California State University, Fresno

LYLES COLLEGE OF ENGINEERING

Department of Electrical and Computer Engineering

ECE 242 – Digital Systems Testing and Testable Design

PROJECT REPORT

Design, Verification, and Testing of a Parallel Arithmetic Unit

Advisor: Dr. Reza Raeisi

Submitted by:

Saachi Jaiswal [301495728]

May 2024

Table of Contents

1. Abstract

In the field of digital systems, the efficiency of arithmetic operations is crucial for enhancing computational throughput and performance. This project presents the design, verification, and testing of a Parallel Arithmetic Unit (PAU) that performs addition and multiplication operations concurrently. Unlike traditional sequential units like Multiply-Accumulate (MAC) circuits, the PAU operates in a purely combinational logical manner. Each component, such as adders and multipliers, performs independently without the sequential dependency on previous outputs.
The unique design of the PAU permits faster processing times by directly calculating and outputting results, which leads to a significant improvement in performance for applications requiring high-speed arithmetic operations. The project includes the complete development cycle from initial design using Verilog HDL to synthesis with Synopsys Design Compiler and comprehensive verification and testing using TetraMAX.
The PAU represents a significant advancement in the design of digital systems hardware, promising substantial benefits in terms of speed and efficiency for complex computational tasks. This report details the methodologies employed, the challenges overcome, and the technical specifications achieved. It makes a compelling case for the adoption of parallel processing units in modern digital architectures.

2. Introduction

2.1 Background on the Importance of Arithmetic Operations in Digital Systems

Arithmetic operations, primarily addition, and multiplication, are the cornerstone of digital systems. They form the basic computation units that underpin a vast array of technologies, from simple microcontrollers to complex processors in supercomputers. At the heart of every digital operation from data processing and signal conditioning to complex algorithms like encryption and image processing lies a series of fundamental arithmetic calculations.
The performance and efficiency of these operations directly influence the overall speed, power consumption, and capabilities of digital systems. Addition and multiplication are integral to a multitude of key processes. In digital signal processing (DSP), for example, filters rely on these operations to modify signals in real-time. In graphics processing units (GPUs), the rapid calculation of pixel values for rendering images heavily depends on efficient arithmetic operations. Additionally, in scientific computing, where large-scale numerical calculations are routine, the speed of these operations can significantly impact the time it takes to model complex phenomena or process large datasets.
The growing demands for higher computational power, with minimal energy consumption and faster processing times, necessitate the continual development of more efficient arithmetic units. As digital systems evolve, the drive towards lower power consumption and higher performance means that the traditional methods of sequential arithmetic computation no longer suffice. This creates a compelling need for innovations that can parallelize these fundamental operations, thereby accelerating their execution and enhancing the overall efficiency of digital systems.
This project addresses this need by proposing the design of a Parallel Arithmetic Unit (PAU) that

aims to significantly speed up arithmetic computations through parallel processing, thereby aligning with the modern requirements of digital technology infrastructures.

2.2 Motivation for Choosing a Parallel Design

The motivation for adopting a parallel design for this project primarily stems from the need to achieve high computational speed and efficiency by leveraging combinational logic. Combinational parallel designs, unlike their sequential counterparts, process multiple inputs simultaneously without depending on previous states or outputs, thus significantly reducing latency, and enhancing throughput.

This architecture enables real-time processing capabilities, which are critical in applications requiring rapid data analysis and immediate response, such as in digital signal processing and real-time computing systems. By implementing a parallel architecture, the project aims to optimize the use of hardware resources, allowing for a more efficient execution of arithmetic operations, such as addition and multiplication, in a single operational cycle. Furthermore, the parallel approach aligns with the increasing demands for more powerful and faster computing solutions in modern digital systems, providing a scalable framework that can support future enhancements and integration with more complex systems. This design not only improves performance but also contributes to better power management and reduced operational complexity, making it a robust solution for high-speed data processing tasks.

2.2.1 Brief Overview of Existing Solutions

The landscape of digital arithmetic operations encompasses a variety of existing solutions, each tailored to specific computational needs and performance requirements. Traditional arithmetic units, such as Arithmetic Logic Units (ALUs), are foundational in computing, and designed to perform a broad range of operations but are often limited by sequential processing constraints. This sequential nature can introduce significant latency, especially as operation complexity increases.

Graphics Processing Units (GPUs) offer a form of parallel processing power, excelling in tasks that require simultaneous handling of multiple data points, such as graphics rendering and machine learning computations. However, GPUs are generally optimized for floating-point operations and might not always deliver the best performance for integer-based arithmetic tasks, which are crucial in many embedded systems and low-power applications.

Application-Specific Integrated Circuits (ASICs) provide tailored solutions by embedding specific arithmetic operations directly into hardware. While offering high performance and efficiency, ASICs lack flexibility; once fabricated, they cannot be reprogrammed or adjusted to new requirements, making them less adaptable to evolving technological needs.

Field-Programmable Gate Arrays (FPGAs) represent a more flexible approach, allowing post-manufacturing programming to fit specific needs. FPGAs can be configured to perform parallel arithmetic operations, providing a balance between the high performance of ASICs and the flexibility of programmable devices. However, they often involve more complex design processes and higher unit costs compared to standard microprocessors.

Each of these solutions has its strengths and limitations, which inform the motivation to explore new designs, such as the parallel combinational approach proposed in this project. This approach aims to harness the speed benefits of parallel processing while eliminating the latency and efficiency issues often encountered in traditional sequential and some parallel configurations. By focusing on combinational logic, the project bypasses the need for data storage elements used in sequential operations, thus reducing complexity, and enhancing speed, which is particularly beneficial for high-performance computing environments.

