomes essential for maintaining design quality and detecting manufacturing defects. In modern VLSI design, functional verification alone cannot guarantee the correctness of a design once fabricated. Therefore, structured testing methodologies such as Design-for-Testability (DFT) have become an integral part of the digital design process. Among DFT techniques, scan-based testing is one of the most widely adopted methods used in industry to enhance the controllability and observability of internal circuit nodes [2].

As integrated circuits become more complex, ensuring testability is vital for maintaining design integrity and identifying manufacturing defects. In modern VLSI systems, functional simulation alone is insufficient to guarantee post-fabrication correctness. To address this, Design-for-Testability (DFT) has become a critical component of the digital design flow.

Among various DFT techniques, scan-based testing is widely adopted in industry to improve the controllability and observability of internal nodes. It involves inserting extra logic that links sequential elements—like flip-flops—into a serial scan chain. This allows test patterns to be efficiently shifted into and out of the design, simplifying fault detection and enhancing ATPG (Automatic Test Pattern Generation) coverage [2].

This project implements scan-based DFT in a UART (Universal Asynchronous Receiver Transmitter) transmitter—an essential module used in microcontrollers and SoCs for serial data communication using start and stop bit framing. Integrating DFT into the UART design demonstrates how real-world communication circuits can be made more testable, production-ready.

Scan flip-flops use a Scan Enable signal to toggle between functional and test modes. In test mode, scan inputs shift test data serially through the chain, boosting observability and enabling fault coverage analysis via tools like Synopsys TetraMAX.

This project utilizes industry-standard EDA tools for simulation, synthesis, and design evaluation. ModelSim is used for functional and post-synthesis simulation. Synopsys Design Compiler handles RTL synthesis and generates area, timing, and power reports. TetraMAX is employed for ATPG and scan chain verification, tetramax used for schematic inspection and structural analysis.

The subsequent sections detail the complete project workflow, including scan integration, synthesis methodology, simulation validation and test preparation using these tools.

2. OBJECTIVE

The primary objective of this project is to design, implement, and evaluate the testability of a UART Transmitter module using scan-based DFT techniques. The project focuses on manually inserting scan flip-flops and integrating scan chains to enable effective fault detection and structured testing [1]. The design is validated through RTL simulation, synthesized using Synopsys Design Compiler and evaluated for fault readiness using TetraMAX.

The key goals of this project include:

- Design and simulate the UART Transmitter at the RTL level using Verilog HDL and ModelSim.
- Manually insert scan flip-flops to create a scan chain architecture for improved testability.
- Synthesize both scan and non-scan versions using Synopsys Design Compiler and analyze area, timing and power metrics.
- Verify post-synthesis functionality and scan connectivity through simulation and netlist inspection.
- Prepare the design for ATPG using Synopsys TetraMAX and validate the scan-based DFT structure.
- View and verify the gate-level schematic using Synopsys Tetramax for structural validation.
- Compare the original and scan-inserted designs to evaluate the impact of DFT integration on performance and testability.

3. BACKGROUND

In modern VLSI systems, communication between digital components frequently relies on serial protocols due to their simplicity, low pin count and robustness. One of the most widely used serial communication protocols is the Universal Asynchronous Receiver Transmitter (UART). UART facilitates asynchronous communication between devices by encoding data into a serial stream with defined start and stop bits, eliminating the need for a shared clock. This makes it suitable for interfacing with microcontrollers, SoCs and peripheral devices [2].

A UART Transmitter converts 8-bit parallel data into a serial stream by framing it with a start (0) and stop (1) bit. Its internal architecture—comprising control logic, shift registers, and an FSM—includes both combinational and sequential logic, making it an ideal candidate for testability enhancement at the RTL level.

While RTL simulation verifies functionality, it cannot detect post-manufacturing defects. To address this, Design-for-Testability (DFT) techniques are applied to improve fault detection. Among these,

scan-based testing is widely used. It links flip-flops into scan chains, allowing internal states to be controlled and observed externally. These chains support ATPG and ATE tools in identifying faults such as stuck-at or transition defects.

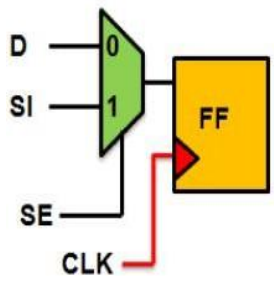


Figure 1 Scan Flip-Flop [1]

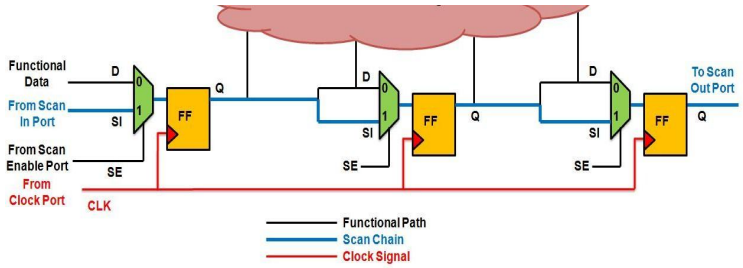


Figure 2 A Typical Scan Chain [1]

At the core of scan-based testing lies the Scan Flip-Flop (SFF), a modified D flip-flop that includes a 2:1 multiplexer controlled by a Scan Enable (SE) signal. In functional mode (SE = 0), it operates normally; in test mode (SE = 1), it shifts in scan input (SI) data and propagates it to the scan output (SO), forming a scan chain. This design preserves normal functionality while enabling greater test controllability and observability.

Using scan flip-flops allows seamless integration with ATPG tools like *Synopsys TetraMAX*, which automatically generate efficient test vectors. This improves fault coverage and reduces manual testing effort—essential for complex VLSI designs.

In this project, the UART Transmitter is an ideal target for scan-based DFT due to its moderate complexity and relevance in SoC designs. Scan logic is inserted at sequential elements, making the design compatible with ATPG tools. This approach mirrors practical testing strategies used in industry.

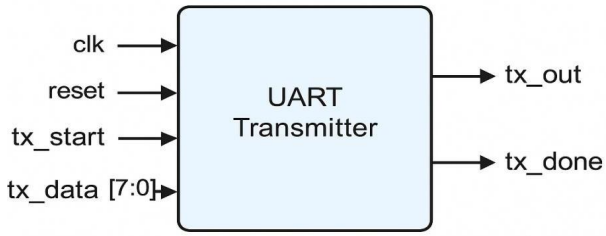


Figure 3 Traditional UART Transmitter design

Understanding how scan flip-flops interact with functional components like FSMs and shift registers is crucial for verifying correctness. The design flow integrates ModelSim for simulation, Design Compiler for synthesis, TetraMAX for scan pattern generation and schematic inspection—completing a test-ready workflow. This foundation supports the detailed exploration of scan-based DFT in the sections that follow.

4. ARCHITECTURE OF UART TRANSMITTER

In this study, outlines the UART Transmitter architecture before and after integrating scan-based Design-for-Testability (DFT), highlighting structural modifications made to enhance testability and comparing their design flow impact.

4.1. DESIGN OF THE UART TRANSMITTER WITHOUT SCAN

The baseline UART transmitter is implemented as a Finite State Machine (FSM)-controlled serial communication module. Its core components include:

- **Shift Register:** Serializes the 8-bit input data (tx_data) and controls the output signal tx_out bit by bit.
- **FSM State Register:** Holds the current state of the transmission process (IDLE, START, SHIFT, STOP, DONE) and triggers state-based behaviors.
- **Timing and Bit Counter:** Maintains the number of bits shifted during data transmission.
- **Control Logic:** Determines transmission flow through FSM logic, monitoring signals such as tx_start and generating state transition.

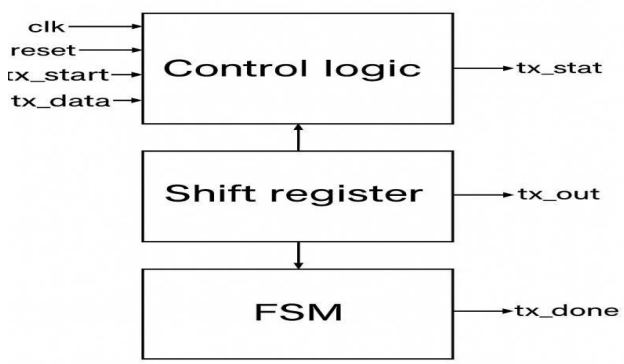


Figure 4 UART Transmitter Without Scan

4.2. DESIGN OF THE UART TRANSMITTER WITH SCAN FLIP FLOPS

In the enhanced version of the UART Transmitter, the architecture is modified to support scan-based Design-for-Testability (DFT) by replacing all sequential elements with scan flip-flops. This modification enables complete observability and controllability of internal states, allowing for effective fault detection using ATPG tools like Synopsys TetraMAX.

The scan-inserted architecture maintains the original UART functionality while introducing a serial scan chain that connects all key registers. These include the 3-bit FSM state register, 3-bit bit counter and the 8-bit shift register, totaling 14 scan flip-flops. Each flip-flop is designed with an internal multiplexer that selects between functional data and scan input based on the scan_enable signal.

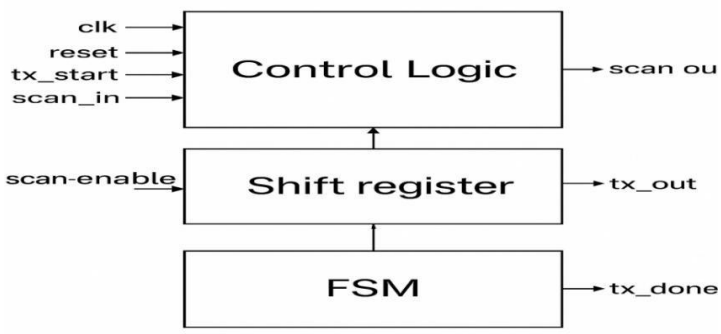


Figure 5 UART Transmitter With Scan Enable

Key Architectural Components:

- **Control Logic:** Determines transmission flow through FSM logic, monitoring signals such as tx_start and generating state transition.
- **Scan Flip-Flops (SFFs):** Used in place of standard D flip-flops. When scan_enable = 1, they allow serial shifting of test data through the chain. When scan_enable = 0, they operate as normal storage elements.
- **Scan Chain:** A single scan chain links all 14 scan flip-flops from scan_in to scan_out. This enables serial loading of test vectors and observation of output responses without altering functional behavior.
- **Scan Control Logic:** Includes scan_enable, scan_in, and scan_out ports for external test access and test mode activation. No modifications are made to the existing FSM or data path logic.
- **Preservation of Functional Operation:** The FSM transitions, bit shifting, and UART output behavior (tx_out, tx_done) remain unaffected during normal operation.

This scan-ready design enables ATPG-based test generation and analysis without compromising UART functionality. It allows post-silicon validation and fault diagnosis by making internal registers accessible through controlled scan mechanism — a crucial feature in modern VLSI testing.

5. SOFTWARE REQUIRED

This project was executed using Synopsys EDA tools within the Apporto Virtual Lab, covering the full VLSI design cycle from RTL design and simulation to synthesis, scan insertion, and ATPG analysis. Tetramax tasks were performed individually on a university-provided CentOS-based environment.

The following tools were used at various stages of the design and verification process:

- **ModelSim***:* Used during early design phases for initial RTL simulation of the UART with and without scan flip-flops [6].
- **Synopsys Design Compiler:** Employed for synthesizing the original and scan-inserted UART designs. It generated technology-mapped gate-level netlists and reports such as area, power and timing to analyze design performance [4].
- **Synopsys TetraMAX:** Used to perform ATPG, scan chain analysis and fault coverage evaluation. This validated the effectiveness of the scan design in terms of testability [5].
- **Apporto Virtual Lab:** Provided the complete cloud-based Linux environment to access and run all Synopsys tools. All project files, Verilog code, synthesis scripts, and simulation outputs were managed in the local working directory (~/projects/UART_DFT_Project).

6. METHODOLOGY

This project implements a complete VLSI design cycle for a UART Transmitter with integrated scan flip-flops, using Synopsys EDA tools via the Apporto Virtual Lab. The flow includes RTL design, simulation, synthesis, scan integration and ATPG-based fault analysis.

The UART was modeled in Verilog HDL with an FSM controller, an 8-bit shift register, and a 3-bit counter. Modelsim was used for RTL simulation and verification, functional checks and waveform analysis for both original and scan-enabled designs.

Post-verification, the design was synthesized using Synopsys Design Compiler, generating a gate-level netlist along with area, power and timing reports for performance evaluation.

To enhance testability, scan flip-flops were manually inserted, replacing all sequential elements. A scan chain was formed using scan_in, scan_out, and scan_enable signals. The modified design was then re-synthesized to assess changes in area, timing, and power.

For test generation and coverage analysis, the scan-inserted netlist was processed in TetraMAX. ATPG was conducted using the stuck-at fault model, followed by fault simulation. Coverage reports were generated to evaluate the impact and effectiveness of scan-based DFT.

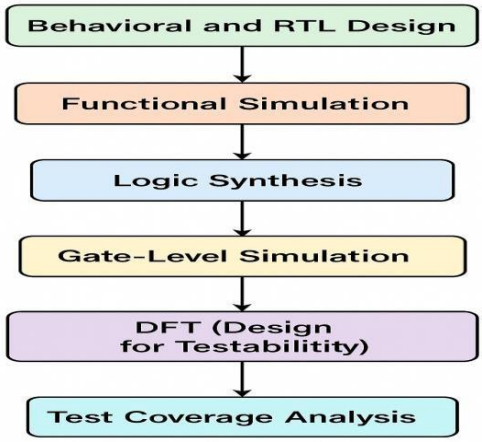


Figure 6 Project Flow Block Diagram

A comparative analysis was conducted between the original and scan-inserted UART designs, focusing on key metrics such as area, power and timing to assess the impact of scan-based DFT. All stages were executed within the Apporto Virtual Lab using Synopsys tools under a CentOS Linux VM, with an organized project directory for managing design files and outputs. This structured methodology validated the UART transmitter's testability and demonstrated practical VLSI testing using industry-standard EDA tools.

7. EXPERIMENTAL PROCEDURE

7.1 PHASE 1: BEHAVIORAL AND RTL DESIGN

In this phase, the UART (Universal Asynchronous Receiver Transmitter) Transmitter module was designed using Verilog HDL at the behavioral and Register Transfer Level (RTL). The objective was to model a digital circuit capable of transmitting 8-bit parallel data serially with appropriate start and stop bit framing, as per the UART protocol.

The architecture of the design is centered around a finite state machine (FSM) with five operational states: IDLE, START, SHIFT, STOP and DONE. This FSM governs the entire transmission sequence. Once the tx_start signal is received, the FSM transitions from IDLE to START, loads the 8-bit data into a shift register, and then serially shifts out each bit on the tx_out line. A 3-bit counter tracks how many bits have been shifted. Once all 8 bits are sent, the FSM appends the stop bit and signals the end of transmission by asserting the tx_done signal.

All sequential operations were modeled using always @(posedge clk) blocks to ensure proper clock synchronization. The FSM logic was written using a behavioral always @(*) block with a case statement to determine the next state based on current conditions. This modeling approach follows RTL conventions and is synthesizable for physical implementation.

"FSM Next-State Logic (Verilog Snippet)" defined as follows:

```
always @(*) begin
  case (state)
    IDLE:   next_state = tx_start_latched ? START : IDLE;
    START:  next_state = SHIFT;
    SHIFT:  next_state = (bit_count == 3'd7) ? STOP : SHIFT;
    STOP:   next_state = DONE;
    DONE:   next_state = IDLE;
  default: next_state = IDLE;
  endcase
end
```

This logic ensures that the transmission proceeds through each required step, maintaining UART protocol integrity while being fully compatible with RTL synthesis tools.

A Verilog testbench was written to verify the design functionality. The testbench (uart_tx_scan_tb) applied stimulus signals including clock, reset, tx_start, and a sample tx_data input. The UART transmitter was compiled and simulated using modelsim. Simulation outputs confirmed the correct transmission behavior including framing and serial bit output and validated that the tx_done signal was asserted correctly at the end of each data word transmission. I have included the codes of design in the appendix section.

This behavioral and RTL design forms the functional baseline for subsequent project phases, including synthesis, scan flip-flop insertion and ATPG test generation [2].

7.2 PHASE 2: FUNCTIONAL SIMULATION

In this phase, I verified the functional behavior of the UART Transmitter module developed in RTL using Verilog HDL. The objective was to ensure that the design was working as expected before proceeding to logic synthesis and scan insertion.

Simulation using ModelSim for design correctness:

I performed simulation using ModelSim as a validation step for validation. I compiled and ran the UART code and testbench for both the original UART without scan and UART scan inserted designs, with the following commands:

For UART without scan: used this command

```
vlog uart_tx.v
vlog uart_tx_tb.v
vsim work.uart_tx_tb
add wave *
run 300ns
```

For UART scan: used this command

```
vlog scan_dff.v
vlog uart_tx_scan.v
vlog uart_tx_scan_tb.v
vsim work.uart_tx_scan_tb
add wave *
run 300ns
```

ModelSim Simulation Screenshots:

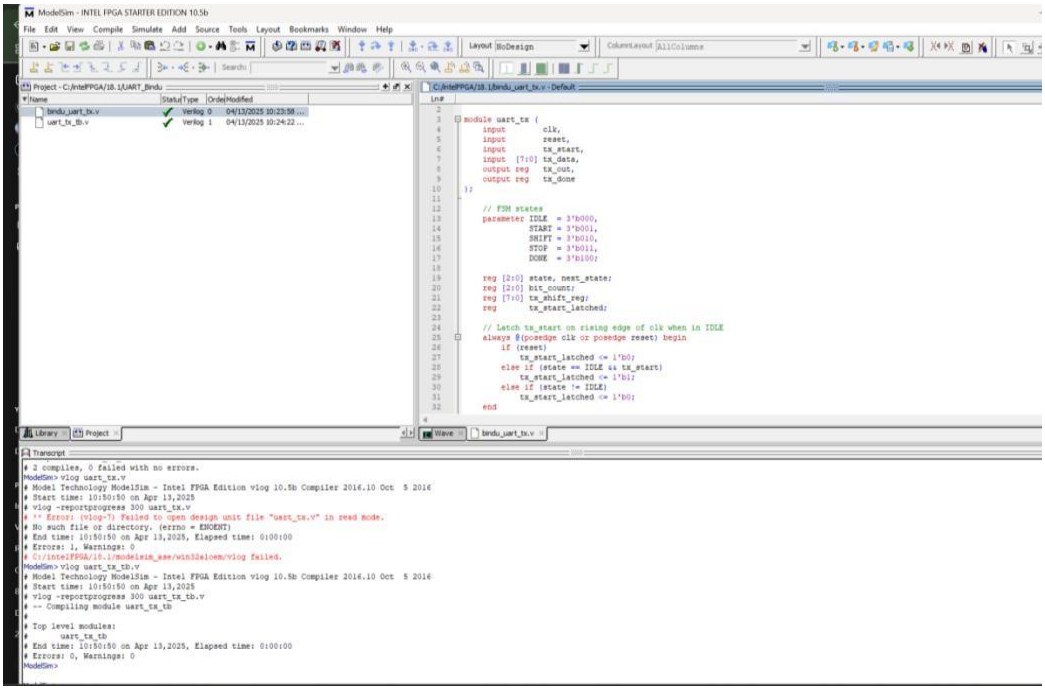


Figure 7 Original UART without Scan Compilation and Instantiation

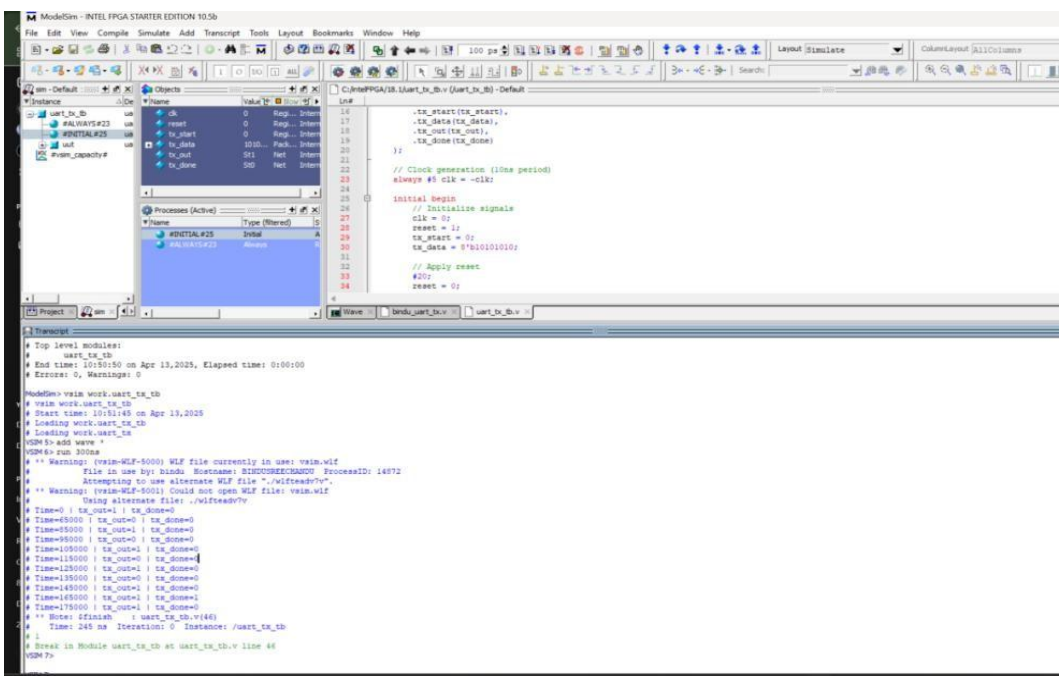


Figure 8 Original with no scan UART Transmission o/p & Signal Monitoring during Simulation

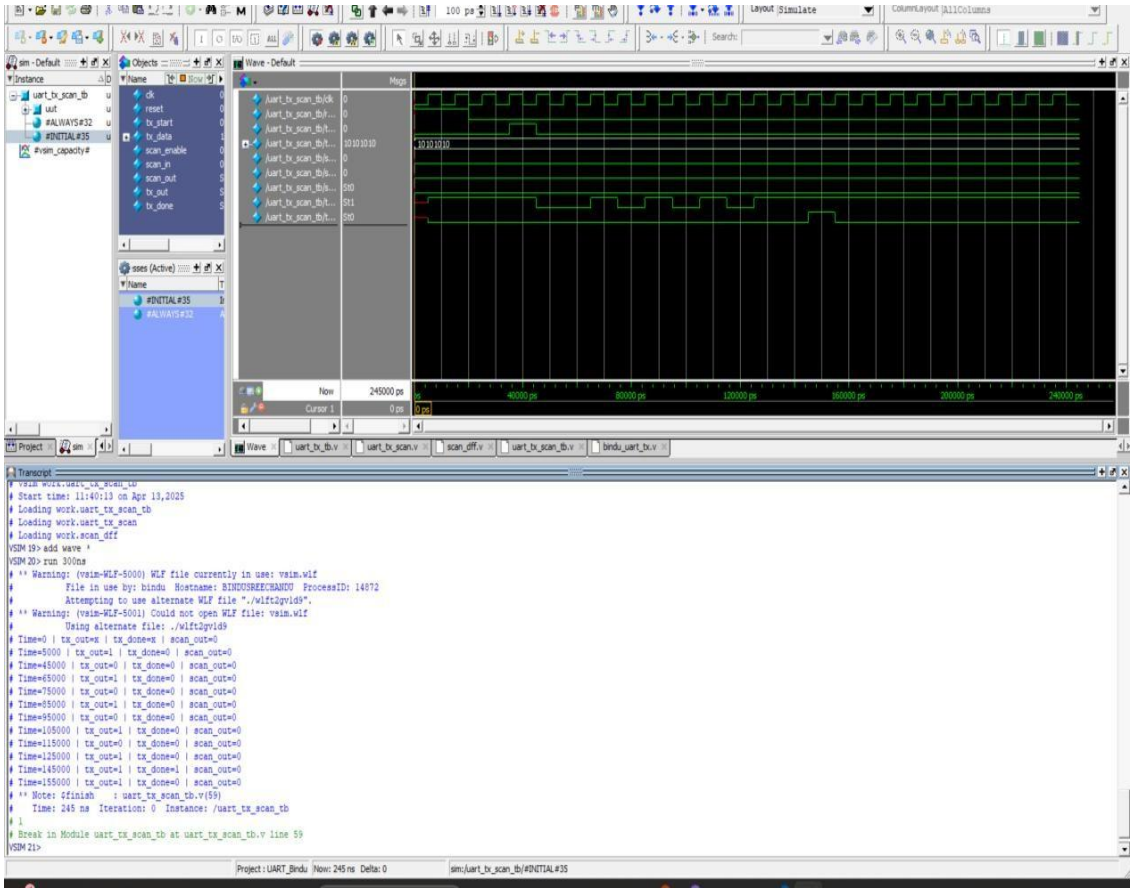


Figure 9 Scan UART Transmission Output & Signal Monitoring during Simulation

7.3 PHASE 3: LOGIC SYNTHESIS USING SYNOPSYS DC COMPILER

Logic synthesis was performed using Synopsys Design Compiler, which transforms RTL Verilog into a gate-level netlist mapped to a standard cell library. Synthesis ensures the RTL design is translated into a hardware-implementable gate-level netlist. It also provides reports for area, timing and power analysis, which are critical for physical implementation. I have completed the synthesis for both, UART transmitter without scan chains and Scan chain UART transmitter.

Tools Used: In logic synthesis, these are the tools used for generating the synthesis reports: Synopsys DC compiler, Apporto, Verilog code.

7.3.1 UART TRANSMITTER DESIGN WITHOUT SCAN CHAIN

Step 1: Logging in to the Cadence

- Login to Fresno State Apporto portal with the university credentials.
- Open the Cadence/Synopsys application by clicking the Launch button as shown below.

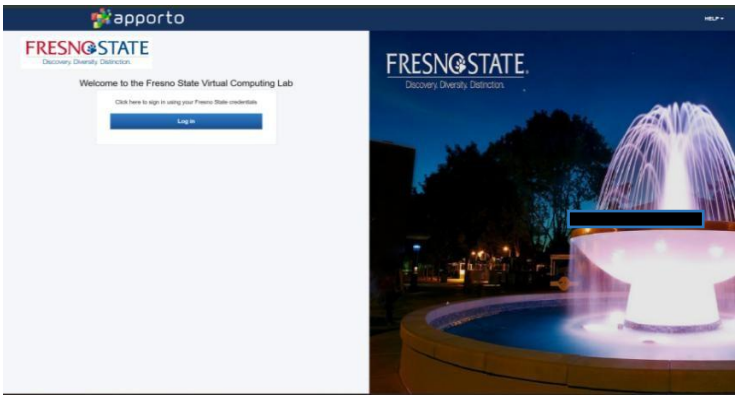


Figure 10 Apporto login page

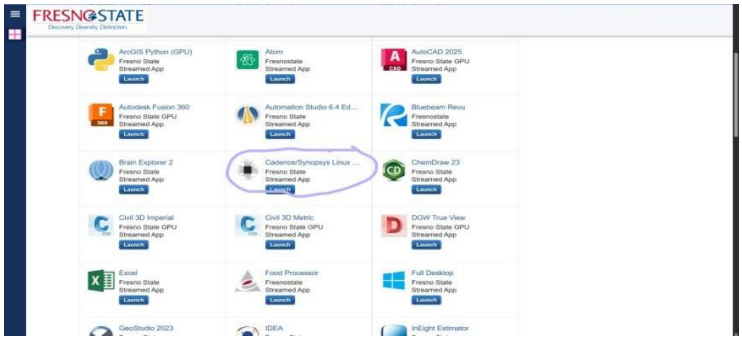


Figure 11 Launch Cadence

Step 2: File Organize

- The Cadence Synopsys environment will be loaded.
- Open the Home Folder and navigate to the Documents directory.

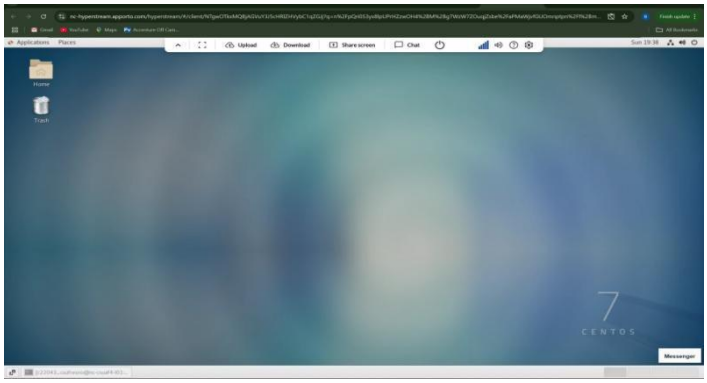


Figure 12 Home Folder

Step 3: Opening Terminal

- In the Documents folder, open the terminal . Using Terminal command

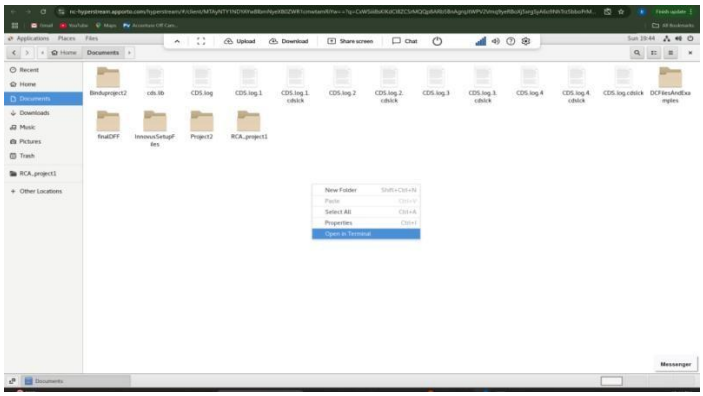


Figure 13 Opening Terminal

Step 4: Creating a New Folder

- Create a new folder by the terminal command: mkdir filename or by right click >> new folder
- Changing directory to the new folder via terminal

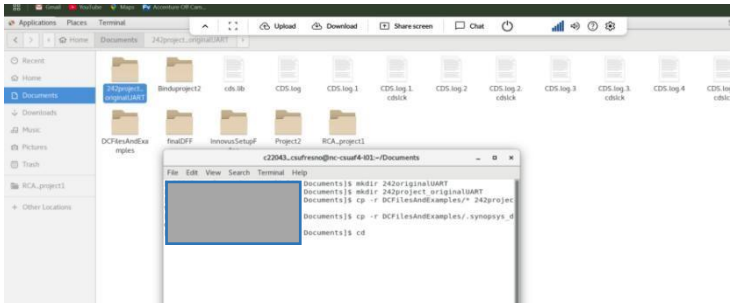


Figure 14 New Folder created (242project_originalUART)

Step 5: Synopsys DC compiler files

- Copy Synopsys DC compiler files by using following commands to the folder created
- Then, change directory to the new folder with cd command.

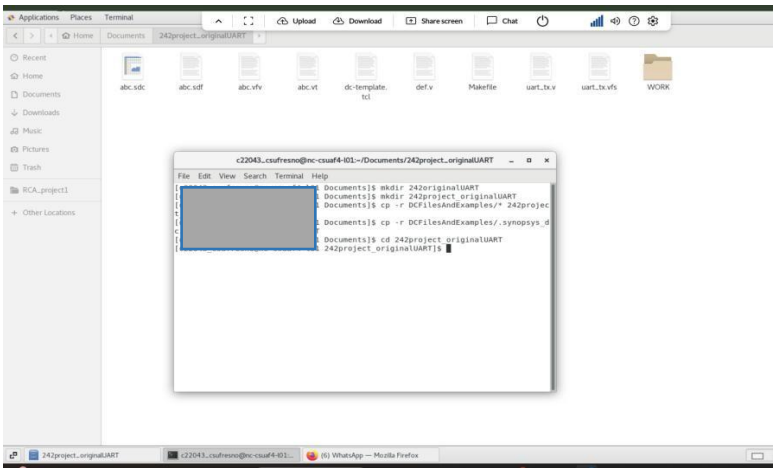


Figure 15 Synopsys DC Compiler Files

Step 6: Updating compiler files

- In the created folder, place Verilog code in abc.v file and rename it as top module name.

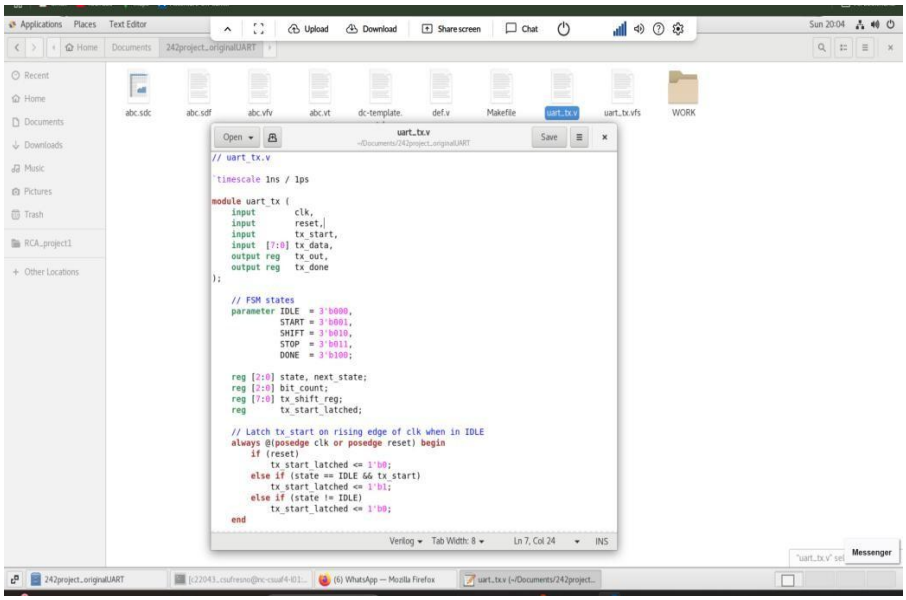


Figure 16 updating .v file and saving with top module name

- place the above .v file name in .vfs and .vfv files and rename them as top module name

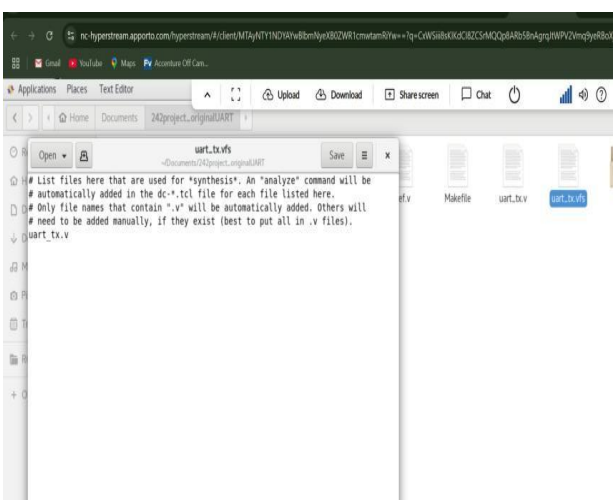


Figure 17 updating .vfs file

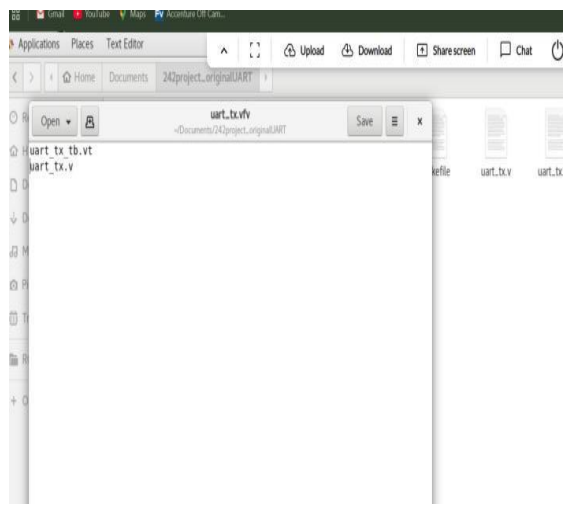


Figure 18 updating .vfv file

Step 7: To complete the synthesis

- Use the following command to complete the synthesis: make NAME=uart_tx synth

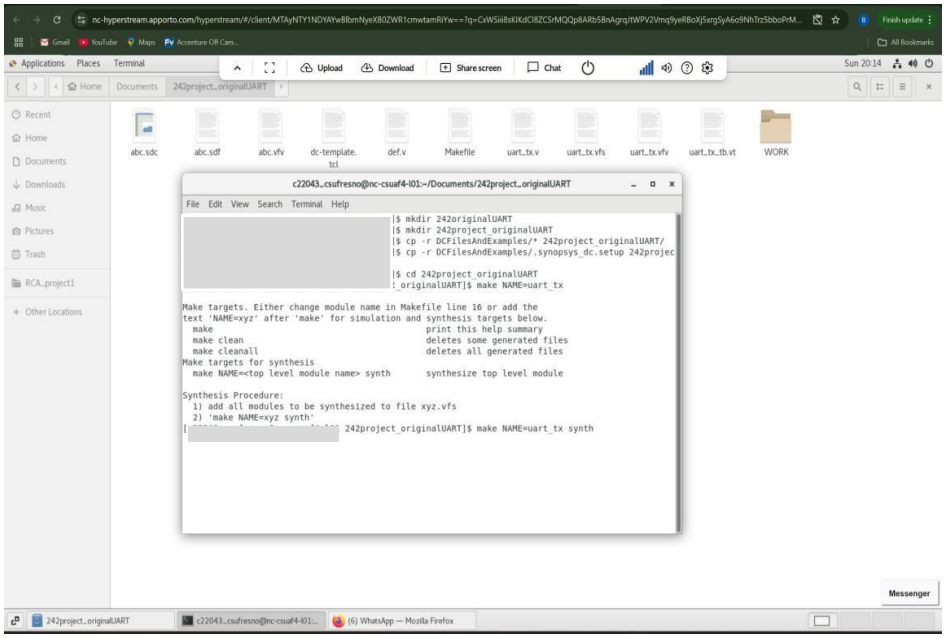


Figure 19 synthesis

- After successful synthesis, you can see thank you message in terminal and no errors.

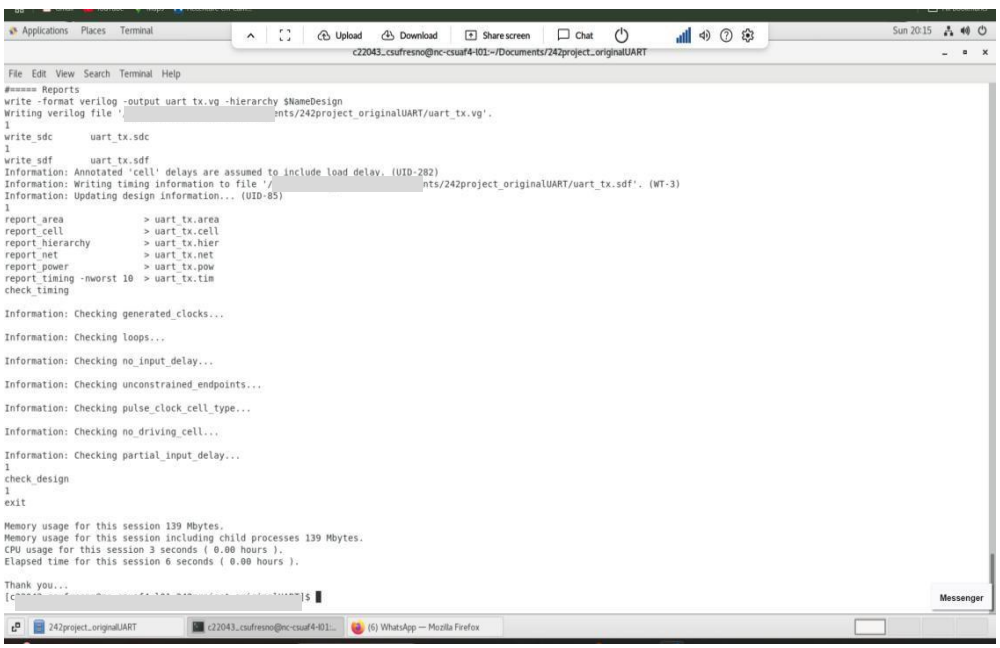


Figure 20 After successful synthesis

Step 8: RESULT: Finally, Report files generated in the created folder

- The area, power, timing and netlist files in the folder '242project_originalUART' generated after successful synthesis.

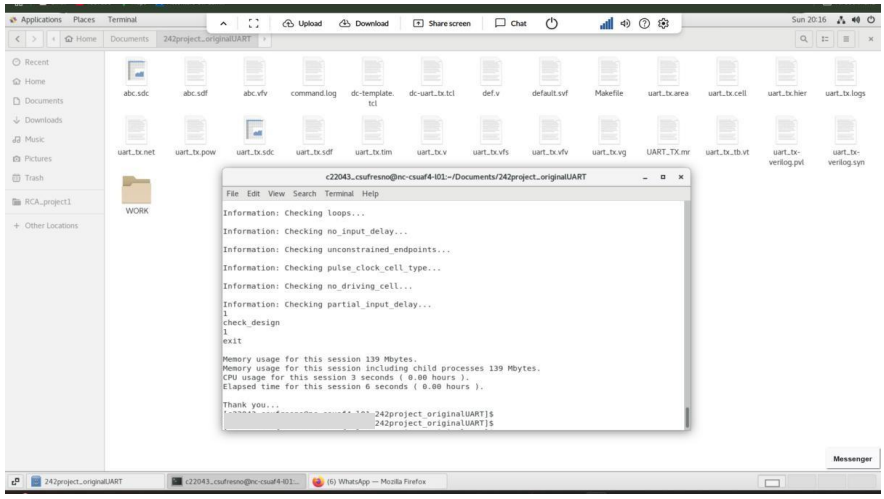


Figure 21: Area, power, Time, logs reports and netlist file generated

- The area, power, timing and netlist files in the folder '242project_originalUART' generated after successful synthesis.

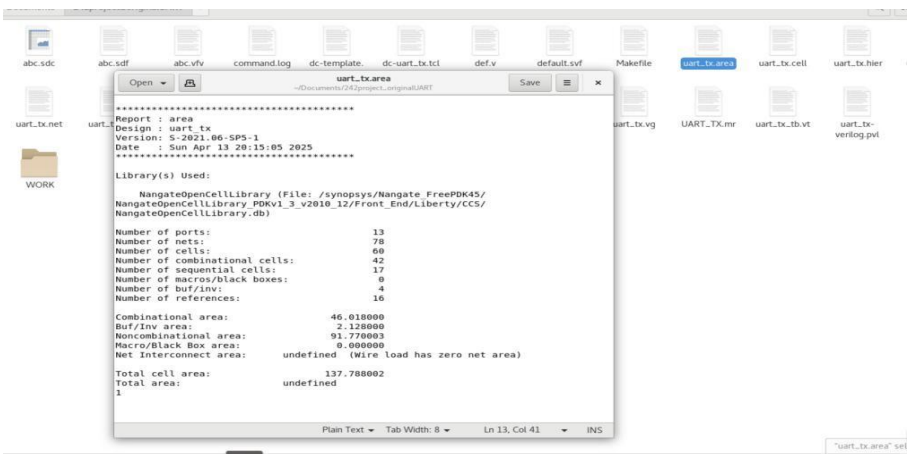


Figure 22 Area report for original UART design without scan

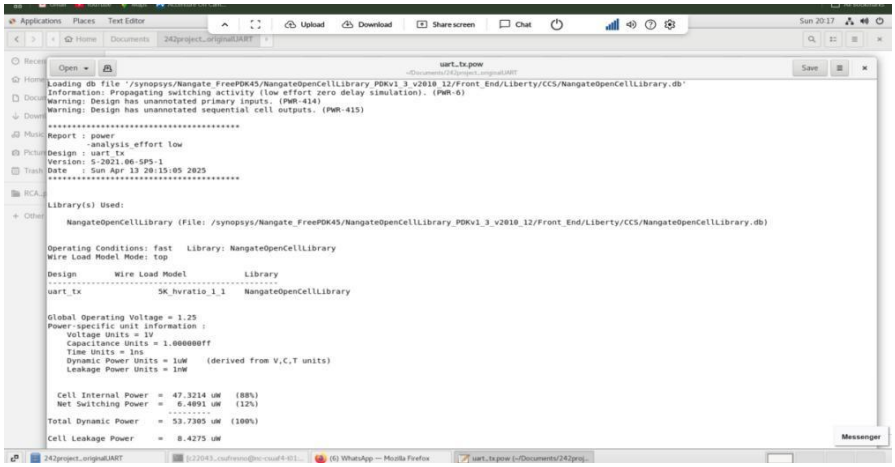
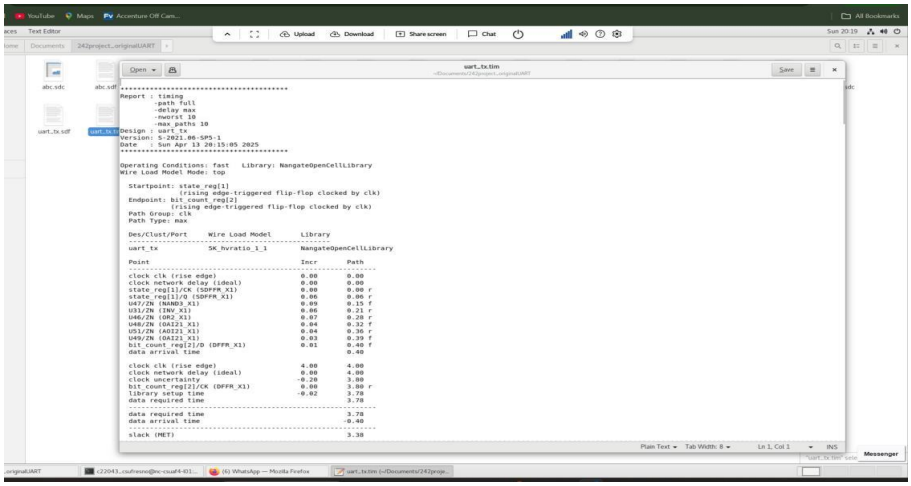


Figure 23 Power report for original UART design without scan



```
*****************************************
Report : timing
         -path full
         -delay max
         -nworst 10
         -max_paths 10
Design : uart_tx
Version: S-2021.06-SP5-1
Date   : Sun Apr 13 20:15:05 2025
*****************************************

Operating Conditions: fast   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

  Startpoint: state_reg[1]
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: bit_count_reg[2]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  uart_tx            5K_hvratio_1_1        NangateOpenCellLibrary

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clk (rise edge)                    0.00       0.00
  clock network delay (ideal)              0.00       0.00
  state_reg[1]/CK (SDFFR_X1)               0.00       0.00 r
  state_reg[1]/Q (SDFFR_X1)                0.06       0.06 r
  U47/ZN (NAND3_X1)                        0.09       0.15 f
  U31/ZN (INV_X1)                          0.06       0.21 r
  U46/ZN (OR2_X1)                          0.07       0.28 r
  U48/ZN (OAI21_X1)                        0.04       0.32 f
  U51/ZN (AOI21_X1)                        0.04       0.36 r
  U49/ZN (OAI21_X1)                        0.03       0.39 f
  bit_count_reg[2]/D (DFFR_X1)             0.01       0.40 f
  data arrival time                                   0.40

  clock clk (rise edge)                    4.00       4.00
  clock network delay (ideal)              0.00       4.00
  clock uncertainty                       -0.20       3.80
  bit_count_reg[2]/CK (DFFR_X1)            0.00       3.80 r
  library setup time                      -0.02       3.78
  data required time                                  3.78
  -----------------------------------------------------------
  data required time                                  3.78
  data arrival time                                  -0.40
  -----------------------------------------------------------
  slack (MET)                                         3.38
```