3 Accessing Tools and Software

3.1 Accessing Apporto Virtual Labs

1. To access the Apporto Virtual Labs:
2. Open your Canvas platform and navigate to the ECE242 course.
3. In the course interface, locate and click on the Apporto Virtual Labs icon.
4. Within Apporto, you will find various software applications listed. To use ModelSim or Synopsys Design Compiler, simply launch the desired software from this list.

3.2 How to Download and Install ModelSim

If you prefer to install ModelSim on your computer:

1. Go to the FPGA Software Download Center on Intel's website by following this link: FPGA Software Download Center.
2. Look for ModelSim - Intel FPGA 10.5b (part of Quartus Prime 16.1) and download it.
3. Ensure you select the correct version for your operating system (Linux, Windows, Mac, etc.) before downloading.
4. Follow the provided instructions to complete the installation.

3.3 How to Download and Install Synopsys Design Compiler

VPN Access
Before initiating the download of Synopsys Design Compiler, ensuring a secure network connection through a VPN is crucial. Follow these steps to set up VPN access:
VPN Installation:
Install a VPN client on your computer to access university or organization-specific resources securely.
It is crucial to have a VPN to avoid connection errors and access restrictions that might occur with Synopsys Design Compiler.
Configuring the VPN:
Search for "Fresno State VPN" or visit Knowledge Base - VPN - Global Protect for a direct link. Follow the step-by-step guide provided for your specific operating system to configure the VPN correctly.

Installation Steps for Synopsys Design Compiler

After configuring your VPN connection, proceed with the installation of Synopsys Design Compiler:

1. Prepare Your System:
    - Download and install Oracle VM VirtualBox from VirtualBox Downloads.
    - Download the VirtualBox "Extensions Pack" from the same location to enable additional functionalities.
2. Setting Up VirtualBox:
    - Install VirtualBox on your system.
    - Install the Extensions Pack either directly through VirtualBox or by double-clicking the Extensions Pack file after VirtualBox installation.
3. Download and Set Up the CentOS Virtual Machine:
    - Download the CentOS VM from Google Drive. Ensure you are logged in with your Fresno State credentials to access this file.
    - Import the downloaded VM appliance into VirtualBox using the default settings. Be patient as this import process may take some time depending on your system's capabilities.
4. Launching the VM:
    - Once the VM is successfully imported, start the virtual machine. For optimal performance, close other programs unless you have a high-performance desktop as running a VM can be resource-intensive.
5. Synopsys Design Compiler Configuration and Usage:
    - Make sure to compile your design in ModelSim before attempting synthesis in Design Compiler, as this can identify potential Verilog issues that Design Compiler might not catch.
    - Pay attention to messages about the inability to read the link_library file or undefined current design, which might indicate setup or configuration issues.
    - Check your .synopsys_dc.setup files for correctness. These files should be present in your working directory or home directory.
6. Additional Resources and Setup Files:
    - Download the DCFilesAndExamples.zip, which contains essential scripts and example files for setup and testing. You can find this package on your course Canvas site.
    - Unpack the files into your working directory, which should include:
    - Makefile: Contains commands for simulation and synthesis.
    - dc-template.tcl: A template for creating a customized command file for Design Compiler.
    - .synopsys_dc.setup: Crucial for setting up the environment. Ensure it starts with a period and is placed correctly as per the instructions.

7. Libraries and Further Configurations:
    - The project uses the 45 nm NanGate FreePDK45 Open Cell Library, which you can access at NanGate.
    - This library includes a variety of functions and standard cells necessary for digital design and synthesis.

4 Project Objectives and Scope

Objective

The primary objective of this project is to design, implement, and verify a parallel arithmetic unit using combinational logic, focusing solely on enhancing computational efficiency and throughput without the overhead of sequential processing. This objective will be achieved through a series of methodically planned steps involving Verilog for design and simulation, Synopsys Design Compiler for synthesis, and TetraMax for test pattern generation and fault analysis. Here's a detailed breakdown of each component of the project objective:

1. **Verilog Design and Simulation**

Objective: Develop a parallel arithmetic unit using Verilog, a hardware description language known for its robustness and flexibility in digital circuit design.
Steps:
Design: Utilize Verilog to create individual modules for the parallel arithmetic operations, focusing on combinational logic to ensure real-time processing capabilities without reliance on previous states.
Simulation: Use ModelSim to simulate the designed Verilog modules, verifying their functionality individually and in combination. This step ensures that all components operate as expected before moving to the synthesis phase.

2. **Synthesis with Synopsys Design Compiler**

Objective: Convert the Verilog HDL code into a gate-level netlist, and obtain area, speed, and power consumption using the Synopsys Design Compiler, a leading synthesis tool in the industry.
Steps:
Setup: Configure the Synopsys Design Compiler environment, including setting up file paths, defining design constraints, and selecting appropriate libraries.
RTL to Gates: Synthesize the RTL code from Verilog to a gate-level netlist, ensuring the design meets the set performance criteria without exceeding resource allocations.

3. **Post-Synthesis Verification**

Objective: Conduct post-synthesis verification to confirm that the gate-level netlist performs as intended under the targeted technology constraints.
Steps:

Simulate Post-Synthesis Netlist: Use the post-synthesis netlist to simulate the design again, ensuring the functionality holds after synthesis.
Analyze Waveforms and Timing: Check waveforms and timing to ensure they meet the design specifications and no new errors have been introduced during synthesis.

4. Test Pattern Generation using TetraMax

Objective: Validate the synthesized design's testability and uncover any potential manufacturing defects using TetraMax, which offers robust Automatic Test Pattern Generation (ATPG) capabilities.
Steps:
Prepare Test Environment: Set up the test environment in TetraMax by importing the synthesized netlist and corresponding technology library files.
Generate Test Patterns: Utilize TetraMax to create test patterns that target potential fault sites within the design, specifically focusing on stuck-at fault models to assess the circuit's response to faults.
Fault Coverage Analysis: Analyze the fault coverage report generated by TetraMax to ensure that a sufficient percentage of possible faults can be detected by the test patterns. Adjustments may be necessary if the initial test patterns do not achieve the desired coverage levels.

Scope of the Project:

The scope of this project encompasses the conceptualization, design, simulation, implementation, and testing of a parallel arithmetic unit designed around combinational logic. The project aims to prove the feasibility of using parallel combinational logic to enhance the efficiency of arithmetic operations typically used in digital signal processing, encryption, scientific computations, and other high-speed computing environments.
The project will explore innovative circuit design techniques and utilize modern digital design tools through the VLSI lifecycle, including HDL programming for design in Verilog, gate-level synthesis using Synopsys Design Compiler, and rigorous testing and verification via TetraMax. This approach aims to achieve a balance between high computational speed and power efficiency while maintaining flexibility for integration into various technological applications.
By accomplishing these objectives, the project will not only demonstrate a technical solution to the limitations of existing arithmetic processing units but also pave the way for future advancements in digital circuit design for arithmetic operations.

5. Methodology

5.1 Design of the Parallel Adder and Multiplier

The design process for the Parallel Adder and Multiplier involved developing separate modules for each arithmetic operation and then integrating them into a unified Parallel Arithmetic Unit (PAU). Below is a detailed overview of the specifications for each unit and their integration.

5.1.1 Specifications for Each Unit

Multiplier Module:

- The Multiplier module is designed to take two 8-bit inputs a and b, and produce a 16-bit output product.
- The operation within the module is straightforward: on every change of inputs a or b, the product is computed as a * b.

Verilog Code:

```
module Multiplier (
input wire [7:0] a,
input wire [7:0] b,
output reg [15:0] product
);
always @* begin
product = a * b;
end
endmodule
```

Adder Module:

- The Adder module handles the addition of a 16-bit number and an 8-bit number, outputting a 16-bit sum.
- This module accepts operand1 (16-bit) and operand2 (8-bit), adding them whenever there is a change in either operand.

Verilog Code:

```
module Adder (
input wire [15:0] operand1,
input wire [7:0] operand2,
output reg [15:0] sum
);
always @* begin
sum = {1'b0, operand1} + {1'b0,operand2};
end
endmodule
```

5.1.2 Integration of the Units into a Single Module

PAU (Parallel Arithmetic Unit) Module:

- The PAU module integrates the Multiplier and Adder modules to perform both operations in a single unit.
- It accepts three 8-bit inputs: a, b, and c. The inputs a and b are fed into the Multiplier, and the resulting product is then added to c using the Adder.
- The final result, a 16-bit number, is stored in result, which is the output of the PAU.

Verilog Code:

```
module p_unit (
  input [7:0] a,
  input [7:0] b,
  input [7:0] c,
  output reg [15:0] result
);
wire [15:0] product;
wire [15:0] sum;

Multiplier mult_inst(.a(a), .b(b), .product(product));
Adder adder_inst(.operand1(product), .operand2(c), .sum(sum));

always @* begin
  result = sum;
end
endmodule
```

Testbench for PAU Module:

- The testbench PAU_unit_tb is designed to validate the functionality of the PAU module.
- It simulates various combinations of inputs to ensure that both the addition and multiplication operations are being performed correctly and the integration works seamlessly.

Verilog Code:

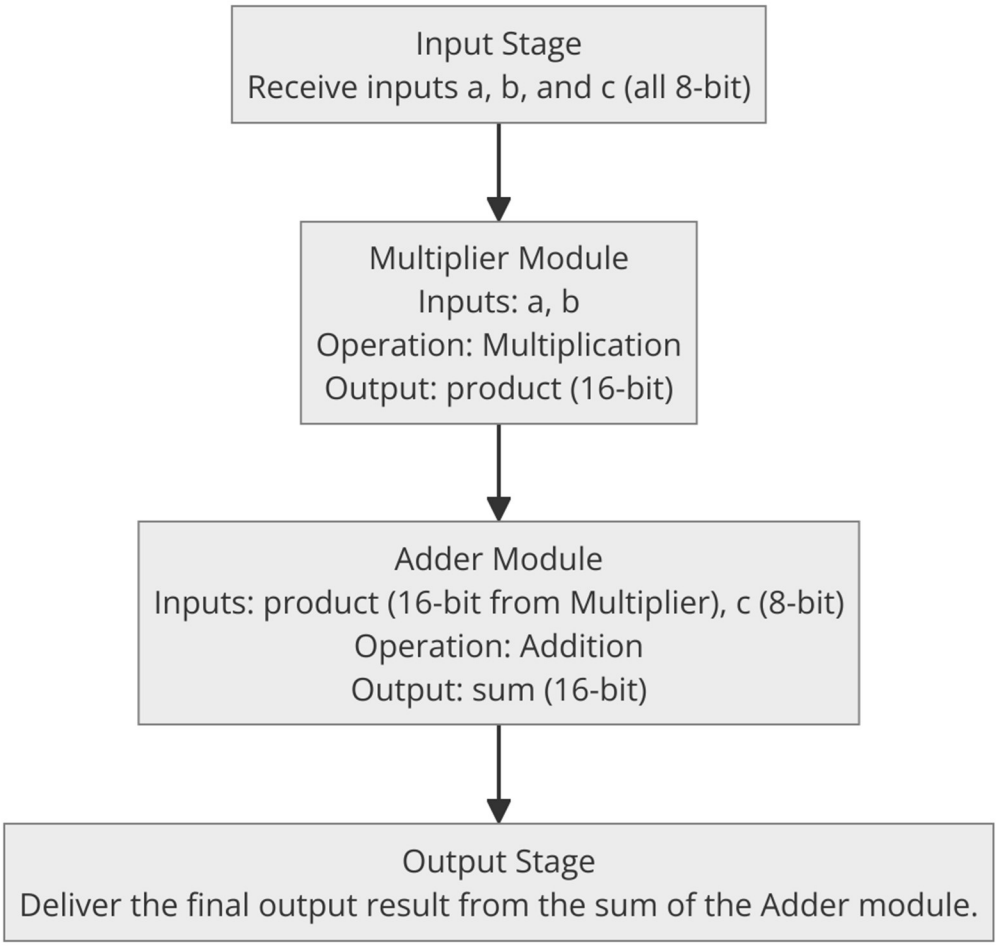
```
module PAU_unit_tb;
reg [7:0] a;
reg [7:0] b;
reg [7:0] c;
wire [15:0] result;
```

```
PAU_unit dut(.a(a), .b(b), .c(c), .result(result));

initial begin
  a = 8'b10101010; b = 8'b01010101; c = 8'b00001111;
  #10;
  a = 8'b11111111; b = 8'b00001111; c = 8'b00001111;
  #10;
  $finish;
end
endmodule
```