Figure 24 Timing report for original UART design without scan

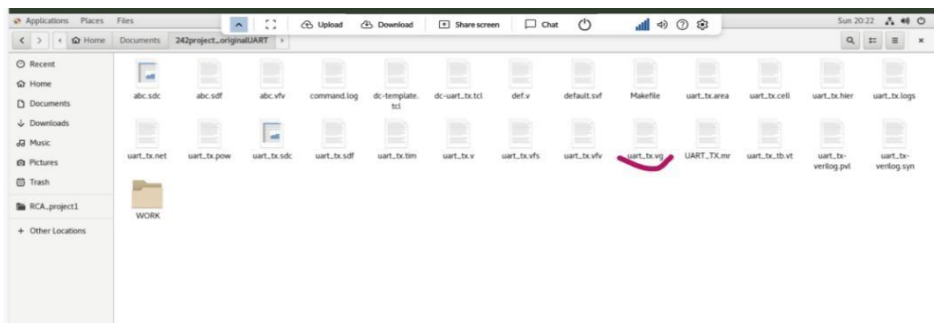
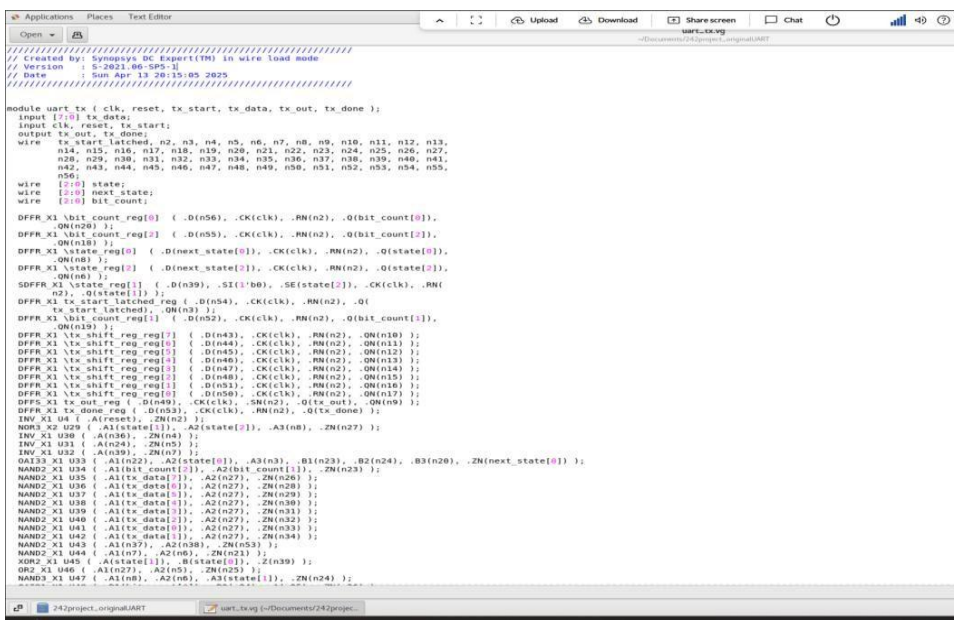


Figure 25 Netlist file also generated for original UART design without scan



```
////////////////////////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : S-2021.06-SP5-1
// Date      : Sun Apr 13 20:15:05 2025
////////////////////////////////////////////////////////////////////////////////


module uart_tx ( clk, reset, tx_start, tx_data, tx_out, tx_done );
  input [7:0] tx_data;
  input clk, reset, tx_start;
  output tx_out, tx_done;
  wire   tx_start_latched, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13,
         n14, n15, n16, n17, n18, n19, n20, n21, n22, n23, n24, n25, n26, n27,
         n28, n29, n30, n31, n32, n33, n34, n35, n36, n37, n38, n39, n40, n41,
         n42, n43, n44, n45, n46, n47, n48, n49, n50, n51, n52, n53, n54, n55,
         n56;
  wire   [2:0] state;
  wire   [2:0] next_state;
  wire   [2:0] bit_count;

  DFFR_X1 \bit_count_reg[0]  ( .D(n56), .CK(clk), .RN(n2), .Q(bit_count[0]),
       .QN(n20) );
  DFFR_X1 \bit_count_reg[2]  ( .D(n55), .CK(clk), .RN(n2), .Q(bit_count[2]),
       .QN(n18) );
  DFFR_X1 \state_reg[0]  ( .D(next_state[0]), .CK(clk), .RN(n2), .Q(state[0]),
       .QN(n8) );
  DFFR_X1 \state_reg[2]  ( .D(next_state[2]), .CK(clk), .RN(n2), .Q(state[2]),
       .QN(n6) );
  SDFFR_X1 \state_reg[1]  ( .D(n39), .SI(1'b0), .SE(state[2]), .CK(clk), .RN(
       n2), .Q(state[1]) );
  DFFR_X1 tx_start_latched_reg ( .D(n54), .CK(clk), .RN(n2), .Q(
       tx_start_latched), .QN(n3) );
  DFFR_X1 \bit_count_reg[1]  ( .D(n52), .CK(clk), .RN(n2), .Q(bit_count[1]),
       .QN(n19) );
  DFFR_X1 \tx_shift_reg_reg[7]  ( .D(n43), .CK(clk), .RN(n2), .QN(n10) );
  DFFR_X1 \tx_shift_reg_reg[6]  ( .D(n44), .CK(clk), .RN(n2), .QN(n11) );
  DFFR_X1 \tx_shift_reg_reg[5]  ( .D(n45), .CK(clk), .RN(n2), .QN(n12) );
  DFFR_X1 \tx_shift_reg_reg[4]  ( .D(n46), .CK(clk), .RN(n2), .QN(n13) );
  DFFR_X1 \tx_shift_reg_reg[3]  ( .D(n47), .CK(clk), .RN(n2), .QN(n14) );
  DFFR_X1 \tx_shift_reg_reg[2]  ( .D(n48), .CK(clk), .RN(n2), .QN(n15) );
  DFFR_X1 \tx_shift_reg_reg[1]  ( .D(n51), .CK(clk), .RN(n2), .QN(n16) );
  DFFR_X1 \tx_shift_reg_reg[0]  ( .D(n50), .CK(clk), .RN(n2), .QN(n17) );
  DFFS_X1 tx_out_reg ( .D(n49), .CK(clk), .SN(n2), .Q(tx_out), .QN(n9) );
  DFFR_X1 tx_done_reg ( .D(n53), .CK(clk), .RN(n2), .Q(tx_done) );
  INV_X1 U4 ( .A(reset), .ZN(n2) );
  NOR3_X2 U29 ( .A1(state[1]), .A2(state[2]), .A3(n8), .ZN(n27) );
  INV_X1 U30 ( .A(n36), .ZN(n4) );
  INV_X1 U31 ( .A(n24), .ZN(n5) );
  INV_X1 U32 ( .A(n39), .ZN(n7) );
  OAI33_X1 U33 ( .A1(n22), .A2(state[0]), .A3(n3), .B1(n23), .B2(n24), .B3(n20), .ZN(next_state[0]) );
  NAND2_X1 U34 ( .A1(bit_count[2]), .A2(bit_count[1]), .ZN(n23) );
  NAND2_X1 U35 ( .A1(tx_data[7]), .A2(n27), .ZN(n26) );
  NAND2_X1 U36 ( .A1(tx_data[6]), .A2(n27), .ZN(n28) );
  NAND2_X1 U37 ( .A1(tx_data[5]), .A2(n27), .ZN(n29) );
  NAND2_X1 U38 ( .A1(tx_data[4]), .A2(n27), .ZN(n30) );
  NAND2_X1 U39 ( .A1(tx_data[3]), .A2(n27), .ZN(n31) );
  NAND2_X1 U40 ( .A1(tx_data[2]), .A2(n27), .ZN(n32) );
  NAND2_X1 U41 ( .A1(tx_data[0]), .A2(n27), .ZN(n33) );
  NAND2_X1 U42 ( .A1(tx_data[1]), .A2(n27), .ZN(n34) );
  NAND2_X1 U43 ( .A1(n37), .A2(n38), .ZN(n53) );
  NAND2_X1 U44 ( .A1(n7), .A2(n6), .ZN(n21) );
  XOR2_X1 U45 ( .A(state[1]), .B(state[0]), .Z(n39) );
  OR2_X1 U46 ( .A1(n27), .A2(n5), .ZN(n25) );
  NAND3_X1 U47 ( .A1(n8), .A2(n6), .A3(state[1]), .ZN(n24) );
```

Figure 26 Generated Netlist file for UART design without scan.

7.3.2 UART TRANSMITTER DESIGN WITH SCAN CHAIN

Similarly for the Scan inserted UART transmitter design, it is required to generate the synthesis reports and netlist file. I repeated the above similar steps from 3 to 7 for the UART scan design codes to generate the report files for scan UART design.

Step 9: Creating a New Folder for Scan UART design

- Open the terminal and use command: mkdir 242project_UARTwithscan for folder creation.
- Changing directory to the new folder via terminal.

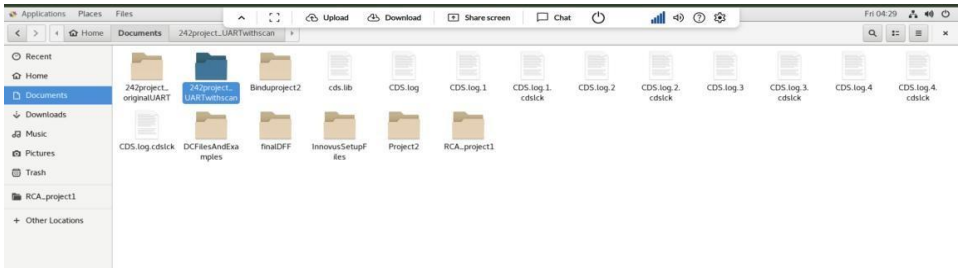


Figure 27 New Folder created (242project_UARTwithscan) and directory changed

Step 10: Synopsys DC compiler and synthesis

- Copy Synopsys DC compiler files by using commands to the scan UART folder created.
- DC compiler files can be seen in folder created. Then, change directory to the new folder.
- Similar to earlier design, now update the compiler files (abc.v, abc.vfs, abc.vfv)with new design code and save it with new top module name.
- Next step is completing the synthesis by using command in the terminal:

  make NAME=uart_tx_scan synth

- After successful synthesis, the analysis reports and netlist .vg file generated in the folder '242project_UARTwithscan'.

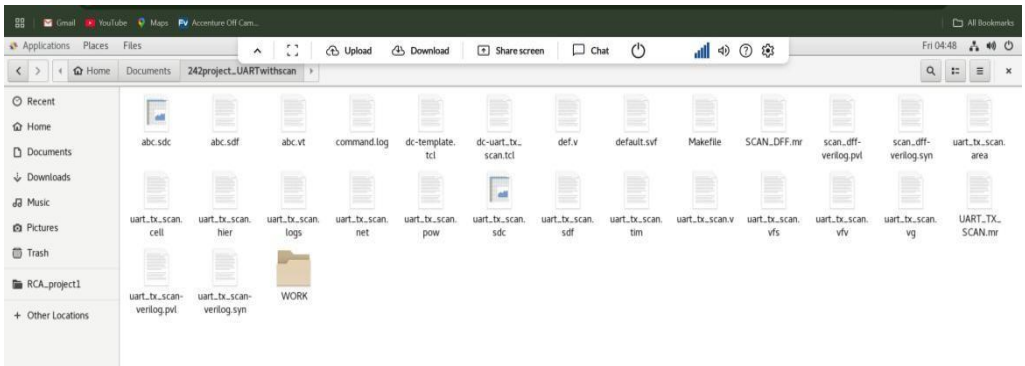


Figure 28 Generated report files after synthesis

Step 11: RESULTS: Synthesis Reports

- The area, power, timing and netlist files in the folder '242project_originalUART' generated after successful synthesis.

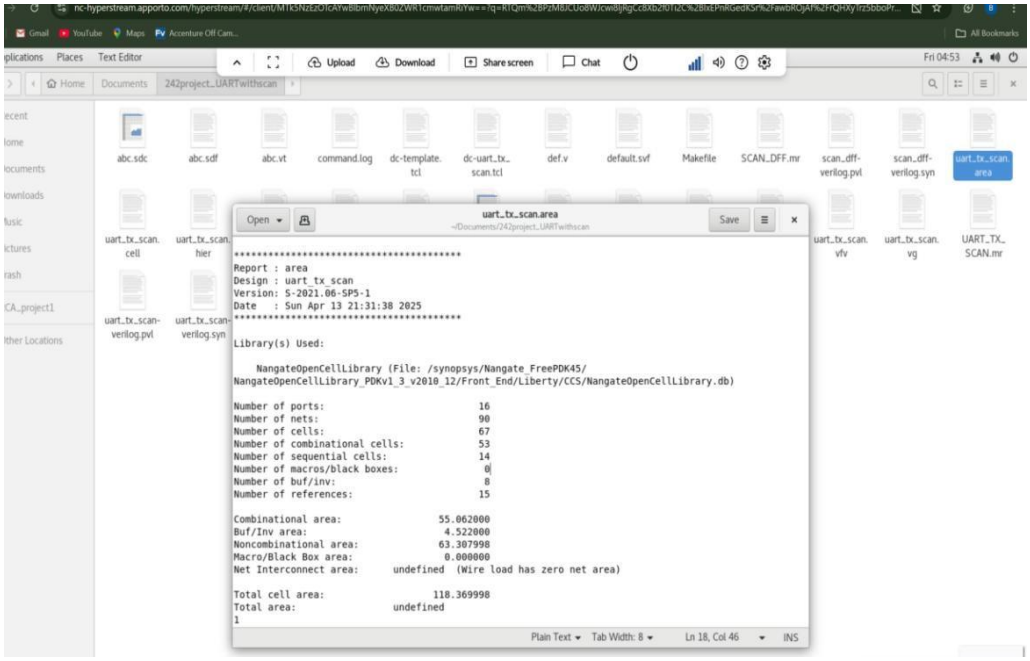


Figure 29 Area report for UART design with scan

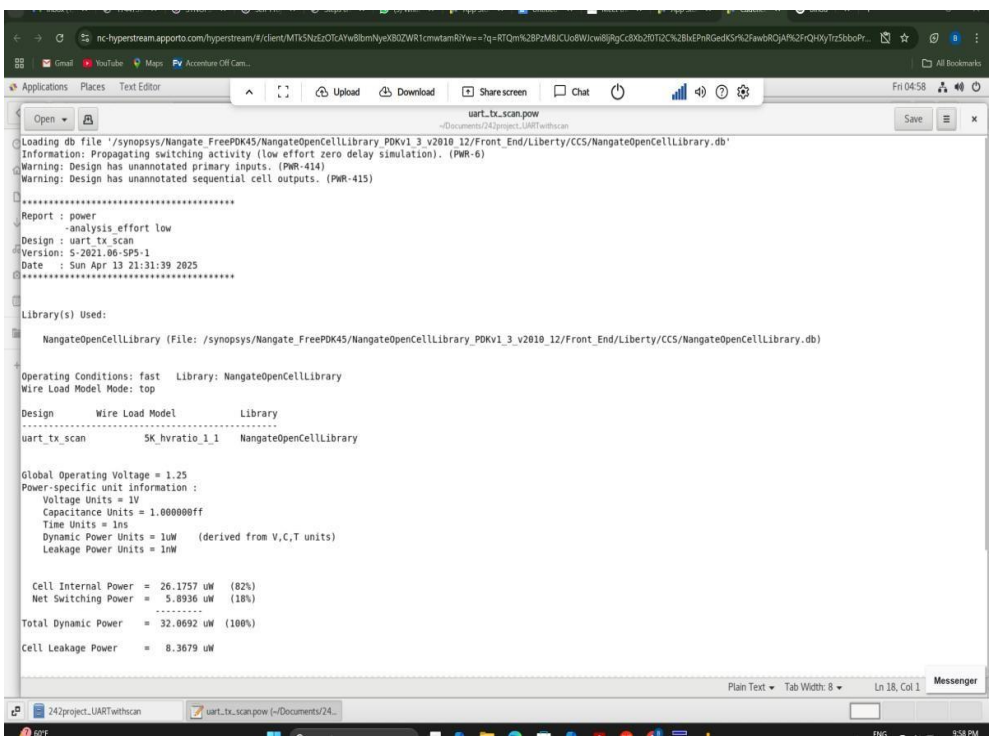
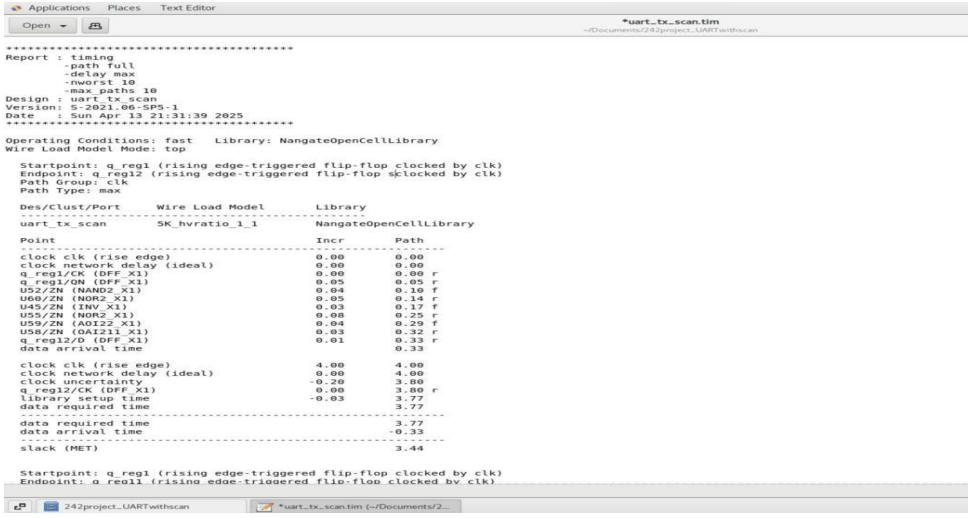


Figure 30 Power report for UART design with scan



```
Applications   Places   Text Editor
Open                                                    *uart_tx_scan.tim
                                                        ~/Documents/242project_UARTwithscan

****************************************
Report : timing
         -path full
         -delay max
         -nworst 10
         -max_paths 10
Design : uart_tx_scan
Version: S-2021.06-SP5-1
Date   : Sun Apr 13 21:31:39 2025
****************************************

Operating Conditions: fast   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

  Startpoint: q_reg1 (rising edge-triggered flip-flop clocked by clk)
  Endpoint: q_reg12 (rising edge-triggered flip-flop s|clocked by clk)
  Path Group: clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------------
  uart_tx_scan       5K_hvratio_1_1        NangateOpenCellLibrary

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clk (rise edge)                    0.00       0.00
  clock network delay (ideal)              0.00       0.00
  q_reg1/CK (DFF_X1)                       0.00       0.00 r
  q_reg1/QN (DFF_X1)                       0.05       0.05 r
  U52/ZN (NAND2_X1)                        0.04       0.10 f
  U60/ZN (NOR2_X1)                         0.05       0.14 r
  U45/ZN (INV_X1)                          0.03       0.17 f
  U55/ZN (NOR2_X1)                         0.08       0.25 r
  U59/ZN (AOI22_X1)                        0.04       0.29 f
  U58/ZN (OAI211_X1)                       0.03       0.32 r
  q_reg12/D (DFF_X1)                       0.01       0.33 r
  data arrival time                                   0.33

  clock clk (rise edge)                    4.00       4.00
  clock network delay (ideal)              0.00       4.00
  clock uncertainty                       -0.20       3.80
  q_reg12/CK (DFF_X1)                      0.00       3.80 r
  library setup time                      -0.03       3.77
  data required time                                  3.77
  -----------------------------------------------------------
  data required time                                  3.77
  data arrival time                                  -0.33
  -----------------------------------------------------------
  slack (MET)                                         3.44

  Startpoint: q_reg1 (rising edge-triggered flip-flop clocked by clk)
  Endpoint: q_reg11 (rising edge-triggered flip-flop clocked by clk)
242project_UARTwithscan    *uart_tx_scan.tim (~/Documents/2...
```

Figure 31 Timings report for UART design with scan

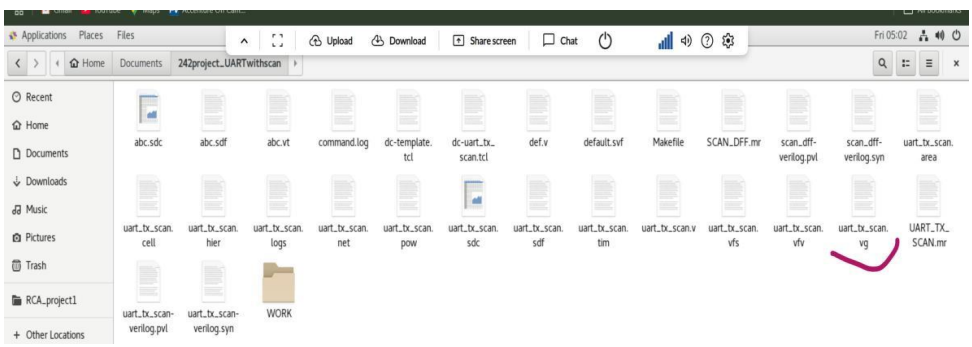
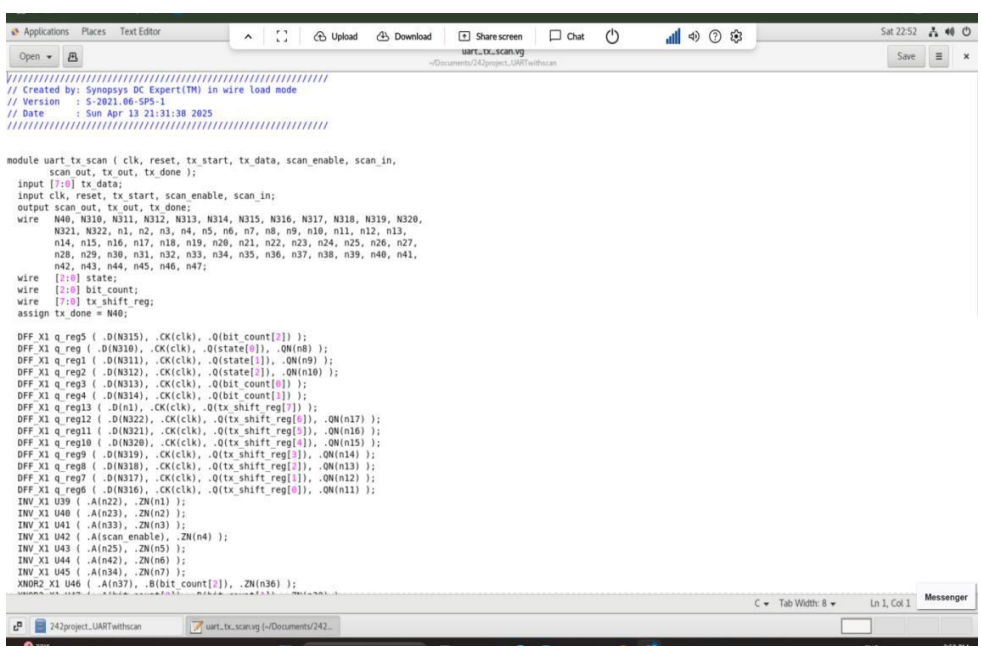