6. RTL Design and Micro-Architecture

The RTL design and micro-architecture stage involves defining the detailed structure of how the Parallel Arithmetic Unit (PAU) will be implemented in hardware. This stage translates the high-level module designs into a synthesizable Register Transfer Level (RTL) design, focusing on how data moves between registers and the operations performed on the data at each stage of the circuit.



Micro-Architecture Description

The micro-architecture of the PAU defines the internal structure and interconnections of the hardware components required to implement the RTL design. Here's a detailed overview:

1. Registers and Buses:
    - Input Registers: Three 8-bit registers to hold a, b, and c upon input.
    - Intermediate Registers: One 16-bit register to store the product from the Multiplier before it is sent to the Adder.
    - Output Register: A 16-bit register to store the final result.

2. Functional Units:
    - Multiplier: A combinational logic block that takes a and b from the input registers, calculates the product, and stores it in the intermediate register.
    - Adder: A combinational logic block that takes the product and c, computes the sum, and stores it in the output register.

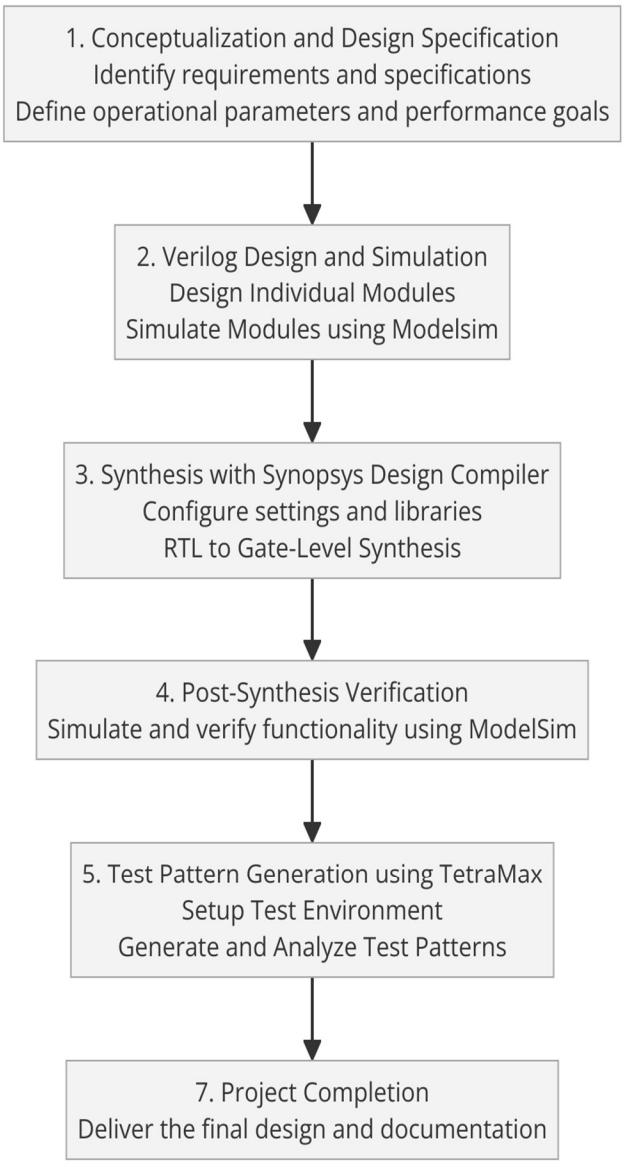
3. Control Logic:
    - A simple control unit that manages the flow of data between registers and functional units, ensuring that operations are triggered at the correct cycle and data is correctly routed through the PAU.

4. Data Path:
    - The data path includes the buses that connect input registers to the Multiplier, the Multiplier to the Adder, and the Adder to the output register. It is optimized for minimal latency and high throughput.

5. Timing and Synchronization:
    - The entire PAU operates under a single clock domain to ensure synchronization across all operations. The clock signals are distributed to all registers and combinational blocks to control the timing of data capture and operation execution.

7. Project Workflow Overview

This is my project workflow, which covers every stage from design in Verilog through post-synthesis verification and test pattern generation using TetraMAX. It covers the whole life cycle of Parallel Arithmetic Unit (PAU) development:



8. Verilog Design and Simulation

RTL Design

Understanding Hardware Description Languages (HDLs) is essential for any digital designer involved in simulating, verifying, synthesizing, and testing their designs. In this project, we utilize Verilog HDL, and our code compilation and simulation are performed using ModelSim – INTEL FPGA STARTER EDITION 10.5b, a Verilog simulation environment provided by Mentor Graphics at Fresno State Apporto Virtual Lab.