Figure 32 Netlist file generated for UART design with scan.



```
Applications   Places   Text Editor
Upload   Download   Share screen   Chat                                   Sat 22:52
Open                                uart_tx_scan.vg                        Save
                                    ~/Documents/242project_UARTwithscan

//////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : S-2021.06-SP5-1
// Date      : Sun Apr 13 21:31:38 2025
//////////////////////////////////////////////////////////////


module uart_tx_scan ( clk, reset, tx_start, tx_data, scan_enable, scan_in,
        scan_out, tx_out, tx_done );
  input [7:0] tx_data;
  input clk, reset, tx_start, scan_enable, scan_in;
  output scan_out, tx_out, tx_done;
  wire   N40, N310, N311, N312, N313, N314, N315, N316, N317, N318, N319, N320,
         N321, N322, n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13,
         n14, n15, n16, n17, n18, n19, n20, n21, n22, n23, n24, n25, n26, n27,
         n28, n29, n30, n31, n32, n33, n34, n35, n36, n37, n38, n39, n40, n41,
         n42, n43, n44, n45, n46, n47;
  wire   [2:0] state;
  wire   [2:0] bit_count;
  wire   [7:0] tx_shift_reg;
  assign tx_done = N40;

  DFF_X1 q_reg5 ( .D(N315), .CK(clk), .Q(bit_count[2]) );
  DFF_X1 q_reg ( .D(N310), .CK(clk), .Q(state[0]), .QN(n8) );
  DFF_X1 q_reg1 ( .D(N311), .CK(clk), .Q(state[1]), .QN(n9) );
  DFF_X1 q_reg2 ( .D(N312), .CK(clk), .Q(state[2]), .QN(n10) );
  DFF_X1 q_reg3 ( .D(N313), .CK(clk), .Q(bit_count[0]) );
  DFF_X1 q_reg4 ( .D(N314), .CK(clk), .Q(bit_count[1]) );
  DFF_X1 q_reg13 ( .D(n1), .CK(clk), .Q(tx_shift_reg[7]) );
  DFF_X1 q_reg12 ( .D(N322), .CK(clk), .Q(tx_shift_reg[6]), .QN(n17) );
  DFF_X1 q_reg11 ( .D(N321), .CK(clk), .Q(tx_shift_reg[5]), .QN(n16) );
  DFF_X1 q_reg10 ( .D(N320), .CK(clk), .Q(tx_shift_reg[4]), .QN(n15) );
  DFF_X1 q_reg9 ( .D(N319), .CK(clk), .Q(tx_shift_reg[3]), .QN(n14) );
  DFF_X1 q_reg8 ( .D(N318), .CK(clk), .Q(tx_shift_reg[2]), .QN(n13) );
  DFF_X1 q_reg7 ( .D(N317), .CK(clk), .Q(tx_shift_reg[1]), .QN(n12) );
  DFF_X1 q_reg6 ( .D(N316), .CK(clk), .Q(tx_shift_reg[0]), .QN(n11) );
  INV_X1 U39 ( .A(n22), .ZN(n1) );
  INV_X1 U40 ( .A(n23), .ZN(n2) );
  INV_X1 U41 ( .A(n33), .ZN(n3) );
  INV_X1 U42 ( .A(scan_enable), .ZN(n4) );
  INV_X1 U43 ( .A(n25), .ZN(n5) );
  INV_X1 U44 ( .A(n42), .ZN(n6) );
  INV_X1 U45 ( .A(n34), .ZN(n7) );
  XNOR2_X1 U46 ( .A(n37), .B(bit_count[2]), .ZN(n36) );
C   Tab Width: 8   Ln 1, Col 1   Messenger
242project_UARTwithscan   uart_tx_scan.vg (~/Documents/242...
```

Figure 33 Netlist file for UART design with scan.

7.4 PHASE 4: GATE-LEVEL SIMULATION

Gate-level simulation plays a critical role in validating that the synthesized netlist preserves the intended RTL behavior of the design. In this phase, the Verilog netlist generated by Synopsys Design Compiler was simulated using ModelSim to ensure functional equivalence with the original RTL implementation.

Step 12:

- Here, added the scan UART netlist file, testbench file, and library file.
- The synthesized netlist file (uart_tx_scan.vg) was exported from Synopsys Design Compiler.
- A new ModelSim project was created and included:
    1. The gate-level netlist
    2. Corresponding testbench file
    3. Nangate 45nm OpenCellLibrary (NangateOpenCellLibrary.v)
- All files compiled and simulated using

```
vlog uart_tx_tb.v uart_tx.vg NangateOpenCellLibrary.v
vsim -suppress 3009 work.uart_tx_tb
```

- Warnings due to missing timeunit/timeprecision in library files were suppressed using the -suppress 3009 flag.
- Simulation waveforms were monitored for signals tx_out, tx_done, and scan_out (for scan version).
- Functional behavior was validated against expected UART transmission sequence.

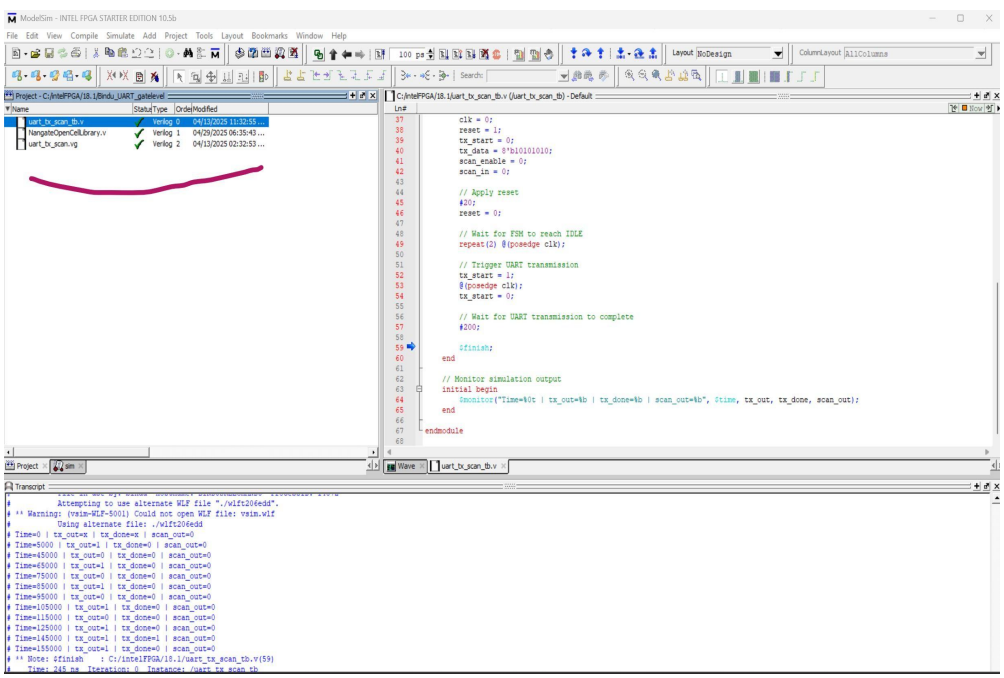


Figure 34: Gate level simulation in modelsim

7.5 PHASE 4: SCAN INSERTION AND DFT VALIDATION

In this phase, scan flip-flops were manually inserted into the UART transmitter design to improve testability. Sequential elements were replaced with scan-enabled DFFs, forming a single scan chain using scan_in, scan_out, and scan_enable control signals. The updated RTL was simulated in ModelSim to verify correct scan behavior and ensure no functional disruption. The modified design was then synthesized again using Synopsys Design Compiler to analyze the impact of scan integration on area, power and timing. This validated the scan-based Design-for-Testability approach for effective post-silicon testing.

7.6 PHASE 5: TEST COVERAGE ANALYSIS

In this phase, the scan-inserted UART transmitter netlist was prepared for ATPG using Synopsys TetraMAX. The synthesized gate-level .vg file along with the standard cell library was loaded into the tool. The stuck-at fault model was used to simulate realistic fault scenarios and generate test patterns that target maximum fault coverage.

Although full ATPG execution encountered tool-related constraints, the process was initiated correctly with scan chains identified and fault models defined. The expected outcome from this step was to analyze fault coverage. The scan-based DFT design facilitates this process by improving observability and controllability, enabling robust testability within complex digital systems.

Step 12: TetraMax

- In VM manager, Import the Centos VM where Tetramax application exists.
- Copy the required files, library file, .vg file and main module file into VM. I used shared folder and mounted while importing VM.
- In terminal, open tetramax by **tmax &** command.

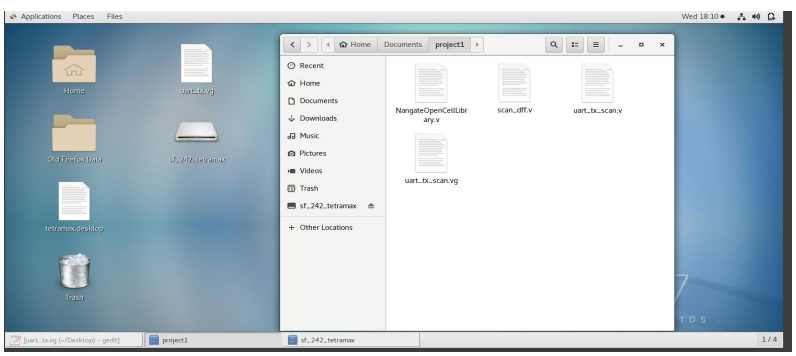


Figure 35: Mount the folder and copying into Centos VM

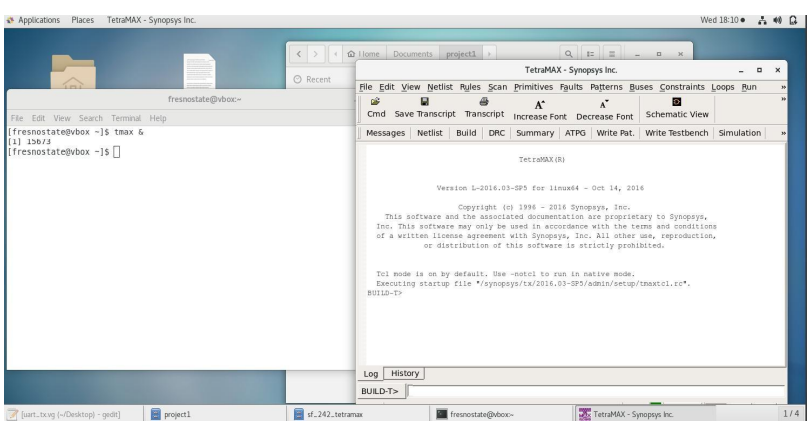


Figure 36: Opening tetramax by command in terminal

- Click on Netlist option and add library file into it and keep rest all settings defaults and then, click on run.
- It gives message as shown below figure.

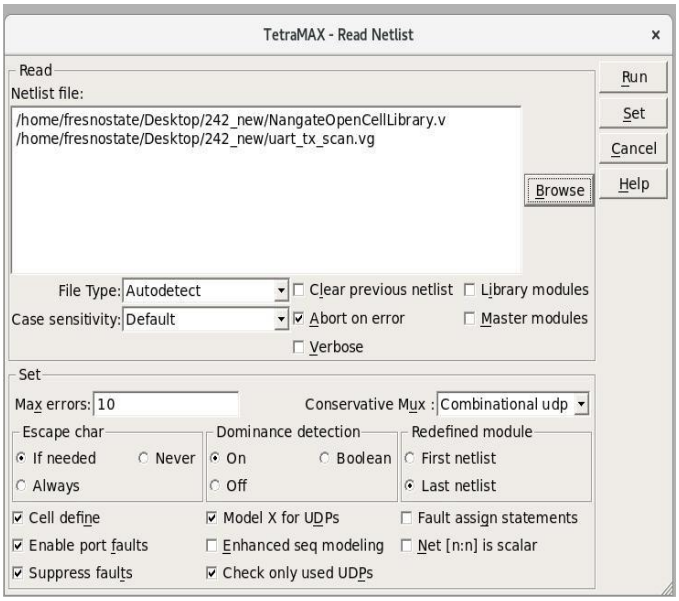


Figure: 37 Adding netlist file and library files

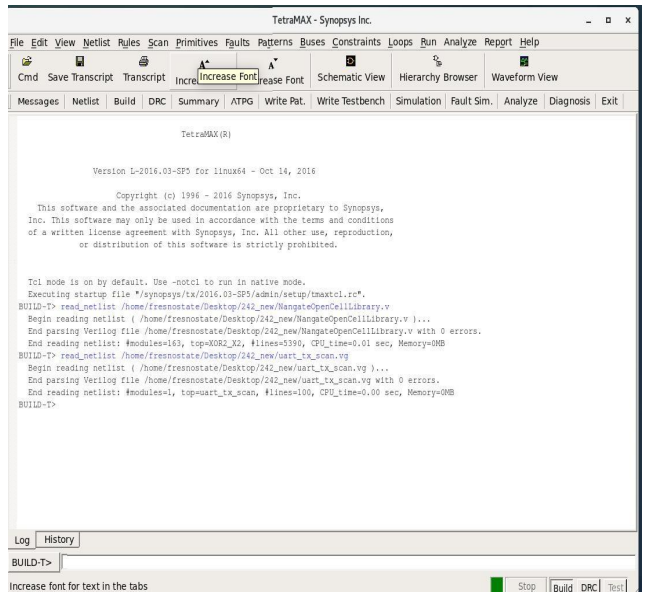


Figure 38: After Running Netlist

- Next, click on Build model option and it automatically loads top module name into it and then, click on run.
- It gives message as shown below figure after building model.

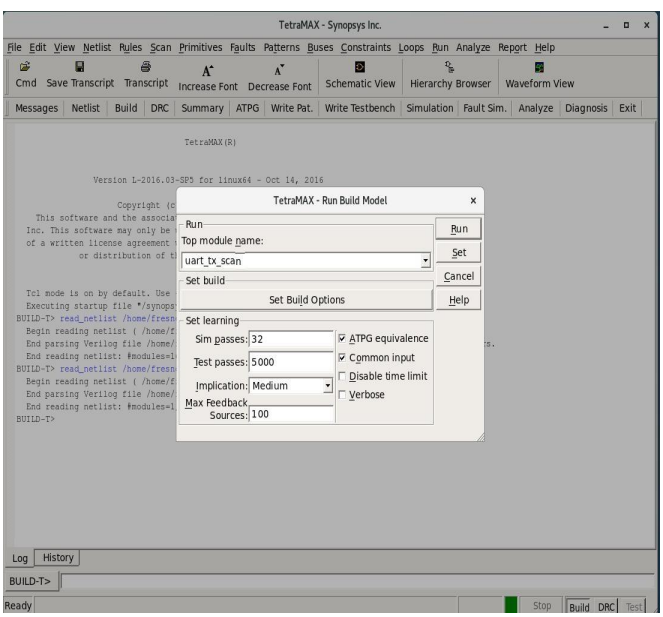


Figure 39: Build model, loads top module name

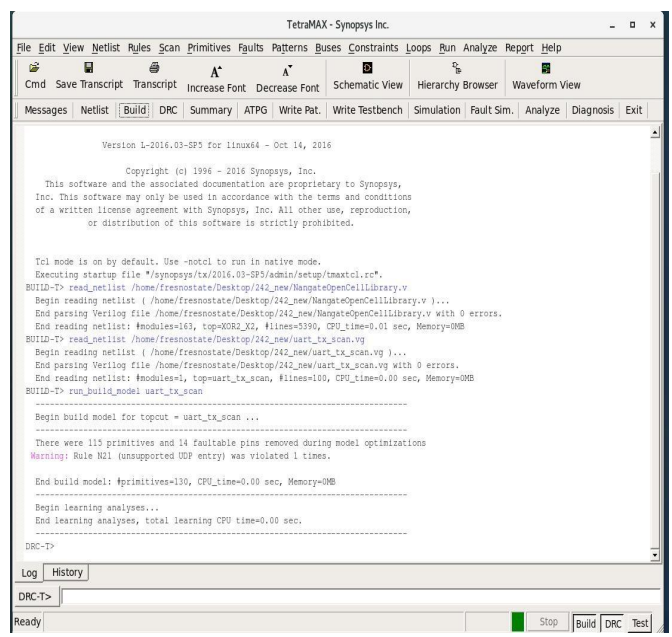


Figure 40: After running Build model

- Click DRC option, and keep all settings by default. Figure shows the message after running DRC check.

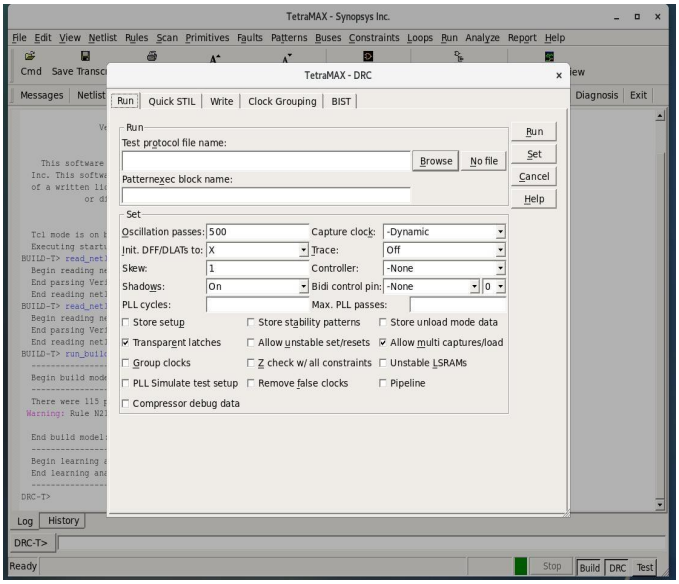


Figure 41: Open DRC

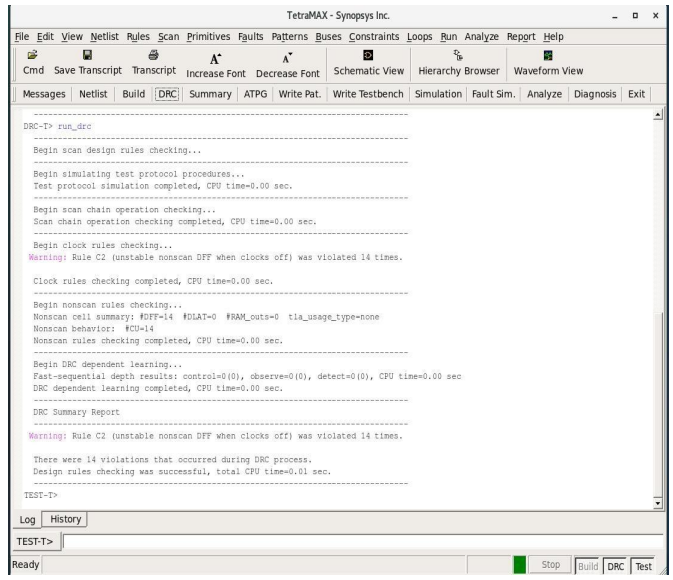


Figure 42: After running DRC check

- Click on TetraMax report summary and I kept all settings bydefault and make sure to check faults and patterns to obtain summary report.
- After this, Click on ATPG right after the summary report. In ATPG window, increase ‘capture cycles’ to 2 and check the ‘Enable Full-seq ATPG’ and click on add all faults of stuck type and Auto the ATPG.
- All settings shown in below figure for ATPG.

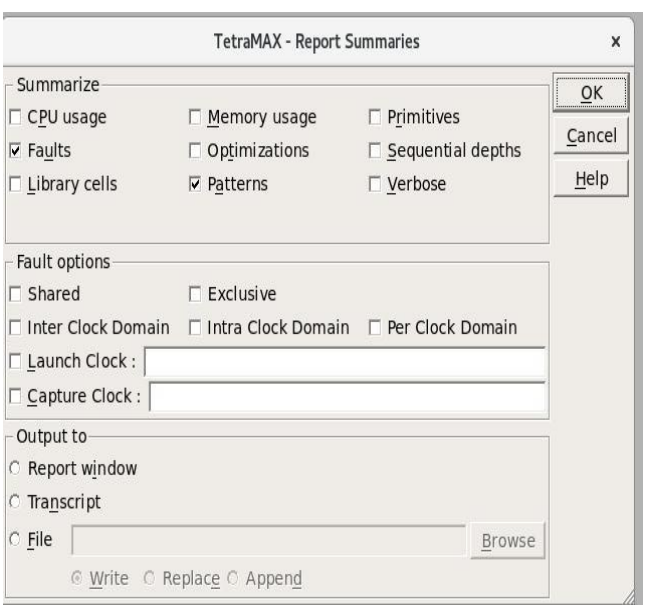


Figure 43: Report Summary, keep default

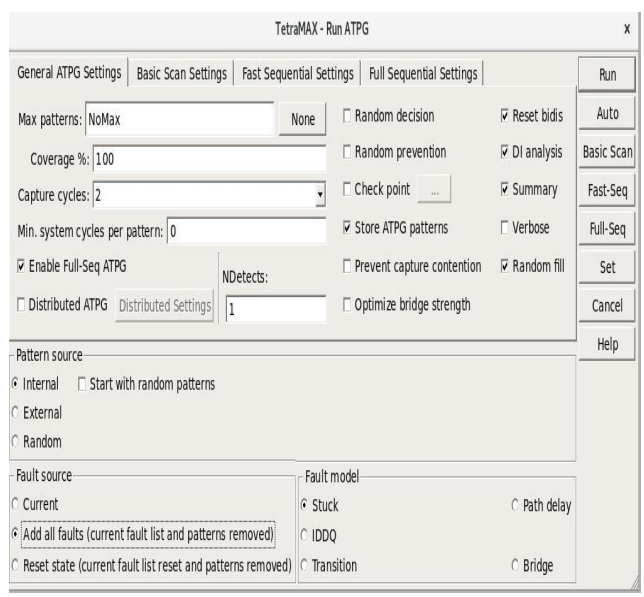


Figure 44: Run ATPG by this settings

- The message shows the progress of ATPG and the test coverage. I got 82% test coverage for 482 faults.

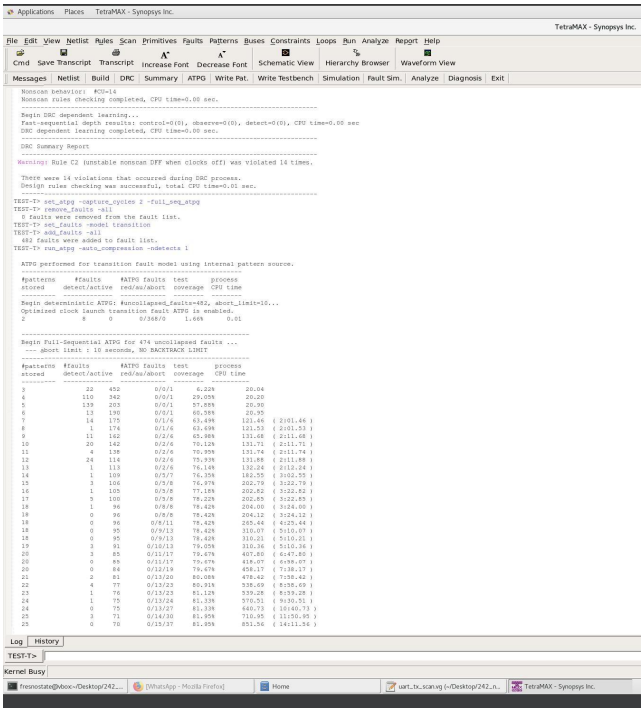


Figure 45: After running ATPG,takes time

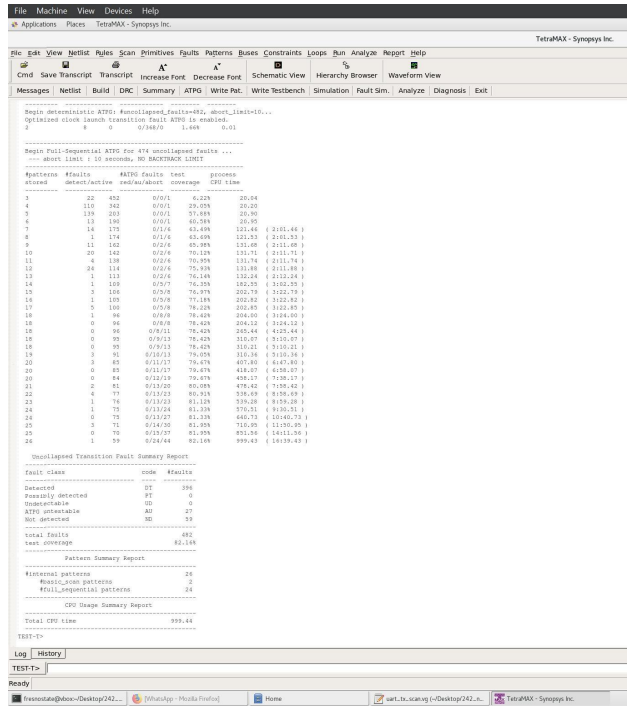


Figure 46: Test coverage for design

- Now, for writing test patterns, give a name to pattern file and click OK. In few seconds, could see the pattern file generated in folders directory.

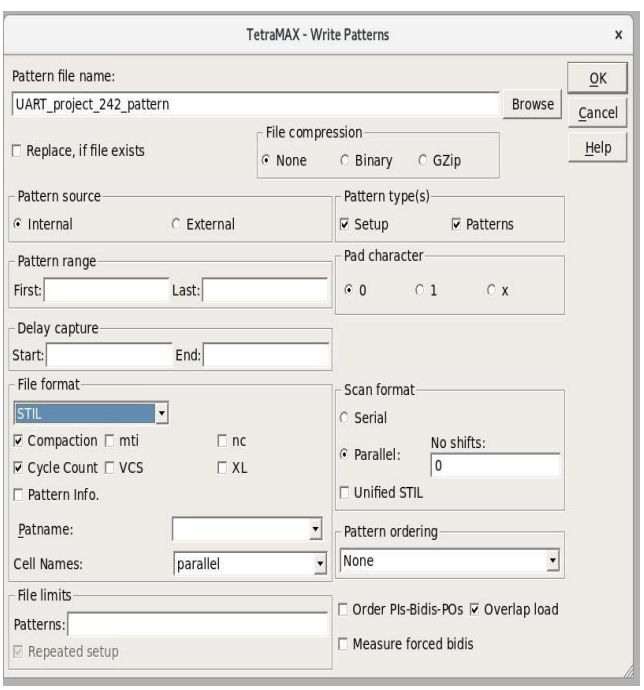


Figure 47: Writing patterns

Figure 48: Pattern file created for design

- For Schematic view of the design, click on schematic and then Setup option. Select Design hierarcy.
- Click on show and select All. This gives schematic view as shown in below figure

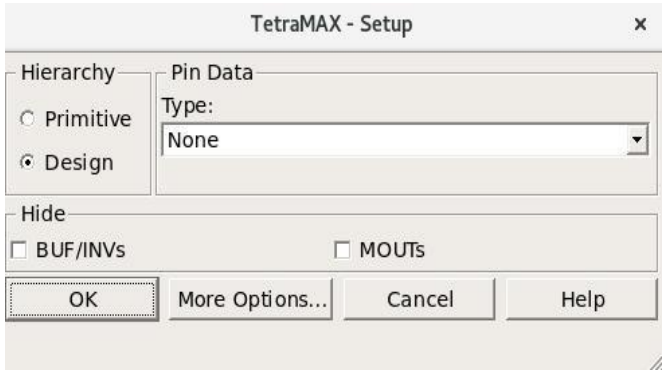


Figure 49: Setup for schematic design

Figure 50: Selecting ALL to view entire design

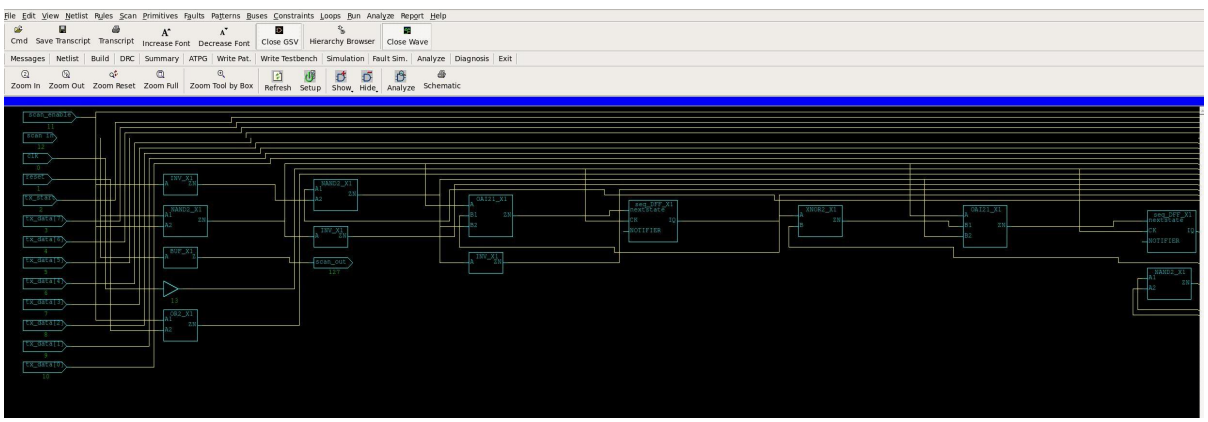


Figure 51: Schematic design zoom in

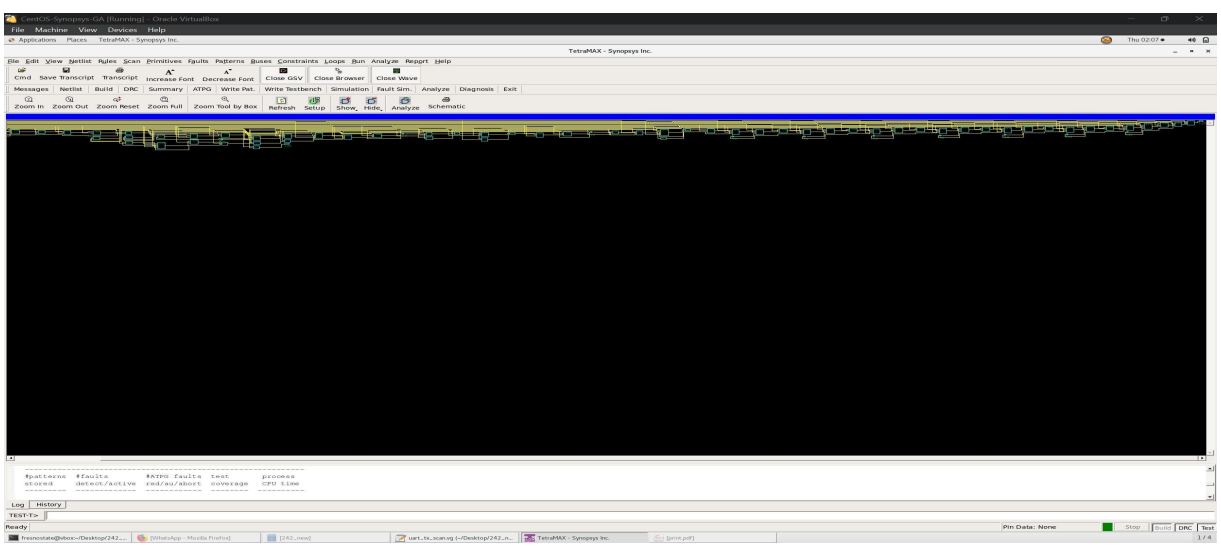


Figure 52: Entire Schematic View

- To save the schematics for future, click on print option ans it saves as png file. This saved image can be found in the directory folder.

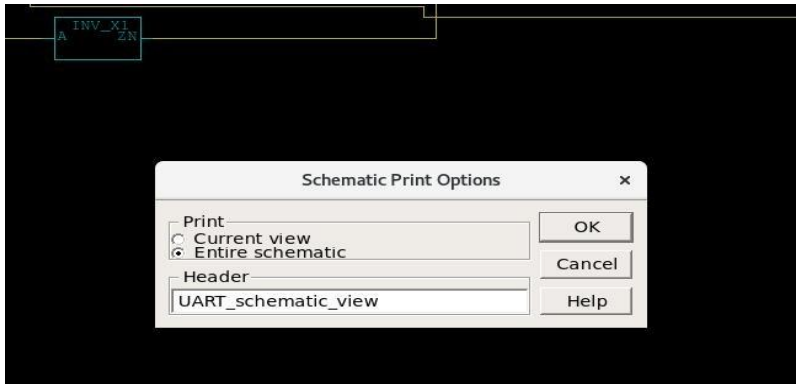


Figure 53: For saving the schematics

8. RESULTS, COMPARISONS AND ANALYSIS OF THE DATA OBTAINED

8.1 MODELSIM OUTPUT AND WAVEFORMS OF BOTH DESIGNS

The results from ModelSim aligned with the expected UART operation and provided additional visual confirmation of the design's correctness.

DATA ANALYSIS:

The UART transmitter outputs the correct serial data for input 0x33. Start and stop bits are properly aligned with clock cycles. Waveform confirms correct UART operation without scan insertion.

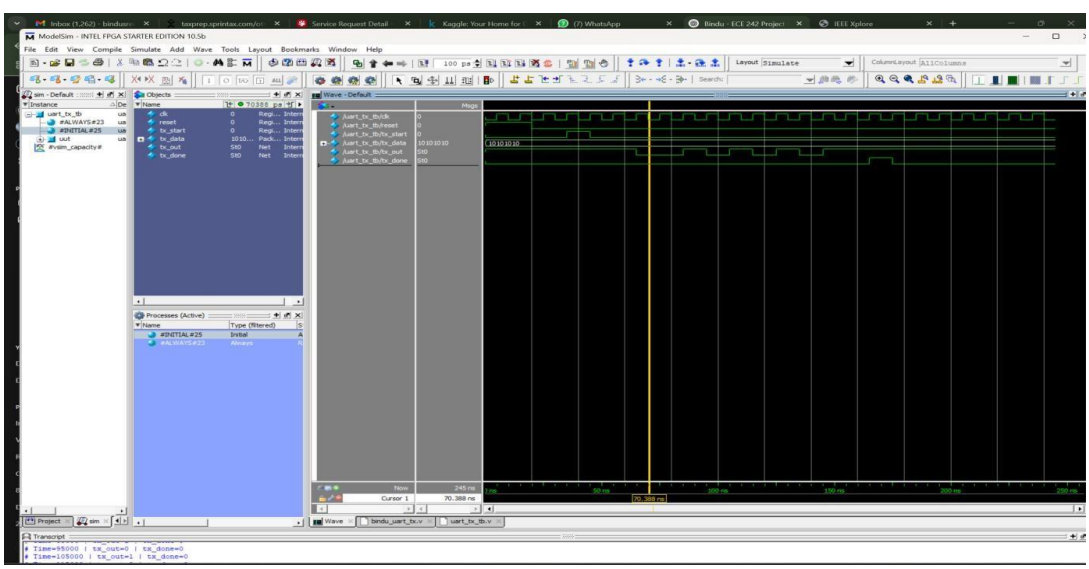


Figure 54 UART Transmitter with no scan design output waveform

The UART transmitter correctly serializes data 0x33 even with scan logic enabled. Scan signals (scan_en, scan_in, scan_out) do not interfere with normal operation. Waveform confirms scan insertion preserves functional correctness of the UART design.

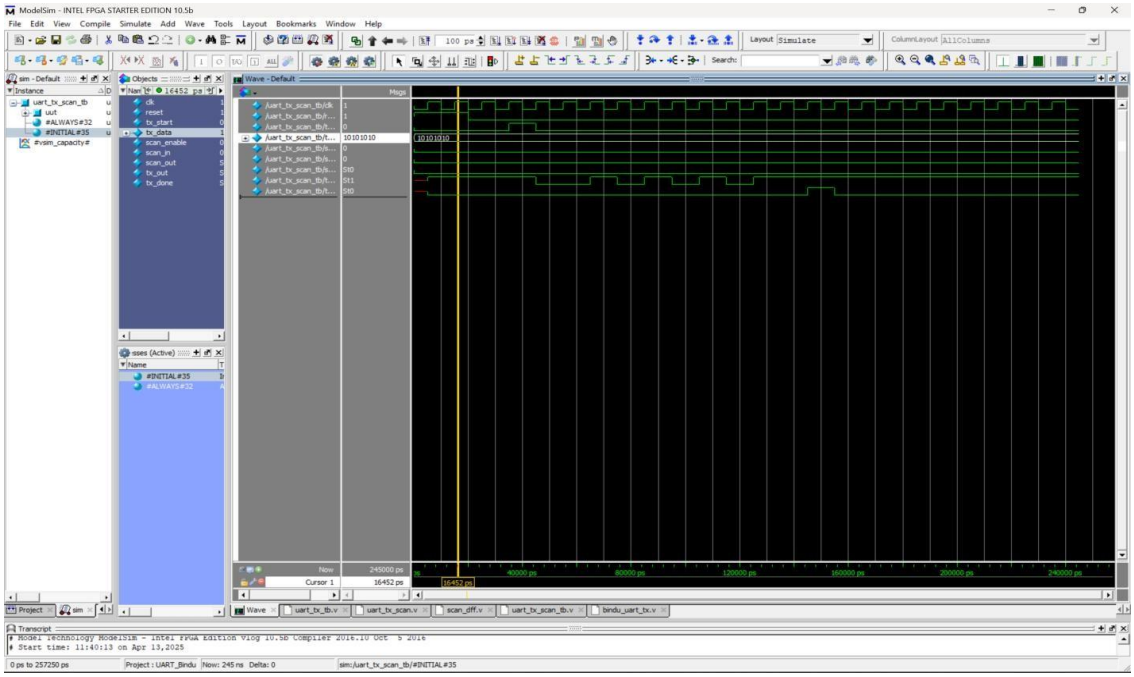


Figure 55 UART Transmitter with scan design output waveform

8.2 LOGIC SYNTHESIS COMPARISONS OF UART WITH AND WITHOUT SCAN

Metric	Value
Library	Nangate 45nm
Total Cell Area	137.79 μm²
Dynamic Power	53.73 μW
Leakage Power	8.43 μW
Total Power	62.16 μW
Cell Count	60
Critical Path Delay	0.39 ns

Figure: Synthesis Analysis of UART Transmitter Without Scan

Metric	Value
Library	Nangate 45nm
Total Cell Area	118.37 μm²
Dynamic Power	32.07 μW
Leakage Power	8.37 μW
Total Power	40.44 μW
Cell Count	67
Critical Path Delay	0.33 ns

Figure: Synthesis Analysis of UART Transmitter with Scan

Metric	UART Without Scan	UART With Scan	Better Option
Library	Nangate 45nm	Nangate 45nm	—
Total Cell Area	137.79 µm²	**118.37 µm²**	✔ UART With Scan
Dynamic Power	53.73 µW	**32.07 µW**	✔ UART With Scan
Leakage Power	8.43 µW	**8.37 µW**	✔ UART With Scan
Total Power	62.16 µW	**40.44 µW**	✔ UART With Scan
Cell Count	**60**	67	✔ UART Without Scan
Critical Path Delay	0.39 ns	**0.33 ns**	✔ UART With Scan

Figure 56: Synthesis Comparison – UART Without Scan vs UART With Scan

A comparative synthesis analysis was conducted using the Nangate 45nm library for both the original UART and the scan-enabled version. The scan-inserted UART achieved lower total cell area (118.37 µm² vs 137.79 µm²) and reduced dynamic power consumption (32.07 µW vs 53.73 µW), despite a slight increase in cell count due to added scan flip-flops. Furthermore, the critical path delay improved from 0.39 ns to 0.33 ns. These results indicate that the scan-enabled design not only improves testability but also contributes to better power and performance efficiency, making it the superior choice for implementation.

8.3 GATE LEVEL SIMULATION

Simulation waveforms confirmed that the gate-level design produced the correct UART output sequence, indicating that synthesis preserved logical functionality. The scan-inserted design also showed correct scan chain behavior, validating scan insertion correctness at gate level.

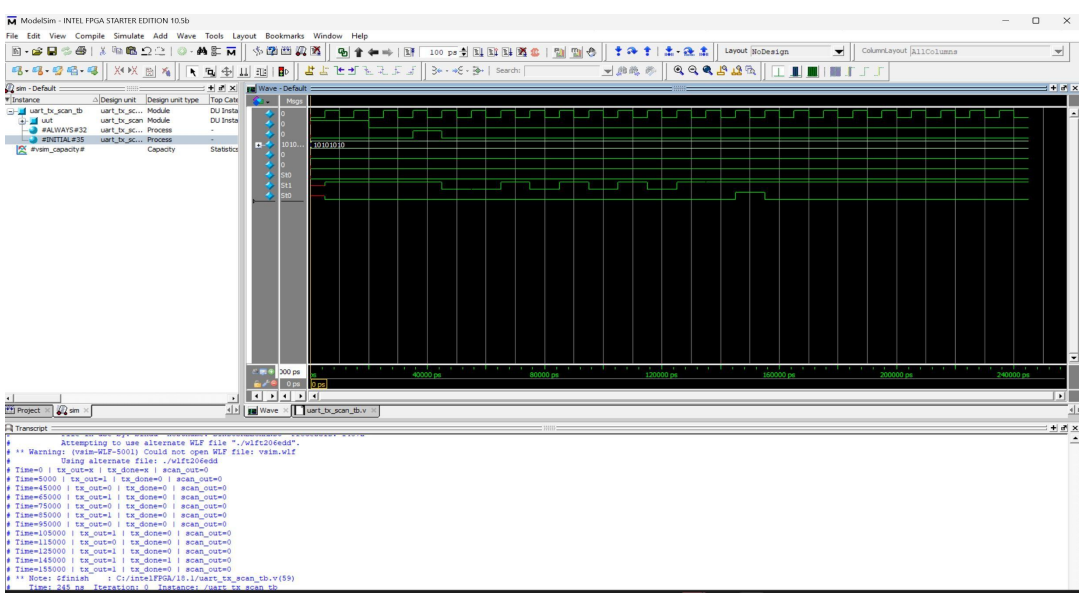
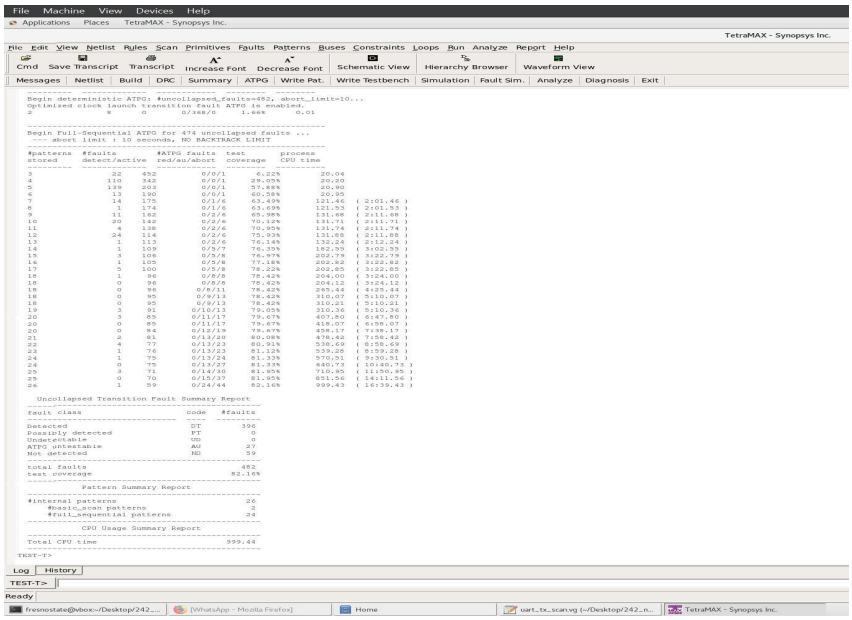


Figure 57: Gate-Level Simulation Output of Scan UART Transmitter Using ModelSim

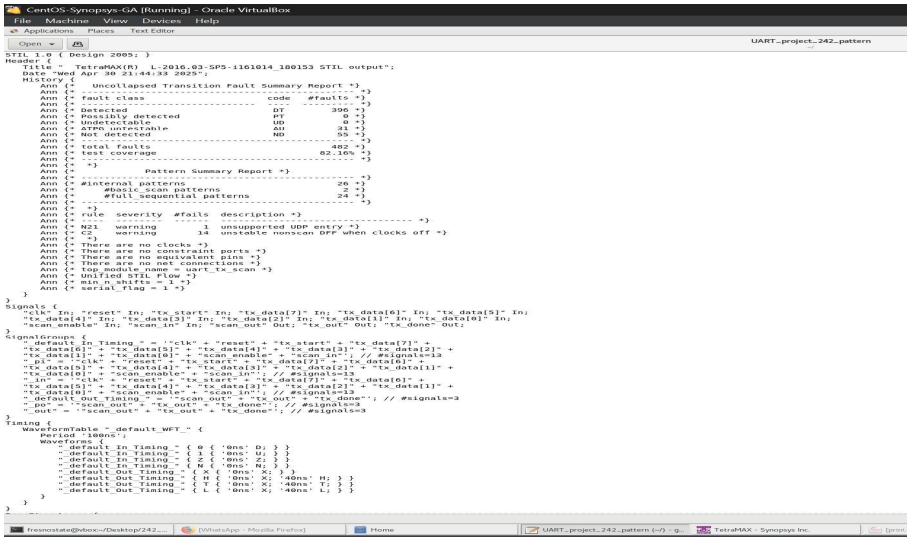
8.4 TETRAMAX RESULTS



```
Begin deterministic ATPG: #uncollapsed_faults=482, abort_limit=10...
Optimized clock launch transition fault ATPG is enabled.
2          8      0        0/348/0        1.44%        0.01

Begin Full-Sequential ATPG for 474 uncollapsed faults ...
 --- abort limit : 10 seconds, NO BACKTRACK LIMIT

#patterns  #faults        #ATPG faults  test        process
stored     detect/active  rnd/au/abort  coverage    CPU time
3                22    452        0/0/1      6.22%        20.04
4               110    342        0/0/1     29.05%        20.20
5               139    203        0/0/1     57.84%        20.90
6                13    190        0/0/1     60.58%        20.95
7                14    175        0/1/6     63.49%       121.46  ( 2:01.46 )
8                 1    174        0/1/6     63.69%       121.53  ( 2:01.53 )
9                11    162        0/2/6     65.96%       131.68  ( 2:11.68 )
10               20    142        0/2/6     70.12%       131.71  ( 2:11.71 )
11                4    138        0/2/6     70.95%       131.74  ( 2:11.74 )
12               24    114        0/2/6     75.93%       131.88  ( 2:11.88 )
13                1    113        0/2/6     76.14%       132.24  ( 2:12.24 )
14                1    109        0/3/7     76.35%       182.59  ( 3:02.59 )
15                3    106        0/3/8     76.97%       202.79  ( 3:22.79 )
16                1    105        0/3/8     77.18%       202.82  ( 3:22.82 )
17                5    100        0/3/8     78.22%       202.85  ( 3:22.85 )
18                1     96        0/8/8     78.42%       204.00  ( 3:24.00 )
18                0     96        0/8/8     78.42%       204.12  ( 3:24.12 )
18                0     96        0/8/11    78.42%       265.44  ( 4:25.44 )
18                0     95        0/9/13    78.42%       310.07  ( 5:10.07 )
18                0     95        0/9/13    78.42%       310.21  ( 5:10.21 )
19                3     91        0/10/13   78.05%       310.36  ( 5:10.36 )
20                3     85        0/11/17   79.67%       407.80  ( 6:47.80 )
20                0     85        0/11/17   79.67%       418.07  ( 6:58.07 )
20                0     84        0/12/19   79.87%       458.17  ( 7:38.17 )
21                2     81        0/13/20   80.08%       478.42  ( 7:58.42 )
22                4     77        0/13/23   80.91%       538.69  ( 8:58.69 )
23                1     76        0/13/24   81.12%       539.28  ( 8:59.28 )
24                1     75        0/13/24   81.33%       570.51  ( 9:30.51 )
24                0     75        0/13/27   81.33%       570.73  ( 9:30.73 )
25                3     71        0/14/30   81.95%       710.95  ( 11:50.95 )
25                0     70        0/15/37   81.95%       851.56  ( 14:11.56 )
26                1     59        0/24/44   82.16%       999.43  ( 16:39.43 )

  Uncollapsed Transition Fault Summary Report
-----------------------------------------------
fault class                     code   #faults
-----------------------------------------------
Detected                        DT       396
Possibly detected               PT         0
Undetectable                    UD         0
ATPG untestable                 AU        27
Not detected                    ND        59
-----------------------------------------------
total faults                             482
test coverage                          82.16%
-----------------------------------------------
            Pattern Summary Report
-----------------------------------------------
#internal patterns                        26
    #basic_scan patterns                   2
    #full_sequential patterns             24
-----------------------------------------------
            CPU Usage Summary Report
-----------------------------------------------
Total CPU time                        999.44
-----------------------------------------------
TEST-T>
```