Steps for Setting Up and Using ModelSim for Verilog Simulation:

8.1. Access ModelSim:

- Navigate to the Fresno State Apporto Virtual Lab by visiting Fresno State Apporto.
- Log in using your Fresno State credentials.

8.2 Launch ModelSim:

- Once logged in, search for ModelSim within the available applications and launch it. If you have ModelSim installed on your desktop, you can skip these steps and open the application directly from your computer.

8.3 Create a New Project in ModelSim:

- In ModelSim, select File from the menu, then choose New followed by Project to open the project creation dialog box.
- In the dialog box, browse to select the folder where you wish to save your project.
- Name your project and click OK to create it.

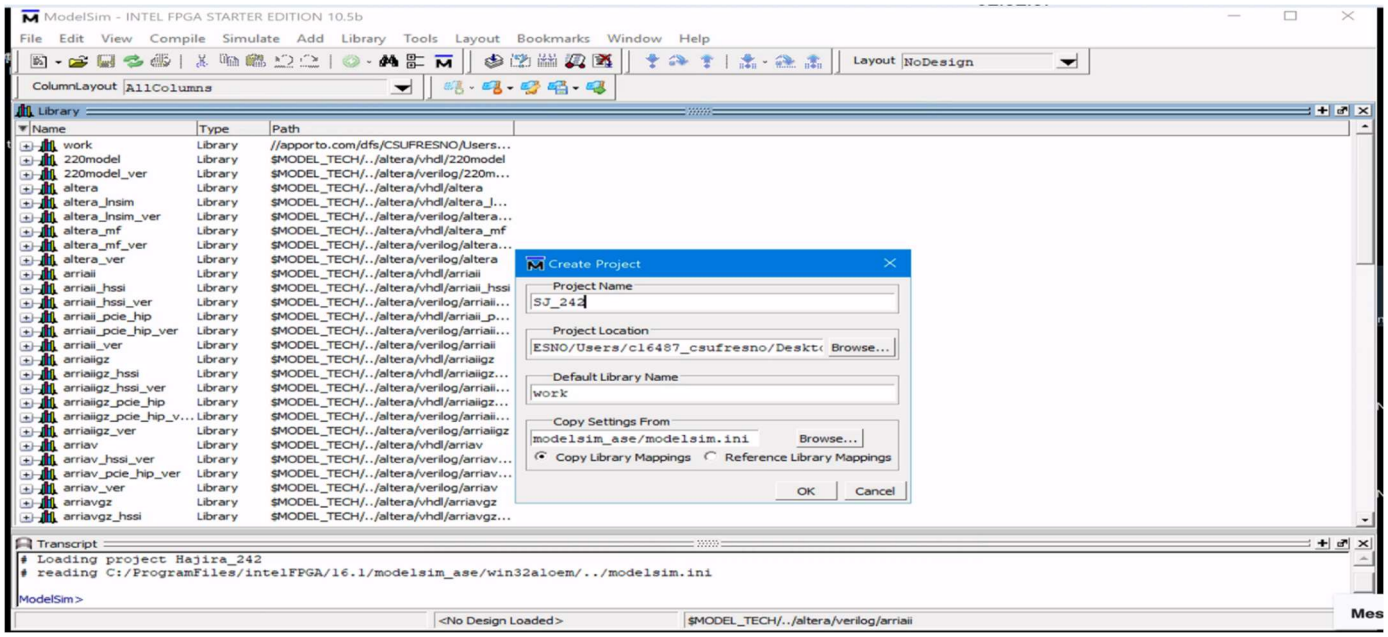


Fig: Creating New Projects in Modelsim

8.4Create a New Verilog File:

Within the project you just created click in the project area, **Add to Project > New File**
When the dialog box for creating a new file appears, enter the file name for your project component. For my project, I kept the file name as **Adder**.
Set the file type to **Verilog**. This will ensure that ModelSim recognizes the file as a Verilog source file, appropriate for writing your HDL code.
Confirm the selection and save the file in your project directory.