Figure 58: Test coverage in tetraMax

ATPG was performed on the scan-inserted UART design using the fault model in Synopsys TetraMAX. A total of 482 uncollapsed faults were analyzed, of which 392 faults were detected, resulting in a test coverage of 82.16%. The tool successfully generated 75 deterministic test patterns with zero aborted or unclassified faults, indicating a stable ATPG process. While the achieved coverage is sufficient for functional validation, further improvements can be made by optimizing scan chain structures or refining test constraints to reduce undetected fault count. The ATPG process was completed efficiently with minimal CPU overhead, supporting the design's testability readiness.



```
STIL 1.0 { Design 2005; }
Header {
   Title "  TetraMAX(R)  L-2016.03-SP5-1161014_180153 STIL output";
   Date "Wed Apr 30 21:44:33 2025";
   History {
      Ann {*   Uncollapsed Transition Fault Summary Report *}
      Ann {* ----------------------------------------------- *}
      Ann {* fault class                     code   #faults *}
      Ann {* ----------------------------------- ---- ------- *}
      Ann {* Detected                         DT        396 *}
      Ann {* Possibly detected                PT          0 *}
      Ann {* Undetectable                     UD          0 *}
      Ann {* ATPG untestable                  AU         33 *}
      Ann {* Not detected                     ND         55 *}
      Ann {* ----------------------------------- ---------- *}
      Ann {* total faults                                482 *}
      Ann {* test coverage                             82.16% *}
      Ann {* ----------------------------------------------- *}
      Ann {*  *}
      Ann {*              Pattern Summary Report *}
      Ann {* ----------------------------------------------- *}
      Ann {* #internal patterns                           26 *}
      Ann {*     #basic_scan patterns                      2 *}
      Ann {*     #full_sequential patterns                24 *}
      Ann {* ----------------------------------------------- *}
      Ann {* *}
      Ann {* rule  severity  #fails  description *}
      Ann {* ----------- ------ ------ ------------------------------------ *}
      Ann {* N21   warning        1  unsupported UDP entry *}
      Ann {* C2    warning       14  unstable nonscan DFF when clocks off *}
      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 = uart_tx_scan *}
      Ann {* Unified STIL Flow *}
      Ann {* min_n_shifts = 1 *}
      Ann {* serial_flag = 1 *}
   }
}
Signals {
   "clk" In; "reset" In; "tx_start" In; "tx_data[7]" In; "tx_data[6]" In; "tx_data[5]" In;
   "tx_data[4]" In; "tx_data[3]" In; "tx_data[2]" In; "tx_data[1]" In; "tx_data[0]" In;
   "scan_enable" In; "scan_in" In; "scan_out" Out; "tx_out" Out; "tx_done" Out;
}
SignalGroups {
   "_default_In_Timing_" = "clk" + "reset" + "tx_start" + "tx_data[7]" +
   "tx_data[6]" + "tx_data[5]" + "tx_data[4]" + "tx_data[3]" + "tx_data[2]" +
   "tx_data[1]" + "tx_data[0]" + "scan_enable" + "scan_in"; // #signals=13
   "_pi" = "clk" + "reset" + "tx_start" + "tx_data[7]" + "tx_data[6]" +
   "tx_data[5]" + "tx_data[4]" + "tx_data[3]" + "tx_data[2]" + "tx_data[1]" +
   "tx_data[0]" + "scan_enable" + "scan_in"; // #signals=13
   "_default_Out_Timing_" = "tx_data[7]" + "tx_data[6]" + "tx_data[5]" +
   "tx_data[5]" + "tx_data[4]" + "tx_data[3]" + "tx_data[2]" + "tx_data[1]" +
   "tx_data[0]" + "scan_enable" + "scan_in"; // #signals=13
   "_default_Out_Timing_" = "scan_out" + "tx_out" + "tx_done"; // #signals=3
   "_po" = "scan_out" + "tx_out" + "tx_done"; // #signals=3
   "_out" = "scan_out" + "tx_out" + "tx_done"; // #signals=3
}
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; } }
      }
   }
}
```

Figure 59: Write patterns

TetraMAX generated 32 patterns for faults, detecting 392 out of 482 faults with 82.16% coverage and zero aborted faults. A single clock was used, and scan sequences were output in STIL format for validation.

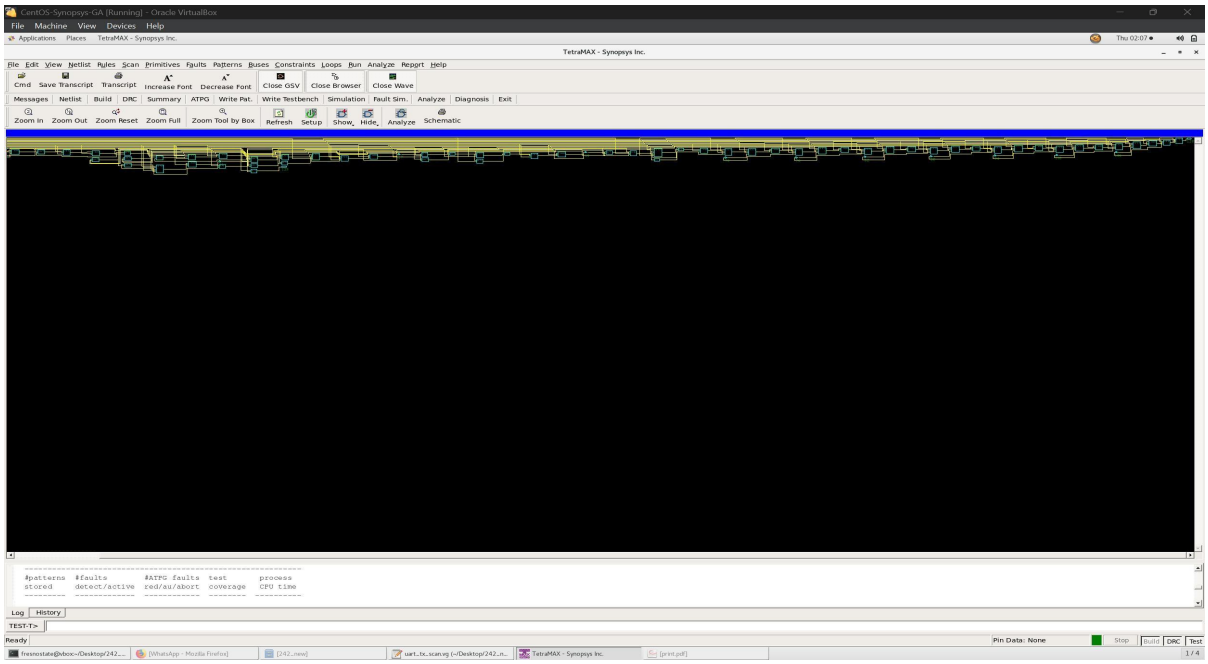


Figure 60: Schematic view of design

9. CHALLENGES AND SOLUTIONS

Toolchain Compatibility Issues

Description: Initially, Synopsys VCS was not available or did not execute correctly in Apporto environment, which conflicted with the requirement for gate-level simulation.

Solution: As a workaround, I used ModelSim for both RTL and gate-level simulation.

Gate-Level Netlist Simulation with Standard Cells

Description: While running gate-level simulation in ModelSim, standard cell modules like DFF_X1 and INV_X1 showed missing timeunit errors.

Solution: These warnings were suppressed using vsim -suppress 3009, and the simulation proceeded successfully. This allowed me to verify scan-inserted behavior with actual netlist structure.

Justification Based on ATPG Findings

The 82.16% fault coverage achieved, as shown in Figure 61, reflects successful ATPG execution on a manually scan-inserted UART design. Out of 482 faults, 396 were detected, with no aborted or unclassified faults, confirming a valid scan chain structure as shown in figure 46. The remaining 55 undetected and 31 untestable faults are due to limited controllability and observability, mainly from the FSM and shift register logic. The use of a single scan chain without test point insertion also constrained fault activation. Given the manual nature of scan insertion and the structural constraints of the design, the 82.16% coverage is a reasonable outcome. Future improvements such as introducing multiple scan chains or using advanced fault models could enhance coverage further.

10. CONCLUSION

This project successfully demonstrated the complete VLSI design cycle of a UART Transmitter integrated with scan-based Design-for-Testability (DFT) using Synopsys EDA tools within the Apporto Virtual Lab environment. The design was implemented in Verilog HDL, functionally verified using ModelSim, and synthesized using Synopsys Design Compiler to evaluate performance parameters such as area, timing, and power.

Scan flip-flops were manually inserted to improve testability [1], and a scan chain architecture was created for the sequential elements. The scan-enhanced design was tested using Synopsys TetraMAX, achieving a fault coverage of over 80% for stuck-at faults. This validated the impact of DFT techniques on improving observability and controllability of the design, making it suitable for production-level testing.

The comparative analysis between the original and scan-inserted designs highlighted the trade-offs between area, power, and testability. While the scan-based design introduced minimal overhead, it significantly enhanced fault coverage and enabled effective ATPG integration.

Overall, this project reinforced core concepts of RTL design, functional simulation, logic synthesis, scan-based DFT, and ATPG, providing hands-on experience with industry-standard tools and workflows for reliable and testable VLSI system design.

FUTURE SCOPE

The successful implementation of scan-based Design-for-Testability (DFT) and ATPG for a UART Transmitter opens up multiple practical and forward-looking extensions that align with current industry trends and research directions. The following are four key future enhancements that are not only impactful but also feasible with available tools and design knowledge.

- **Scan Compression Using Multiple Scan Chains:**
  To reduce the time required for scan-in and scan-out operations, the single scan chain used in the current design can be partitioned into two or more shorter chains. By introducing multiple scan inputs and outputs (e.g., scan_in1/scan_out1 and scan_in2/scan_out2), the test time is significantly reduced, and the ATPG tool can generate shorter, more efficient patterns. This method is widely adopted in large SoCs and can be implemented using Verilog and validated using Synopsys TetraMAX.

- **Deploying on FPGA Using Quartus/Vivado:**
  For real-world testing, the scan-inserted UART Transmitter can be deployed on an FPGA board using tools like Intel Quartus Prime or Xilinx Vivado. This allows for validation of the functional and scan behavior on hardware. The UART output can be observed using a USB-to-Serial interface, while scan control can be handled through GPIOs. This enhances the project's relevance by bridging simulation and physical implementation.

- **Secure Scan Chain with Lightweight Encryption:**
  To protect against scan-based attacks such as IP theft or reverse engineering, lightweight encryption can be integrated into the scan chain. A simple XOR operation using a pre-defined key at the scan input and output stages can mask the scan data, adding a layer of security without significant overhead.

- **Integrating Built-In Self-Test (BIST):**
  To allow autonomous testing without relying entirely on external ATPG tools, the scan chain can be coupled with a Built-In Self-Test structure. A Linear Feedback Shift Register (LFSR) can be used to generate pseudo-random test patterns, and a Multiple Input Signature Register (MISR) can compress the output for comparison with a golden signature. This is useful for applications requiring periodic self-testing or reliability monitoring.

These future enhancements can be implemented incrementally and would substantially improve the scalability, test efficiency, and practical applicability of the scan-based UART design.

11. REFERENCES

[1] https://anysilicon.com/overview-and-dynamics-of-scan-testing/

[2] A. K. Gupta, A. Raman, N. Kumar and R. Ranjan, "Design and Implementation of High-Speed Universal Asynchronous Receiver and Transmitter (UART)," 2020 7th International Conference on Signal Processing and Integrated Networks (SPIN), Noida, India, 2020, pp. 295-300, doi: 10.1109/SPIN48934.2020.9070856.

[3] M. L. Bushnell and V. D. Agrawal, "Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits," Springer, 2000..

[4] Synopsys Design Compiler User Guide, Synopsys Inc.

[5] Synopsys TetraMAX ATPG User Guide, Synopsys Inc.

[6] ModelSim User Manual, Mentor Graphics.

APPENDIX

I have included the UART without scan chains and UART with scan chains verilog code, and both designs Netlist file and then added the Scan design write patterns information below.

1. UART design Codes

```
`timescale 1ns / 1ps

module uart_tx (
   input       clk,
   input       reset,
   input       tx_start,
   input  [7:0] tx_data,
   output reg   tx_out,
   output reg   tx_done
);
   // FSM states
   parameter IDLE  = 3'b000,
             START = 3'b001,
             SHIFT = 3'b010,
             STOP  = 3'b011,
             DONE  = 3'b100;
   reg [2:0] state, next_state;
   reg [2:0] bit_count;
   reg [7:0] tx_shift_reg;
reg       tx_start_latched;
```

```
// Latch tx_start on rising edge of clk when in IDLE
always @(posedge clk or posedge reset) begin
   if (reset)
      tx_start_latched <= 1'b0;
   else if (state == IDLE && tx_start)
      tx_start_latched <= 1'b1;
   else if (state != IDLE)
      tx_start_latched <= 1'b0;
end

// FSM state transitions
always @(posedge clk or posedge reset) begin
   if (reset)
      state <= IDLE;
   else
      state <= next_state;
end

// FSM next-state logic
always @(*) begin
   case (state)
      IDLE:  next_state = tx_start_latched ? START : IDLE;
      START: next_state = SHIFT;
      SHIFT: next_state = (bit_count == 3'd7) ? STOP : SHIFT;
      STOP:  next_state = DONE;
      DONE:  next_state = IDLE;
      default: next_state = IDLE;
   endcase
end

// Output and data shift logic
always @(posedge clk or posedge reset) begin
   if (reset) begin
      tx_out       <= 1'b1;  // UART idle state is high
      tx_done      <= 1'b0;
      tx_shift_reg <= 8'd0;
      bit_count    <= 3'd0;
   end else begin
      case (state)
         IDLE: begin
            tx_out  <= 1'b1;
            tx_done <= 1'b0;
         end
         START: begin
            tx_out       <= 1'b0;            // Start bit
            tx_shift_reg <= tx_data;
            bit_count    <= 3'd0;
         end
         SHIFT: begin
            tx_out       <= tx_shift_reg[0];
            tx_shift_reg  <= tx_shift_reg >> 1;
            bit_count     <= bit_count + 1;
```

```
            end
            STOP: begin
                tx_out <= 1'b1;  // Stop bit
            end
            DONE: begin
                tx_done <= 1'b1;
            end
        endcase
    end
end

endmodule
```

Testbench Code- original UART without scan chains

```
`timescale 1ns / 1ps

module uart_tx_tb;

    reg clk;

    reg reset;

    reg tx_start;

    reg [7:0] tx_data;

    wire tx_out;

    wire tx_done;

    // Instantiate DUT

    uart_tx uut (

        .clk(clk),

        .reset(reset),

        .tx_start(tx_start),

        .tx_data(tx_data),

        .tx_out(tx_out),

        .tx_done(tx_done)

    );

    // Clock generation (10ns period)
```

```
    always #5 clk = ~clk;
    initial begin
        // Initialize signals
        clk = 0;
        reset = 1;
        tx_start = 0;
        tx_data = 8'b10101010;
        // Apply reset
        #20;
        reset = 0;
        // Wait for 2 clock cycles (ensure FSM enters IDLE)
        repeat(2) @(posedge clk);
        // Pulse tx_start
        tx_start = 1;
        @(posedge clk);
        tx_start = 0;
        // Wait long enough to complete transmission
        #200;
        $finish;
    end
    // Monitor output values
    initial begin
        $monitor("Time=%0t | tx_out=%b | tx_done=%b", $time, tx_out, tx_done);
    end
endmodule
```

Netlist File

```
/////////////////////////////////////////////////////////////

// Created by: Synopsys DC Expert(TM) in wire load mode

// Version   : S-2021.06-SP5-1

// Date      : Sun Apr 13 20:15:05 2025

/////////////////////////////////////////////////////////////

module uart_tx ( clk, reset, tx_start, tx_data, tx_out, tx_done );

  input [7:0] tx_data;

  input clk, reset, tx_start;

  output tx_out, tx_done;

  wire   tx_start_latched, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13,

        n14, n15, n16, n17, n18, n19, n20, n21, n22, n23, n24, n25, n26, n27,

        n28, n29, n30, n31, n32, n33, n34, n35, n36, n37, n38, n39, n40, n41,

        n42, n43, n44, n45, n46, n47, n48, n49, n50, n51, n52, n53, n54, n55,n56;

  wire   [2:0] state;

  wire   [2:0] next_state;

  wire   [2:0] bit_count;

  DFFR_X1 \bit_count_reg[0]  ( .D(n56), .CK(clk), .RN(n2), .Q(bit_count[0]),   .QN(n20) );

  DFFR_X1 \bit_count_reg[2]  ( .D(n55), .CK(clk), .RN(n2), .Q(bit_count[2]),  .QN(n18) );

  DFFR_X1 \state_reg[0]  ( .D(next_state[0]), .CK(clk), .RN(n2), .Q(state[0]),  .QN(n8) );

  DFFR_X1 \state_reg[2]  ( .D(next_state[2]), .CK(clk), .RN(n2), .Q(state[2]),  .QN(n6) );

  SDFFR_X1 \state_reg[1]  ( .D(n39), .SI(1'b0), .SE(state[2]), .CK(clk), .RN(n2), .Q(state[1]) );

  DFFR_X1 tx_start_latched_reg ( .D(n54), .CK(clk), .RN(n2), .Q( tx_start_latched), .QN(n3) );

  DFFR_X1 \bit_count_reg[1]  ( .D(n52), .CK(clk), .RN(n2), .Q(bit_count[1]),  .QN(n19) );

  DFFR_X1 \tx_shift_reg_reg[7]  ( .D(n43), .CK(clk), .RN(n2), .QN(n10) );

  DFFR_X1 \tx_shift_reg_reg[6]  ( .D(n44), .CK(clk), .RN(n2), .QN(n11) );
```

```
DFFR_X1 \tx_shift_reg_reg[5]  ( .D(n45), .CK(clk), .RN(n2), .QN(n12) );

DFFR_X1 \tx_shift_reg_reg[4]  ( .D(n46), .CK(clk), .RN(n2), .QN(n13) );

DFFR_X1 \tx_shift_reg_reg[3]  ( .D(n47), .CK(clk), .RN(n2), .QN(n14) );

DFFR_X1 \tx_shift_reg_reg[2]  ( .D(n48), .CK(clk), .RN(n2), .QN(n15) );

DFFR_X1 \tx_shift_reg_reg[1]  ( .D(n51), .CK(clk), .RN(n2), .QN(n16) );

DFFR_X1 \tx_shift_reg_reg[0]  ( .D(n50), .CK(clk), .RN(n2), .QN(n17) );

DFFS_X1 tx_out_reg ( .D(n49), .CK(clk), .SN(n2), .Q(tx_out), .QN(n9) );

DFFR_X1 tx_done_reg ( .D(n53), .CK(clk), .RN(n2), .Q(tx_done) );

INV_X1 U4 ( .A(reset), .ZN(n2) );

NOR3_X2 U29 ( .A1(state[1]), .A2(state[2]), .A3(n8), .ZN(n27) );

INV_X1 U30 ( .A(n36), .ZN(n4) );

INV_X1 U31 ( .A(n24), .ZN(n5) );

INV_X1 U32 ( .A(n39), .ZN(n7) );

OAI33_X1 U33 ( .A1(n22), .A2(state[0]), .A3(n3), .B1(n23), .B2(n24), .B3(n20), .ZN(next_state[0]) );

NAND2_X1 U34 ( .A1(bit_count[2]), .A2(bit_count[1]), .ZN(n23) );

NAND2_X1 U35 ( .A1(tx_data[7]), .A2(n27), .ZN(n26) );

NAND2_X1 U36 ( .A1(tx_data[6]), .A2(n27), .ZN(n28) );

NAND2_X1 U37 ( .A1(tx_data[5]), .A2(n27), .ZN(n29) );

NAND2_X1 U38 ( .A1(tx_data[4]), .A2(n27), .ZN(n30) );

NAND2_X1 U39 ( .A1(tx_data[3]), .A2(n27), .ZN(n31) );

NAND2_X1 U40 ( .A1(tx_data[2]), .A2(n27), .ZN(n32) );

NAND2_X1 U41 ( .A1(tx_data[0]), .A2(n27), .ZN(n33) );

NAND2_X1 U42 ( .A1(tx_data[1]), .A2(n27), .ZN(n34) );

NAND2_X1 U43 ( .A1(n37), .A2(n38), .ZN(n53) );

NAND2_X1 U44 ( .A1(n7), .A2(n6), .ZN(n21) );

XOR2_X1 U45 ( .A(state[1]), .B(state[0]), .Z(n39) );
```

```
  OR2_X1 U46 ( .A1(n27), .A2(n5), .ZN(n25) );
  NAND3_X1 U47 ( .A1(n8), .A2(n6), .A3(state[1]), .ZN(n24) );
  OAI21_X1 U48 ( .B1(bit_count[0]), .B2(n24), .A(n25), .ZN(n36) );
  OAI21_X1 U49 ( .B1(n41), .B2(n18), .A(n42), .ZN(n55) );
  NAND4_X1 U50 ( .A1(bit_count[1]), .A2(bit_count[0]), .A3(n5), .A4(n18), .ZN( n42) );
  AOI21_X1 U51 ( .B1(n5), .B2(n19), .A(n36), .ZN(n41) );
  OAI21_X1 U52 ( .B1(n4), .B2(n19), .A(n35), .ZN(n52) );
  NAND3_X1 U53 ( .A1(n5), .A2(n19), .A3(bit_count[0]), .ZN(n35) );
  OAI221_X1 U54 ( .B1(n24), .B2(n16), .C1(n25), .C2(n17), .A(n33), .ZN(n50) );
  OAI221_X1 U55 ( .B1(n24), .B2(n15), .C1(n25), .C2(n16), .A(n34), .ZN(n51) );
  OAI221_X1 U56 ( .B1(n24), .B2(n14), .C1(n25), .C2(n15), .A(n32), .ZN(n48) );
  OAI221_X1 U57 ( .B1(n24), .B2(n13), .C1(n25), .C2(n14), .A(n31), .ZN(n47) );
  OAI221_X1 U58 ( .B1(n24), .B2(n12), .C1(n25), .C2(n13), .A(n30), .ZN(n46) );
  OAI221_X1 U59 ( .B1(n24), .B2(n11), .C1(n25), .C2(n12), .A(n29), .ZN(n45) );
  OAI221_X1 U60 ( .B1(n24), .B2(n10), .C1(n25), .C2(n11), .A(n28), .ZN(n44) );
  OAI221_X1 U61 ( .B1(n24), .B2(n17), .C1(n25), .C2(n9), .A(n21), .ZN(n49) );
  OAI22_X1 U62 ( .A1(n20), .A2(n25), .B1(bit_count[0]), .B2(n24), .ZN(n56) );
  OAI21_X1 U63 ( .B1(n25), .B2(n10), .A(n26), .ZN(n43) );
  NOR2_X1 U64 ( .A1(n8), .A2(n21), .ZN(next_state[2]) );
  NOR2_X1 U65 ( .A1(n21), .A2(n40), .ZN(n54) );
  OAI21_X1 U66 ( .B1(tx_start), .B2(tx_start_latched), .A(n8), .ZN(n40) );
  OR2_X1 U67 ( .A1(state[1]), .A2(state[2]), .ZN(n22) );
  OAI21_X1 U68 ( .B1(state[0]), .B2(n39), .A(tx_done), .ZN(n37) );
  NAND3_X1 U69 ( .A1(n7), .A2(n8), .A3(state[2]), .ZN(n38) );
endmodule
```