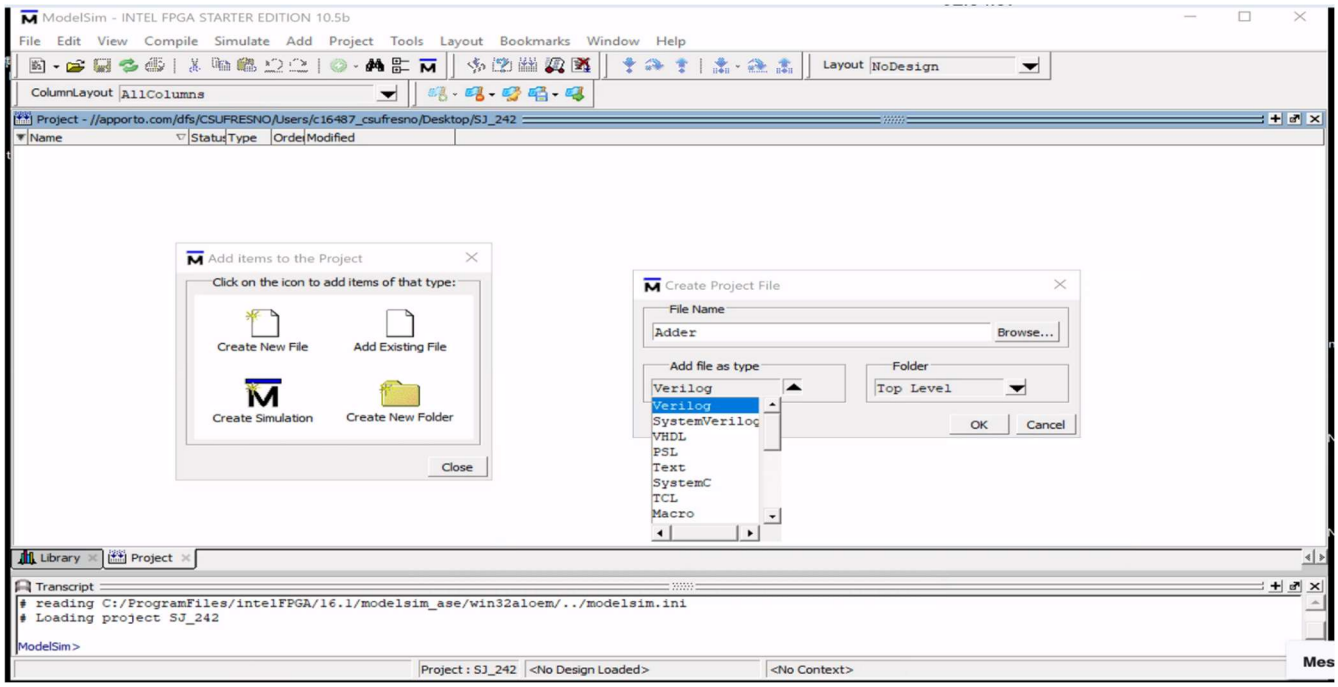


Fig: Create a New verilog file

8.5 Write and Compile Verilog Modules in ModelSim

Writing the Verilog Codes:

- **Create Individual Verilog Files:** Start by creating separate Verilog files for each module within my ModelSim project. We will need to create files for:
  Multiplier.v
  Adder.v
  p_unit.v (The main PAU module)
  tb_p_unit.v (Testbench for the PAU)

Implementing the Modules:

- **Multiplier Module (Multiplier.v):** Code the logic for the multiplier which takes two inputs and outputs their product.
- **Adder Module (Adder.v):** Code the logic for the adder which sums two inputs and provides the output.
- **PAU Module (p_unit.v):** Integrate the multiplier and adder modules within this file, ensuring proper data flow between them.
- **Testbench (tb_p_unit.v):** Write the testbench which will apply test vectors to the **p_unit.v** to simulate its functionality and verify the integration of the multiplier and adder modules.

Compiling the Modules:

- **Open the Project in ModelSim:** Navigate to the project directory where your files are saved.

Compile the Modules:

- Select all the Verilog files you have created (**Multiplier.v**, **Adder.v**, **p_unit.v**, and **tb_p_unit.v**).
- Right-click and choose **Compile > Compile all** from the context menu as shown in the below figure.

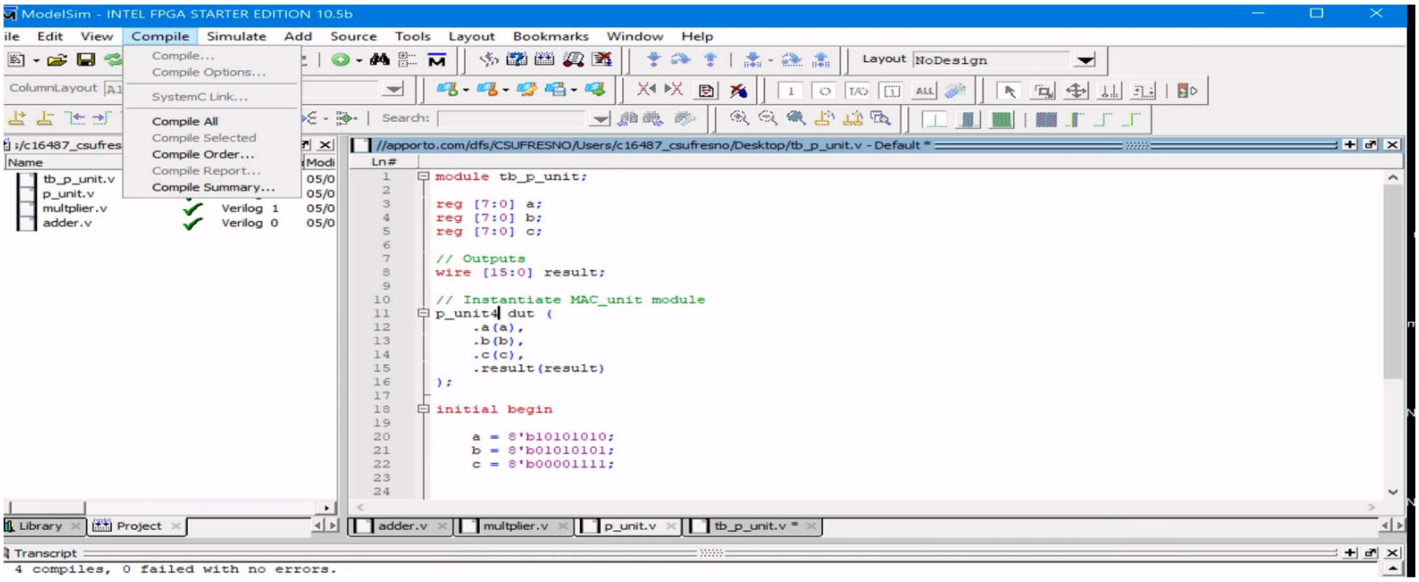


Fig: Compiling all modules in Verilog

Checking the Compilation Results:

Successful Compilation: Look for messages in the transcript window. If there are any errors, ModelSim will display error messages. Review your code for syntax or semantic mistakes and recompile after corrections.
If there are no errors then it will show the successful compilation of all modules as shown in the below figure.