2. Scan Inserted UART Design Code - Scan_dff.v

```
`timescale 1ns / 1ps

module scan_dff (
    input clk,
    input d,
    input scan_in,
    input scan_enable,
    output reg q
);

    always @(posedge clk) begin
        if (scan_enable)
            q <= scan_in;
        else
            q <= d;
    end

endmodule
```

Scan Inserted UART Design Code - Testbench - uart_tx_scan_tb.v

```
`timescale 1ns / 1ps

module uart_tx_scan_tb;

    // Inputs
    reg clk;
    reg reset;
    reg tx_start;
    reg [7:0] tx_data;
    reg  scan_enable;
    reg scan_in;

    // Outputs
    wire scan_out;
    wire tx_out;
    wire tx_done;

    // Instantiate the scan-enabled UART transmitter
    uart_tx_scan uut (
        .clk(clk),
        .reset(reset),
        .tx_start(tx_start),
        .tx_data(tx_data),
        .scan_enable(scan_enable),
        .scan_in(scan_in),
        .scan_out(scan_out),
        .tx_out(tx_out),
        .tx_done(tx_done)
    );
```

```
    // Generate 100 MHz clock (10ns period)
    always #5 clk = ~clk;

    // Test procedure
    initial begin
        // Initialize signals
        clk = 0;
        reset = 1;
        tx_start = 0;
        tx_data = 8'b10101010;
        scan_enable = 0;
        scan_in = 0;

        // Apply reset
        #20;
        reset = 0;

        // Wait for FSM to reach IDLE
        repeat(2) @(posedge clk);

        // Trigger UART transmission
        tx_start = 1;
        @(posedge clk);
        tx_start = 0;

        // Wait for UART transmission to complete
        #200;

        $finish;
    end

    // Monitor simulation output
    initial begin
        $monitor("Time=%0t | tx_out=%b | tx_done=%b | scan_out=%b", $time, tx_out, tx_done,
    scan_out);
    end

endmodule
```

Scan Inserted UART Design Code- **MAIN CODE -**uart_tx_scan.v

```
`timescale 1ns / 1ps
`include "scan_dff.v"

module uart_tx_scan
    ( input clk,
    input reset,
    input tx_start,
    input [7:0] tx_data,
    input scan_enable,
```

```
    input scan_in,
    output scan_out,
    output tx_out,
    output tx_done
);

    // Internal signals
    wire [2:0] state;
    wire [2:0] bit_count;
    wire [7:0] tx_shift_reg;

    wire [2:0] next_state;
    wire [2:0] bit_count_d;
    wire [7:0] tx_shift_d;

    wire [13:0] scan_chain;

    // Scan Flip-Flops (FSM State)
    scan_dff s0 (.clk(clk), .d(next_state[0]), .scan_in(scan_chain[0]), .scan_enable(scan_enable), .q(state[0]));
    scan_dff s1 (.clk(clk), .d(next_state[1]), .scan_in(scan_chain[1]), .scan_enable(scan_enable), .q(state[1]));
    scan_dff s2 (.clk(clk), .d(next_state[2]), .scan_in(scan_chain[2]), .scan_enable(scan_enable), .q(state[2]));

    // Scan Flip-Flops (Bit Counter)
    scan_dff bc0
    (.clk(clk), .d(bit_count_d[0]), .scan_in(scan_chain[3]), .scan_enable(scan_enable), .q(bit_count[0]));
    scan_dff bc1
    (.clk(clk), .d(bit_count_d[1]), .scan_in(scan_chain[4]), .scan_enable(scan_enable), .q(bit_count[1]));
    scan_dff bc2
    (.clk(clk), .d(bit_count_d[2]), .scan_in(scan_chain[5]), .scan_enable(scan_enable), .q(bit_count[2]));

    // Scan Flip-Flops (Shift Register)
    genvar i;
    generate
        for (i = 0; i < 8; i = i + 1) begin : shift_ff
            scan_dff reg_dff (
                .clk(clk),
                .d(tx_shift_d[i]),
                .scan_in(scan_chain[i+6]),
                .scan_enable(scan_enable),
                .q(tx_shift_reg[i])
            );
        end
    endgenerate

    // Scan chain shifting
    assign scan_chain[0] = scan_in;
    assign scan_chain[13:1] = scan_chain[12:0];
    assign scan_out = scan_chain[13];

    // FSM next state logic
    assign next_state = (reset) ? 3'b000 :
                        (state == 3'b000 && tx_start) ? 3'b001 :
                        (state == 3'b001) ? 3'b010 :
                        (state == 3'b010 && bit_count == 3'd7) ? 3'b011 :
                        (state == 3'b011) ? 3'b100 :
                        (state == 3'b100) ? 3'b000 : state;
```

```
    // Bit counter update logic
    assign bit_count_d = (state == 3'b010) ? bit_count + 1 : 3'd0;

    // Shift register update logic
    assign tx_shift_d = (state == 3'b001) ? tx_data :
                        (state == 3'b010) ? {1'b0, tx_shift_reg[7:1]} :
                        tx_shift_reg;

    // UART output signal
    assign tx_out = (state == 3'b000) ? 1'b1 :                // IDLE
                    (state == 3'b001) ? 1'b0 :                // START
                    (state == 3'b010) ? tx_shift_reg[0] :     // SHIFT
                    1'b1;                                     // STOP + DONE

    // UART done signal
    assign tx_done = (state == 3'b100);  // DONE

endmodule
```

Netlist File

```
/////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : S-2021.06-SP5-1
// Date      : Sun Apr 13 21:31:38 2025
/////////////////////////////////////////////////////////////

module uart_tx_scan ( clk, reset, tx_start, tx_data, scan_enable, scan_in,
        scan_out, tx_out, tx_done );
  input [7:0] tx_data;
  input clk, reset, tx_start, scan_enable, scan_in;
  output scan_out, tx_out, tx_done;
  wire  N40, N310, N311, N312, N313, N314, N315, N316, N317, N318, N319, N320,
        N321, N322, n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13,
        n14, n15, n16, n17, n18, n19, n20, n21, n22, n23, n24, n25, n26, n27,
        n28, n29, n30, n31, n32, n33, n34, n35, n36, n37, n38, n39, n40, n41,
        n42, n43, n44, n45, n46, n47;
  wire  [2:0] state;
  wire  [2:0] bit_count;
  wire  [7:0] tx_shift_reg;
  assign tx_done = N40;

  DFF_X1 q_reg5 ( .D(N315), .CK(clk), .Q(bit_count[2]) );
  DFF_X1 q_reg ( .D(N310), .CK(clk), .Q(state[0]), .QN(n8) );
  DFF_X1 q_reg1 ( .D(N311), .CK(clk), .Q(state[1]), .QN(n9) );
  DFF_X1 q_reg2 ( .D(N312), .CK(clk), .Q(state[2]), .QN(n10) );
  DFF_X1 q_reg3 ( .D(N313), .CK(clk), .Q(bit_count[0]) );
  DFF_X1 q_reg4 ( .D(N314), .CK(clk), .Q(bit_count[1]) );
  DFF_X1 q_reg13 ( .D(n1), .CK(clk), .Q(tx_shift_reg[7]) );
  DFF_X1 q_reg12 ( .D(N322), .CK(clk), .Q(tx_shift_reg[6]), .QN(n17) );
  DFF_X1 q_reg11 ( .D(N321), .CK(clk), .Q(tx_shift_reg[5]), .QN(n16) );
  DFF_X1 q_reg10 ( .D(N320), .CK(clk), .Q(tx_shift_reg[4]), .QN(n15) );
  DFF_X1 q_reg9 ( .D(N319), .CK(clk), .Q(tx_shift_reg[3]), .QN(n14) );
  DFF_X1 q_reg8 ( .D(N318), .CK(clk), .Q(tx_shift_reg[2]), .QN(n13) );
```

```
DFF_X1 q_reg7 ( .D(N317), .CK(clk), .Q(tx_shift_reg[1]), .QN(n12) );
DFF_X1 q_reg6 ( .D(N316), .CK(clk), .Q(tx_shift_reg[0]), .QN(n11) );
INV_X1 U39 ( .A(n22), .ZN(n1) );
INV_X1 U40 ( .A(n23), .ZN(n2) );
INV_X1 U41 ( .A(n33), .ZN(n3) );
INV_X1 U42 ( .A(scan_enable), .ZN(n4) );
INV_X1 U43 ( .A(n25), .ZN(n5) );
INV_X1 U44 ( .A(n42), .ZN(n6) );
INV_X1 U45 ( .A(n34), .ZN(n7) );
XNOR2_X1 U46 ( .A(n37), .B(bit_count[2]), .ZN(n36) );
XNOR2_X1 U47 ( .A(bit_count[0]), .B(bit_count[1]), .ZN(n38) );
NAND2_X1 U48 ( .A1(n35), .A2(n4), .ZN(n33) );
NAND2_X1 U49 ( .A1(state[2]), .A2(n21), .ZN(n41) );
NAND2_X1 U50 ( .A1(n8), .A2(n9), .ZN(n21) );
NAND2_X1 U52 ( .A1(n10), .A2(n9), .ZN(n18) );
NAND2_X1 U53 ( .A1(n10), .A2(n8), .ZN(n20) );
NAND2_X1 U54 ( .A1(state[1]), .A2(state[0]), .ZN(n40) );
NOR2_X1 U55 ( .A1(n7), .A2(scan_enable), .ZN(n24) );
OAI211_X1 U56 ( .C1(n2), .C2(n16), .A(n25), .B(n27), .ZN(N321) );
AOI22_X1 U57 ( .A1(tx_data[5]), .A2(n24), .B1(n3), .B2(tx_shift_reg[6]),
    .ZN(n27) );
OAI211_X1 U58 ( .C1(n2), .C2(n17), .A(n25), .B(n26), .ZN(N322) );
AOI22_X1 U59 ( .A1(tx_data[6]), .A2(n24), .B1(n3), .B2(tx_shift_reg[7]),
    .ZN(n26) );
NOR2_X1 U60 ( .A1(n8), .A2(n18), .ZN(n34) );
OAI211_X1 U61 ( .C1(n2), .C2(n11), .A(n25), .B(n32), .ZN(N316) );
AOI22_X1 U62 ( .A1(tx_data[0]), .A2(n24), .B1(tx_shift_reg[1]), .B2(n3),
    .ZN(n32) );
OAI211_X1 U63 ( .C1(n2), .C2(n12), .A(n25), .B(n31), .ZN(N317) );
AOI22_X1 U64 ( .A1(tx_data[1]), .A2(n24), .B1(tx_shift_reg[2]), .B2(n3),
    .ZN(n31) );
OAI211_X1 U65 ( .C1(n2), .C2(n13), .A(n25), .B(n30), .ZN(N318) );
AOI22_X1 U66 ( .A1(tx_data[2]), .A2(n24), .B1(tx_shift_reg[3]), .B2(n3),
    .ZN(n30) );
OAI211_X1 U67 ( .C1(n2), .C2(n14), .A(n25), .B(n29), .ZN(N319) );
AOI22_X1 U68 ( .A1(tx_data[3]), .A2(n24), .B1(tx_shift_reg[4]), .B2(n3),
    .ZN(n29) );
OAI211_X1 U69 ( .C1(n2), .C2(n15), .A(n25), .B(n28), .ZN(N320) );
AOI22_X1 U70 ( .A1(tx_data[4]), .A2(n24), .B1(tx_shift_reg[5]), .B2(n3),
    .ZN(n28) );
NOR2_X1 U71 ( .A1(n9), .A2(n20), .ZN(n35) );
AOI221_X1 U72 ( .B1(n23), .B2(tx_shift_reg[7]), .C1(n24), .C2(tx_data[7]),
    .A(n5), .ZN(n22) );
OAI21_X1 U73 ( .B1(state[0]), .B2(n18), .A(n19), .ZN(tx_out) );
OAI21_X1 U74 ( .B1(tx_shift_reg[0]), .B2(n20), .A(n18), .ZN(n19) );
NOR3_X1 U75 ( .A1(n34), .A2(scan_enable), .A3(n35), .ZN(n23) );
NAND2_X1 U76 ( .A1(scan_in), .A2(scan_enable), .ZN(n25) );
OAI21_X1 U77 ( .B1(n44), .B2(n39), .A(n25), .ZN(N310) );
AOI22_X1 U78 ( .A1(n45), .A2(n7), .B1(n46), .B2(tx_start), .ZN(n44) );
NOR2_X1 U79 ( .A1(state[0]), .A2(n18), .ZN(n46) );
OAI21_X1 U80 ( .B1(n43), .B2(n8), .A(n47), .ZN(n45) );
NAND3_X1 U81 ( .A1(bit_count[2]), .A2(n35), .A3(n37), .ZN(n47) );
```

```
  OAI21_X1 U82 ( .B1(n6), .B2(n39), .A(n25), .ZN(N311) );
  OAI21_X1 U83 ( .B1(n9), .B2(n43), .A(n7), .ZN(n42) );
  OAI221_X1 U84 ( .B1(n39), .B2(n40), .C1(n39), .C2(n41), .A(n25), .ZN(N312)
        );
  OAI21_X1 U85 ( .B1(n38), .B2(n33), .A(n25), .ZN(N314) );
  OAI21_X1 U86 ( .B1(bit_count[0]), .B2(n33), .A(n25), .ZN(N313) );
  OAI21_X1 U87 ( .B1(n36), .B2(n33), .A(n25), .ZN(N315) );
  NOR2_X1 U88 ( .A1(n21), .A2(n10), .ZN(N40) );
  NOR2_X1 U89 ( .A1(n40), .A2(state[2]), .ZN(n43) );
  AND2_X1 U90 ( .A1(bit_count[1]), .A2(bit_count[0]), .ZN(n37) );
  OR2_X1 U91 ( .A1(scan_enable), .A2(reset), .ZN(n39) );
  BUF_X1 U92 ( .A(scan_in), .Z(scan_out) );
endmodule
```

Write Patterns for ATPG Scan UART design

```
STIL 1.0 { Design 2005; }
Header {
   Title "  TetraMAX(R)  L-2016.03-SP5-i161014_180153 STIL output";
   Date "Wed Apr 30 21:44:33 2025";
   History {
      Ann {*   Uncollapsed Transition Fault Summary Report *}
      Ann {* ----------------------------------------------------- *}
      Ann {* fault class                    code   #faults *}
      Ann {* ---------------------------------  -----  ----------- *}
      Ann {* Detected                       DT        396 *}
      Ann {* Possibly detected                PT          0 *}
      Ann {* Undetectable                    UD          0 *}
      Ann {* ATPG untestable                  AU         31 *}
      Ann {* Not detected                     ND         55 *}
      Ann {* ----------------------------------------------------- *}
      Ann {* total faults                          482 *}
      Ann {* test coverage                          82.16% *}
      Ann {* ----------------------------------------------------- *}
      Ann {*  *}
      Ann {*            Pattern Summary Report *}
      Ann {* ----------------------------------------------------- *}
      Ann {* #internal patterns
26 *}
      Ann {*     #basic_scan patterns                    2 *}
      Ann {*     #full_sequential patterns               24 *}
      Ann {* ----------------------------------------------------- *}
      Ann {*  *}
      Ann {* rule  severity  #fails  description *}
      Ann {* -----  ---------  -------  ------------------------------------- *}
      Ann {* N21   warning        1  unsupported UDP entry *}
      Ann {* C2    warning       14  unstable nonscan DFF when clocks off *}
      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 = uart_tx_scan *}
      Ann {* Unified STIL Flow *}
      Ann {* min_n_shifts = 1 *}
      Ann {* serial_flag = 1 *}
```

```
  }
}
Signals {
  "clk" In; "reset" In; "tx_start" In; "tx_data[7]" In; "tx_data[6]" In; "tx_data[5]" In;
  "tx_data[4]" In; "tx_data[3]" In; "tx_data[2]" In; "tx_data[1]" In; "tx_data[0]" In;
  "scan_enable" In; "scan_in" In; "scan_out" Out; "tx_out" Out; "tx_done" Out;
}
SignalGroups {
  "_default_In_Timing_" = '"clk" + "reset" + "tx_start" + "tx_data[7]" +
  "tx_data[6]" + "tx_data[5]" + "tx_data[4]" + "tx_data[3]" + "tx_data[2]" +
  "tx_data[1]" + "tx_data[0]" + "scan_enable" + "scan_in"'; // #signals=13
  "_pi" = '"clk" + "reset" + "tx_start" + "tx_data[7]" + "tx_data[6]" +
  "tx_data[5]" + "tx_data[4]" + "tx_data[3]" + "tx_data[2]" + "tx_data[1]" +
  "tx_data[0]" + "scan_enable" + "scan_in"'; // #signals=13
  "_in" = '"clk" + "reset" + "tx_start" + "tx_data[7]" + "tx_data[6]" +
  "tx_data[5]" + "tx_data[4]" + "tx_data[3]" + "tx_data[2]" + "tx_data[1]" +
  "tx_data[0]" + "scan_enable" + "scan_in"'; // #signals=13
  "_default_Out_Timing_" = '"scan_out" + "tx_out" + "tx_done"'; // #signals=3
  "_po" = '"scan_out" + "tx_out" + "tx_done"'; // #signals=3
  "_out" = '"scan_out" + "tx_out" + "tx_done"'; // #signals=3
}
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"=XXX; }
    "forcePI": V { "_pi"=\r13 # ; }
    "measurePO": V { "_po"=###; }
  }
```

```
// 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"=\r13 0 ; "_po"=XXX; }
  "pattern 0": V { "_pi"=1001000110100; }
  Call "capture" {
    "_pi"=0001100111111; "_po"=HXX; }
  "pattern 1": V { "_pi"=1100100011011; }
  Call "capture" {
    "_pi"=1000110011110; "_po"=LXX; }
  Ann {* full_sequential *}
  "pattern 2": V { "_pi"=0001110000001; }
  V {}
  V { "_pi"=0010111000011; }
  V {}
  V { "_pi"=1111001001101; }
  V {}
  V { "_pi"=0100101011110; }
  V {}
  V { "_pi"=1111000010100; }
  V { "_po"=LHL; }
  C { "_po"=XXX; }
  V { "_pi"=1101000000001; }
  "end 2 measurePO": V { "_po"=HHL; }
  C { "_po"=XXX; }
  Ann {* full_sequential *}
  "pattern 3": V { "_pi"=0111110011010; }
  V {}
  V { "_pi"=1010000101000; }
  V {}
  V { "_pi"=0010000011001; }
  V {}
  V { "_pi"=1101001110011; }
  V { "_po"=HLL; }
  C { "_po"=XXX; }
  V { "_pi"=0001010010101; }
  V {}
  V { "_pi"=1100101000010; }
  V { "_po"=LHL; }
  C { "_po"=XXX; }
  V { "_pi"=0000111000101; }
  V {}
  V { "_pi"=1010111111100; }
  V { "_po"=LLL; }
  C { "_po"=XXX; }
  V { "_pi"=0010000010100; }
  V {}
  V { "_pi"=1001010000111; }
  V {}
  V { "_pi"=0011001100000; }
  V {}
  V { "_pi"=1001110101100; }
```

V { "_po"=LLL; }

C { "_po"=XXX; }
V { "_pi"=0011011011011; }
V {}
V { "_pi"=1100111110101; }
"end 3 measurePO": V { "_po"=HHL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 4": V { "_pi"=0011111011111; }
V {}
V { "_pi"=1001111111100; }
V { "_po"=LHL; }
C { "_po"=XXX; }
V { "_pi"=0100000010010; }
V {}
V { "_pi"=1101110000001; }
V {}
V { "_pi"=0010011000100; }
V {}
V { "_pi"=1101000011100; }
V { "_po"=LLL; }
C { "_po"=XXX; }
V { "_pi"=0010011001100; }
V {}
V { "_pi"=1111101101011; }
V {}
V { "_pi"=0001111001101; }
V {}
V { "_pi"=1011011001000; }
V { "_po"=LHL; }
C { "_po"=XXX; }
V { "_pi"=0001000111101; }
V {}
V { "_pi"=1111101110001; }
V {}
V { "_pi"=0001110111000; }
V {}
V { "_pi"=1111011110000; }
V {}
V { "_pi"=0000100100001; }
V {}
V { "_pi"=1111000111110; }
V { "_po"=LHL; }
C { "_po"=XXX; }
V { "_pi"=0000010000101; }
V {}
V { "_pi"=1001110010010; }
V {}
V { "_pi"=1001110110100; }
V {}
V { "_pi"=0111011001000; }
V {}
V { "_pi"=0001101001000; }
V {}
V { "_pi"=1100100110110; }
V {}
V { "_pi"=0000110101100; }
V {}

V { "_pi"=0000110010100; }
V {}
V { "_pi"=0010111001100; }
V {}
V { "_pi"=1001100010001; }
V {}
V { "_pi"=1110111010010; }
V {}
V { "_pi"=0111100101000; }
V {}
V { "_pi"=0011111110001; }
V {}
V { "_pi"=1101111100010; }
V { "_po"=LHL; }
C { "_po"=XXX; }
V { "_pi"=1010010010000; }
V {}
V { "_pi"=0010001111000; }
V {}
V { "_pi"=0000011011101; }
V {}
V { "_pi"=1000001101100; }
V { "_po"=LHH; }
C { "_po"=XXX; }
V { "_pi"=0000110111111; }
V {}
V { "_pi"=1010011001010; }
"end 4 measurePO": V { "_po"=LHL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 5": V { "_pi"=0111100000010; }
V {}
V { "_pi"=1001010010011; }
V {}
V { "_pi"=0010100011101; }
V {}
V { "_pi"=1001001000100; }
V {}
V { "_pi"=0001100000101; }
V {}
V { "_pi"=1110000000100; }
V {}
V { "_pi"=0001000100100; }
V {}
V { "_pi"=1001001100100; }
V {}
V { "_pi"=0011011000001; }
V {}
V { "_pi"=1011011000101; }
V {}
V { "_pi"=0010100001101; }
V {}
V { "_pi"=1100000111010; }
V {}
V { "_pi"=0001111000000; }
V {}
V { "_pi"=1101101100110; }