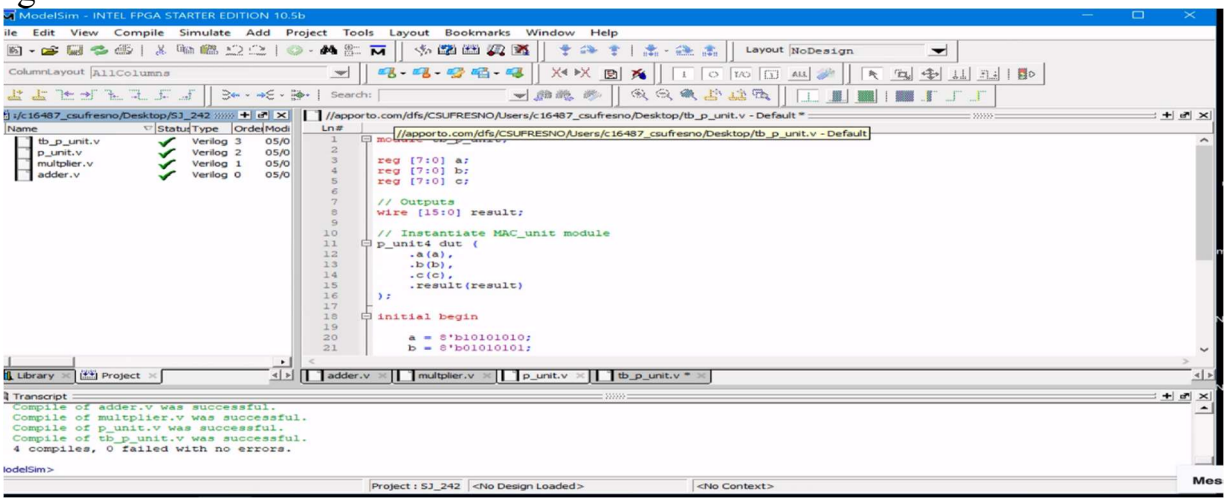


Fig: Successful compilation of all modules

8.6 Start the Simulation:

- In ModelSim, initiate the simulation process by selecting the **Simulate** option from the main menu.
- Locate and Run the Testbench:
- Navigate through the directory structure within ModelSim to **>work>tb_p_unit**.
- Choose your testbench module (**tb_p_unit**) and launch the simulation shown in the below figures.