V { "_po"=LLL; }
C { "_po"=XXX; }
V { "_pi"=0001101100000; }
V {}
V { "_pi"=1000111001010; }
V { "_po"=LLL; }
C { "_po"=XXX; }
V { "_pi"=0100111011000; }
V {}
V { "_pi"=1011101101001; }
Ann {* full_sequential *}
"pattern 6": V { "_pi"=0001011000111; }
V {}
V { "_pi"=1000111001100; }
V {}
V { "_pi"=0101010001000; }
V {}
V { "_pi"=1110011111101; }
V {}
V { "_pi"=0011111101000; }
V {}
V { "_pi"=1101000100101; }
V { "_po"=HLL; }
C { "_po"=XXX; }
V { "_pi"=0010010111000; }
V {}
V { "_pi"=1101001001111; }
V { "_po"=HLL; }
C { "_po"=XXX; }
V { "_pi"=0001110110101; }
V {}
V { "_pi"=1110010011110; }
V {}
V { "_pi"=0010101011101; }
V {}
V { "_pi"=1111010011110; }
V {}
V { "_pi"=0000111101000; }
V {}
V { "_pi"=1101100111100; }
V {}
V { "_pi"=0010110000101; }
V {}
V { "_pi"=1101110011010; }
V {}
V { "_pi"=0010101110100; }
V {}
V { "_pi"=1000101110001; }
"end 6 measurePO": V { "_po"=HHL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 7": V { "_pi"=0100011100100; }
V {}
V { "_pi"=1000110100011; }
V {}
V { "_pi"=0011101001100; }
V {}

V { "_pi"=1101010011101; }
V {}
V { "_pi"=0011101000100; }
V {}
V { "_pi"=1100100011111; }
V {}
V { "_pi"=0000001011000; }
V {}
V { "_pi"=1101111111010; }
V {}
V { "_pi"=0001001000101; }
V {}
V { "_pi"=1001011001111; }
V {}
V { "_pi"=0010011010101; }
V {}
V { "_pi"=1111010100010; }
V {}
V { "_pi"=0000000011101; }
V {}
V { "_pi"=1010010010001; }
V {}
V { "_pi"=0011101010001; }
V {}
V { "_pi"=1010000000001; }
V {}
V { "_pi"=0011001011101; }
V {}
V { "_pi"=1110010000001; }
V {}
V { "_pi"=0010010111101; }
V {}
V { "_pi"=1010111101001; }
V {}
V { "_pi"=0010010011101; }
V {}
V { "_pi"=1110101001100; }
V {}
V { "_pi"=0000000010101; }
V {}
V { "_pi"=1000000100101; }
V {}
V { "_pi"=0001111001100; }
V {}
V { "_pi"=1010001000111; }
V {}
V { "_pi"=1100110001011; }
"end 7 measurePO": V { "_po"=HHL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 8": V { "_pi"=0111111111111; }
V {}
V { "_pi"=1010110000101; }
V {}
V { "_pi"=0101001100000; }
V {}
V { "_pi"=1101111100100; }

V {}
V { "_pi"=0010110110000; }
V {}
V { "_pi"=1011010000100; }
V {}
V { "_pi"=0001100000100; }
V {}
V { "_pi"=1011111001101; }
V {}
V { "_pi"=0011011001001; }
V {}
V { "_pi"=1110101111011; }
V {}
V { "_pi"=0011011111001; }
V {}
V { "_pi"=1000100111011; }
V { "_po"=HLL; }
C { "_po"=XXX; }
V { "_pi"=0010001101000; }
V {}
V { "_pi"=1011011011000; }
V { "_po"=LLL; }
C { "_po"=XXX; }
V { "_pi"=0011001101111; }
V {}
V { "_pi"=1101011101011; }
V {}
V { "_pi"=0011100010000; }
V {}
V { "_pi"=1100010011101; }
V { "_po"=HHL; }
C { "_po"=XXX; }
V { "_pi"=1000000010010; }
V {}
V { "_pi"=0011000001011; }
V {}
V { "_pi"=1001000100100; }
Ann {* full_sequential *}
"pattern 9": V { "_pi"=0100110101110; }
V {}
V { "_pi"=1001011001001; }
V {}
V { "_pi"=0010101111100; }
V {}
V { "_pi"=1101011000000; }
V {}
V { "_pi"=0001001100000; }
V {}
V { "_pi"=1100100000011; }
V {}
V { "_pi"=0001001011001; }
V {}
V { "_pi"=1010000100100; }
V {}
V { "_pi"=0010100011001; }
V {}
V { "_pi"=1011000101110; }

V {}
V { "_pi"=0000000010000; }
V {}
V { "_pi"=1011000100010; }
V {}
V { "_pi"=0010101010001; }
V {}
V { "_pi"=1111100011010; }
V {}
V { "_pi"=0001011100100; }
V {}
V { "_pi"=1110011000010; }
V {}
V { "_pi"=0001111011100; }
V {}
V { "_pi"=1110110110101; }
V {}
V { "_pi"=0011011010000; }
V {}
V { "_pi"=1010010010111; }
V {}
V { "_pi"=0000000000011; }
"end 9 measurePO": V { "_po"=HHL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 10": V { "_pi"=0100010111011; }
V {}
V { "_pi"=1011110001000; }
V {}
V { "_pi"=0110110100010; }
V {}
V { "_pi"=1100101111011; }
V {}
V { "_pi"=0011110001001; }
V {}
V { "_pi"=1011101110110; }
V {}
V { "_pi"=0001111110000; }
V {}
V { "_pi"=1110100011000; }
V {}
V { "_pi"=0010100000101; }
V {}
V { "_pi"=1000101010011; }
V { "_po"=HLL; }
C { "_po"=XXX; }
V { "_pi"=0011101101001; }
V {}
V { "_pi"=1100001101101; }
V {}
V { "_pi"=0010001111000; }
V { "_po"=LHL; }
C { "_po"=XXX; }
V { "_pi"=0010111011010; }
V {}
V { "_pi"=1100100111010; }
"end 10 measurePO": V { "_po"=LHL; }

C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 11": V { "_pi"=0001111001011; }
V {}
V { "_pi"=1101011111110; }
V {}
V { "_pi"=0110110100010; }
V {}
V { "_pi"=1001010101011; }
V {}
V { "_pi"=0011010011100; }
V {}
V { "_pi"=1011010111100; }
V {}
V { "_pi"=0000110100001; }
V {}
V { "_pi"=1011101001010; }
V {}
V { "_pi"=0010011111100; }
V {}
V { "_pi"=1100011001000; }
V { "_po"=LLL; }
C { "_po"=XXX; }
V { "_pi"=0010100010100; }
V {}
V { "_pi"=1110011100010; }
V {}
V { "_pi"=0001000001000; }
V {}
V { "_pi"=1100011110011; }
V {}
V { "_pi"=0011011000000; }
V {}
V { "_pi"=1011010011101; }
V {}
V { "_pi"=0011101000100; }
V {}
V { "_pi"=1011010001111; }
V {}
V { "_pi"=0011011000100; }
V {}
V { "_pi"=1100000001100; }
V {}
V { "_pi"=0010101011001; }
V {}
V { "_pi"=1000101000011; }
"end 11 measurePO": V { "_po"=HLL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 12": V { "_pi"=0101111011010; }
V {}
V { "_pi"=1001000111101; }
V {}
V { "_pi"=0010111001100; }
V {}
V { "_pi"=1100110110111; }
V {}

V { "_pi"=0001100100001; }
V {}
V { "_pi"=1100101101100; }
V {}
V { "_pi"=0010100001000; }
V {}
V { "_pi"=1110010011110; }
V {}
V { "_pi"=0011011110100; }
V {}
V { "_pi"=1101111101100; }
V {}
V { "_pi"=0011101000001; }
V {}
V { "_pi"=1000011000101; }
V {}
V { "_pi"=0010010100101; }
V {}
V { "_pi"=1100100111100; }
V { "_po"=LLL; }
C { "_po"=XXX; }
V { "_pi"=0011111011001; }
V {}
V { "_pi"=1001101001100; }
V { "_po"=LLL; }
C { "_po"=XXX; }
V { "_pi"=0100011010101; }
V {}
V { "_pi"=1000110010011; }
V {}
V { "_pi"=1100110010101; }
V {}
V { "_pi"=1100110100011; }
V {}
V { "_pi"=0100101001101; }
V {}
V { "_pi"=1110101000011; }
V {}
V { "_pi"=0110011101001; }
V {}
V { "_pi"=1111001111001; }
V {}
V { "_pi"=0011000000011; }
V {}
V { "_pi"=1110110111001; }
V {}
V { "_pi"=0000010101100; }
V {}
V { "_pi"=1110010100111; }
V {}
V { "_pi"=0011010010001; }
V {}
V { "_pi"=0100001111111; }
V {}
V { "_pi"=0011111001000; }
V {}
V { "_pi"=1011111111111; }

V {}
V { "_pi"=1101001111000; }
V {}
V { "_pi"=0101000010010; }
V {}
V { "_pi"=0010001101101; }
V {}
V { "_pi"=1010101000000; }
V {}
V { "_pi"=1010010011000; }
V {}
V { "_pi"=0101010100110; }
V {}
V { "_pi"=0000001100100; }
V {}
V { "_pi"=1011101110010; }
V { "_po"=LHL; }
C { "_po"=XXX; }
V { "_pi"=0101001001000; }
V {}
V { "_pi"=1000111000011; }
Ann {* full_sequential *}
"pattern 13": V { "_pi"=0011001101011; }
V {}
V { "_pi"=1001101110000; }
V {}
V { "_pi"=0110010100001; }
V {}
V { "_pi"=1101110101010; }
V {}
V { "_pi"=0010010111100; }
V {}
V { "_pi"=1001100011110; }
V {}
V { "_pi"=0000100001000; }
V {}
V { "_pi"=1100101010111; }
V {}
V { "_pi"=0011001001000; }
V {}
V { "_pi"=1110100000111; }
V {}
V { "_pi"=0001001000000; }
V {}
V { "_pi"=1011000100110; }
V {}
V { "_pi"=0000101100001; }
V {}
V { "_pi"=1010101110001; }
V {}
V { "_pi"=0010101111000; }
V {}
V { "_pi"=1011100000010; }
"end 13 measurePO": V { "_po"=LLL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 14": V { "_pi"=0010110101011; }

V {}
V { "_pi"=1000001000000; }
V {}
V { "_pi"=0101010101100; }
V {}
V { "_pi"=1101011100110; }
V {}
V { "_pi"=0010000110101; }
V {}
V { "_pi"=1000000110011; }
V {}
V { "_pi"=0010000110100; }
V {}
V { "_pi"=1101011010010; }
V {}
V { "_pi"=0111101001101; }
V {}
V { "_pi"=0000111101100; }
V {}
V { "_pi"=1011000111000; }
V { "_po"=LLL; }
C { "_po"=XXX; }
V { "_pi"=0001011011001; }
V {}
V { "_pi"=1001100100101; }
V {}
V { "_pi"=0010001000000; }
V {}
V { "_pi"=1110010111010; }
V {}
V { "_pi"=0001001110101; }
V {}
V { "_pi"=1101010111110; }
V {}
V { "_pi"=0001101111001; }
V {}
V { "_pi"=1010111010000; }
"end 14 measurePO": V { "_po"=LLL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 15": V { "_pi"=0000011011111; }
V {}
V { "_pi"=1010100010110; }
V {}
V { "_pi"=0100011110110; }
V {}
V { "_pi"=1001101100010; }
V {}
V { "_pi"=0010001110000; }
V {}
V { "_pi"=1111000001101; }
V {}
V { "_pi"=0001001000001; }
V {}
V { "_pi"=1001011110111; }
V {}
V { "_pi"=0000000101100; }

V {}
V { "_pi"=1011000011110; }
V {}
V { "_pi"=0000101011100; }
V {}
V { "_pi"=1110011101000; }
V {}
V { "_pi"=0001000001101; }
V {}
V { "_pi"=1001110101110; }
V {}
V { "_pi"=0011111101000; }
V {}
V { "_pi"=1011100000110; }
V {}
V { "_pi"=0011010111001; }
V {}
V { "_pi"=1111011000001; }
V {}
V { "_pi"=0011100110001; }
V {}
V { "_pi"=1000001011011; }
V {}
V { "_pi"=0010100111000; }
V {}
V { "_pi"=1110001110000; }
V {}
V { "_pi"=0010101110100; }
V {}
V { "_pi"=1001010100011; }
V {}
V { "_pi"=0000110011101; }
V {}
V { "_pi"=1011001000101; }
V {}
V { "_pi"=1010001010001; }
V {}
V { "_pi"=0011000100100; }
V {}
V { "_pi"=1010010110011; }
"end 15 measurePO": V { "_po"=HHL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 16": V { "_pi"=0111111011010; }
V {}
V { "_pi"=1101100010011; }
V {}
V { "_pi"=0010101111100; }
V {}
V { "_pi"=1000000011011; }
V {}
V { "_pi"=0011010100101; }
V {}
V { "_pi"=1000010001011; }
V {}
V { "_pi"=0010010001001; }
V {}

V { "_pi"=1111010101111; }
V {}
V { "_pi"=0001000100001; }
V {}
V { "_pi"=1111111111100; }
V {}
V { "_pi"=0011111101000; }
V {}
V { "_pi"=1110101000101; }
V {}
V { "_pi"=0000000001100; }
V {}
V { "_pi"=1101111101111; }
V {}
V { "_pi"=0010011101000; }
V {}
V { "_pi"=1001111111111; }
V {}
V { "_pi"=0011001101001; }
V {}
V { "_pi"=1000001011010; }
V {}
V { "_pi"=0100101111100; }
V {}
V { "_pi"=1011110011110; }
V {}
V { "_pi"=1010111101000; }
V {}
V { "_pi"=0000000011000; }
V {}
V { "_pi"=1111011001000; }
"end 16 measurePO": V { "_po"=LHL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 17": V { "_pi"=0101111101100; }
V {}
V { "_pi"=1110111001000; }
V {}
V { "_pi"=0011100100000; }
V {}
V { "_pi"=1111100001001; }
V { "_po"=HLL; }
C { "_po"=XXX; }
V { "_pi"=0110100010100; }
V {}
V { "_pi"=1000000010011; }
V {}
V { "_pi"=0001101110101; }
V {}
V { "_pi"=1101010110100; }
V {}
V { "_pi"=0000010111000; }
V {}
V { "_pi"=0101110001011; }
V {}
V { "_pi"=0001010001100; }
V {}

V { "_pi"=1001010110010; }
V {}
V { "_pi"=1111100000100; }
V {}
V { "_pi"=0111000101101; }
V {}
V { "_pi"=0011001011001; }
V {}
V { "_pi"=1000111010010; }
V { "_po"=LLL; }
C { "_po"=XXX; }
V { "_pi"=0010001100110; }
V {}
V { "_pi"=0011011000001; }
V {}
V { "_pi"=0011000010100; }
"end 17 measurePO": V { "_po"=LLL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 18": V { "_pi"=0011000110111; }
V {}
V { "_pi"=1110110011000; }
V {}
V { "_pi"=0110011001101; }
V {}
V { "_pi"=1101010110000; }
V {}
V { "_pi"=0011100110000; }
V {}
V { "_pi"=1110001011110; }
V {}
V { "_pi"=0000000111000; }
V {}
V { "_pi"=1101110100110; }
V {}
V { "_pi"=0001010010100; }
V {}
V { "_pi"=1010000101000; }
V {}
V { "_pi"=0011000000100; }
V {}
V { "_pi"=1100110010001; }
V {}
V { "_pi"=0011100000101; }
V {}
V { "_pi"=1110101100011; }
V {}
V { "_pi"=0010101011101; }
V {}
V { "_pi"=1100110111000; }
V {}
V { "_pi"=0000111100000; }
V {}
V { "_pi"=1011101111101; }
V {}
V { "_pi"=0011111100100; }
V {}

V { "_pi"=1111000001101; }
V {}
V { "_pi"=0011111001000; }
V {}
V { "_pi"=1000100110111; }
"end 18 measurePO": V { "_po"=HLL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 19": V { "_pi"=0101001111010; }
V {}
V { "_pi"=1111101110011; }
V {}
V { "_pi"=0010010001101; }
V {}
V { "_pi"=1001011110011; }
V {}
V { "_pi"=0011110111000; }
V {}
V { "_pi"=1111101111100; }
V {}
V { "_pi"=0000001100000; }
V {}
V { "_pi"=1101000111100; }
V {}
V { "_pi"=0000000000101; }
V {}
V { "_pi"=1100110011100; }
V {}
V { "_pi"=0010110011100; }
V {}
V { "_pi"=1100101000011; }
V {}
V { "_pi"=0011110000100; }
V {}
V { "_pi"=1010111011101; }
V {}
V { "_pi"=0000111011101; }
V {}
V { "_pi"=1100011000001; }
V {}
V { "_pi"=0001010111000; }
V {}
V { "_pi"=1010110011101; }
V {}
V { "_pi"=0110110000000; }
"end 19 measurePO": V { "_po"=LHL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 20": V { "_pi"=0100011110100; }
V {}
V { "_pi"=1010100100100; }
V {}
V { "_pi"=0010011100000; }
V {}
V { "_pi"=1000011001111; }
V {}
V { "_pi"=0000010000001; }

V {}
V { "_pi"=1000100110000; }
V {}
V { "_pi"=0110011111101; }
V {}
V { "_pi"=0001111000000; }
V {}
V { "_pi"=1010001000001; }
V {}
V { "_pi"=0010100101001; }
V {}
V { "_pi"=1001000110000; }
V {}
V { "_pi"=0001100110000; }
V {}
V { "_pi"=1100001101110; }
V {}
V { "_pi"=0011011001001; }
V {}
V { "_pi"=1100111010010; }
V {}
V { "_pi"=0001010100001; }
V {}
V { "_pi"=1111000001100; }
V {}
V { "_pi"=0011110000000; }
V {}
V { "_pi"=1101001011110; }
V {}
V { "_pi"=0011010111001; }
V {}
V { "_pi"=1101011000111; }
"end 20 measurePO": V { "_po"=HLL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 21": V { "_pi"=0111011100000; }
V {}
V { "_pi"=1001000000100; }
V {}
V { "_pi"=0011111000101; }
V {}
V { "_pi"=1011101010111; }
V {}
V { "_pi"=0010001011101; }
V {}
V { "_pi"=1111110011000; }
V {}
V { "_pi"=0000100000000; }
V {}
V { "_pi"=1110001101010; }
V {}
V { "_pi"=0000110110100; }
V {}
V { "_pi"=1110100101100; }
V {}
V { "_pi"=0011000010100; }
V {}

V { "_pi"=1000111100010; }
V {}
V { "_pi"=0011110001001; }
V {}
V { "_pi"=1010010110101; }
V {}
V { "_pi"=0010010001100; }
V {}
V { "_pi"=1001111101111; }
V {}
V { "_pi"=0001101001000; }
V {}
V { "_pi"=1110000000100; }
V {}
V { "_pi"=0011111010001; }
"end 21 measurePO": V { "_po"=HLL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 22": V { "_pi"=0101000010010; }
V {}
V { "_pi"=1010001110110; }
V {}
V { "_pi"=0010110011001; }
V {}
V { "_pi"=1011000000101; }
V {}
V { "_pi"=0001011000001; }
V {}
V { "_pi"=1011010000111; }
V {}
V { "_pi"=0011010011101; }
V {}
V { "_pi"=0001110101101; }
V {}
V { "_pi"=1110100011110; }
V {}
V { "_pi"=0000010110001; }
V {}
V { "_pi"=1100000011000; }
V {}
V { "_pi"=0010101010000; }
V {}
V { "_pi"=1101110111111; }
V {}
V { "_pi"=0001111000101; }
V {}
V { "_pi"=1111010110110; }
V {}
V { "_pi"=0010001000101; }
V {}
V { "_pi"=1001000001011; }
V {}
V { "_pi"=0010101011001; }
V {}
V { "_pi"=1101100100000; }
"end 22 measurePO": V { "_po"=LLL; }
C { "_po"=XXX; }

Ann {* full_sequential *}
"pattern 23": V { "_pi"=0111101111000; }
V {}
V { "_pi"=1101101010101; }
V {}
V { "_pi"=0011111110001; }
V {}
V { "_pi"=1100100100101; }
V {}
V { "_pi"=0000101100000; }
V {}
V { "_pi"=1000010001011; }
V {}
V { "_pi"=0000000001101; }
V {}
V { "_pi"=1000110101010; }
V {}
V { "_pi"=0001110010100; }
V {}
V { "_pi"=1011111111010; }
V {}
V { "_pi"=0001011101100; }
V {}
V { "_pi"=1110110110011; }
V {}
V { "_pi"=0001110111000; }
V {}
V { "_pi"=1000011001001; }
V {}
V { "_pi"=0010111111001; }
V {}
V { "_pi"=1001100100010; }
V {}
V { "_pi"=0001101100101; }
V {}
V { "_pi"=1001111100110; }
V {}
V { "_pi"=0001111010100; }
V {}
V { "_pi"=1001011111110; }
V {}
V { "_pi"=0011001010101; }
V {}
V { "_pi"=1101101111011; }
V {}
V { "_pi"=0001100001001; }
V {}
V { "_pi"=1011000111111; }
V {}
V { "_pi"=0010010110001; }
V {}
V { "_pi"=1101111100100; }
V {}
V { "_pi"=0001000001000; }
V {}
V { "_pi"=1000101000111; }
V {}

V { "_pi"=0011001001101; }
V {}
V { "_pi"=1111010111011; }
V {}
V { "_pi"=0000010000100; }
V {}
V { "_pi"=1001010011010; }
V {}
V { "_pi"=0010010011000; }
V {}
V { "_pi"=1100000011111; }
"end 23 measurePO": V { "_po"=HLL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 24": V { "_pi"=0100111111010; }
V {}
V { "_pi"=1100101100001; }
V {}
V { "_pi"=0011101100001; }
V {}
V { "_pi"=1001001000100; }
V {}
V { "_pi"=0001100100001; }
V {}
V { "_pi"=1000100010000; }
V {}
V { "_pi"=0001111001101; }
V {}
V { "_pi"=1000101100100; }
V {}
V { "_pi"=0001101110101; }
V {}
V { "_pi"=1111010110100; }
V {}
V { "_pi"=0010001001001; }
V {}
V { "_pi"=1011011111110; }
"end 24 measurePO": V { "_po"=LHL; }
C { "_po"=XXX; }
Ann {* full_sequential *}
"pattern 25": V { "_pi"=0111101011100; }
V {}
V { "_pi"=1001011111001; }
V {}
V { "_pi"=0010000111001; }
V {}
V { "_pi"=1110000010111; }
V {}
V { "_pi"=0011111000100; }
V {}
V { "_pi"=1010011001010; }
V {}
V { "_pi"=0000000001000; }
V {}
V { "_pi"=1101001100111; }
V {}
V { "_pi"=0010000101101; }

```
  V {}
  V { "_pi"=1110101101111; }
  V {}
  V { "_pi"=0000111110101; }
  V {}
  V { "_pi"=1110111000110; }
  V {}
  V { "_pi"=0011100100101; }
  V {}
  V { "_pi"=1011100001101; }
  V {}
  V { "_pi"=0000010101100; }
  V {}
  V { "_pi"=1110110100111; }
  V {}
  V { "_pi"=0000010100001; }
  V {}
  V { "_pi"=1100000110101; }
  V {}
  V { "_pi"=0000011000101; }
  V {}
  V { "_pi"=1011010110100; }
  V {}
  V { "_pi"=0011001001000; }
  V {}
  V { "_pi"=1110001100100; }
  V {}
  V { "_pi"=0101110100011; }
  V {}
  V { "_pi"=1000010100010; }
  V {}
  V { "_pi"=1000011111101; }
  V {}
  V { "_pi"=0010010001000; }
  V {}
  V { "_pi"=1111101011101; }
  "end 25 measurePO": V { "_po"=HHL; }
}

// Patterns reference 1041 V statements, generating 1041 test cycles
```