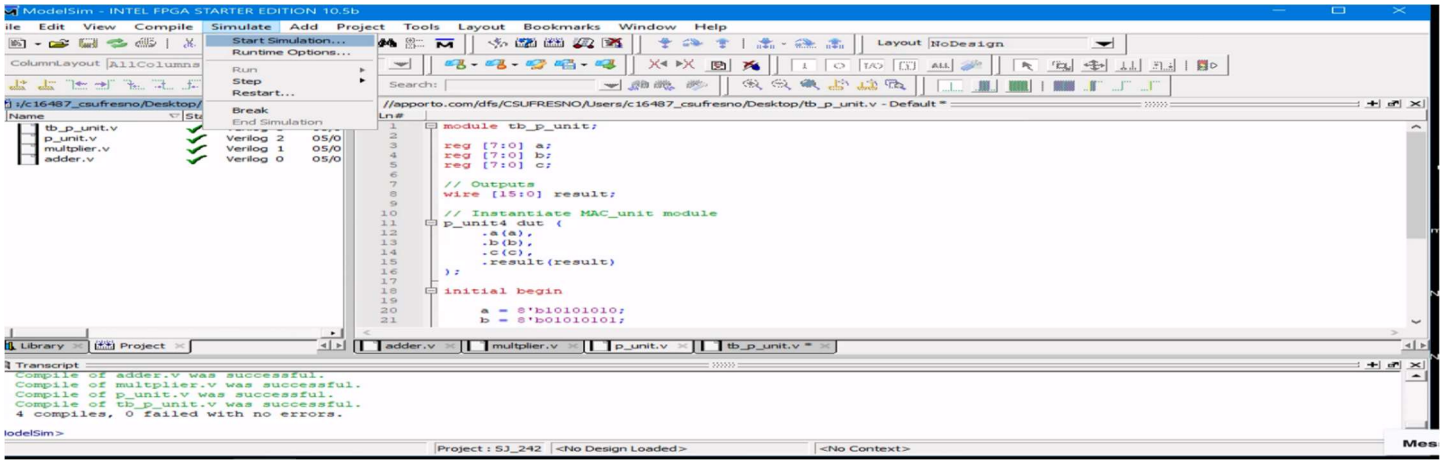


Fig: Select the simulation option from the menu

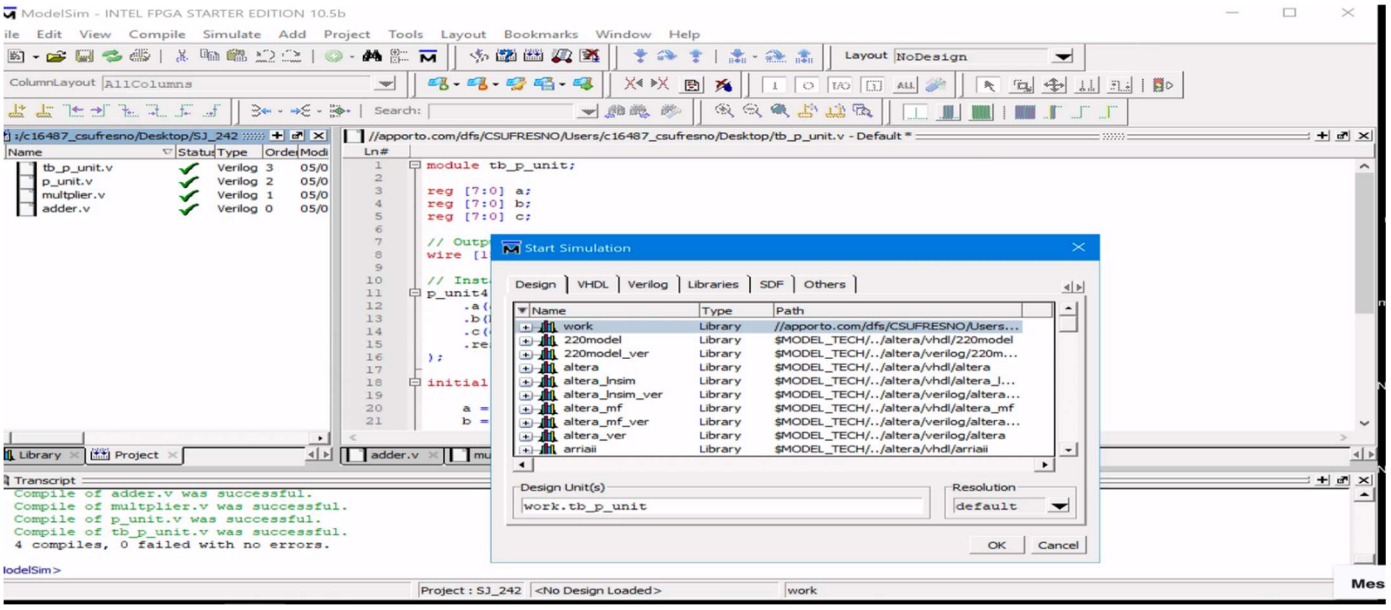


Fig: Select the testbench file that is to be simulated.

8.7Add waveforms and verify the correct functionality of your design:

- After initiating the simulation, select the desired signals or objects in your project that you want to monitor. Right-click on these selected objects and choose "Add Wave" from the context menu to display them in the waveform viewer. With the blank waveform window open, type **run -all** in the command

console to execute the simulation and populate the waveform viewer with the signal activities throughout the simulation duration.

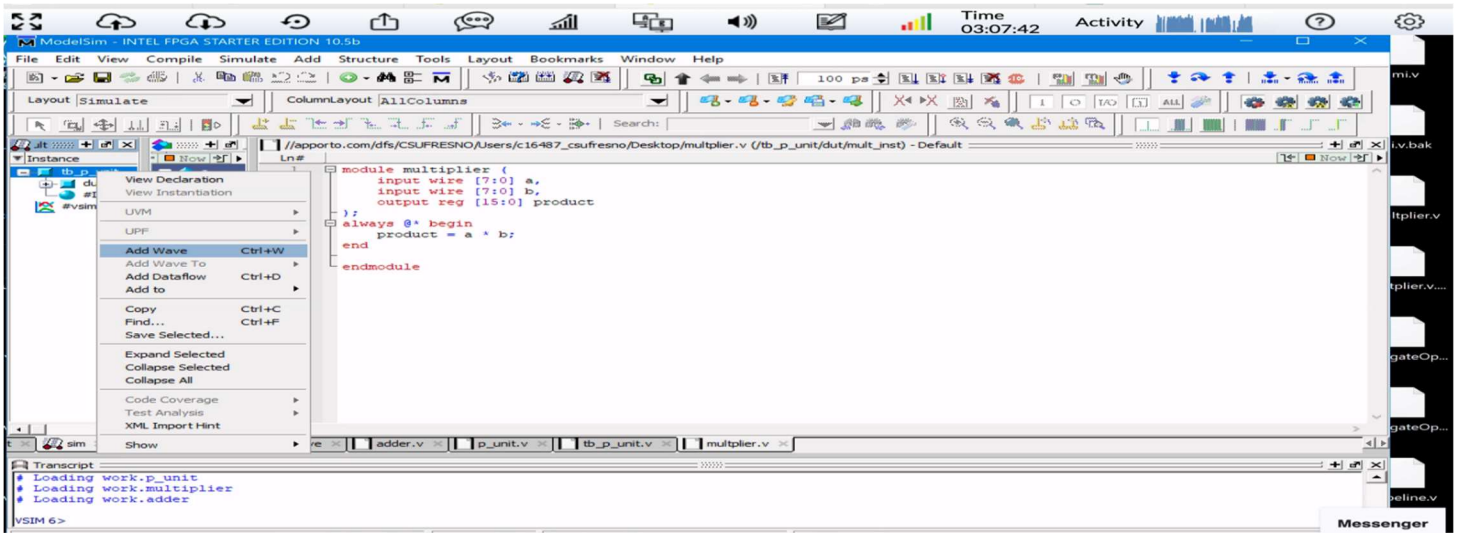


Fig: Adding waveforms

8.8 Completing Pre-synthesis Verification

To finalize the pre-synthesis verification simulation and prepare for the next stages of your project, follow these steps:

Handling Pop-Up Queries:

If a window appears during the simulation asking whether to finish, select "No" to continue observing the simulation results.

Viewing the Simulation Waveform:

Examine the resultant simulation waveform which displays the test vectors from our testbench shown in the below figure and ensure that all functionalities are behaving as expected.

Concluding the Simulation:

Once you have verified the functionality, type quit-sim in the command tab to exit the simulation. This allows you to either add further design and test files to the current project or shift focus to other projects.

Pre-Synthesis Verification:

This simulation step is crucial for confirming that the design meets the required functionalities before moving on to the synthesis stage. The verified design file is now ready to be forwarded to the synthesis process, which will be explained in the next section.

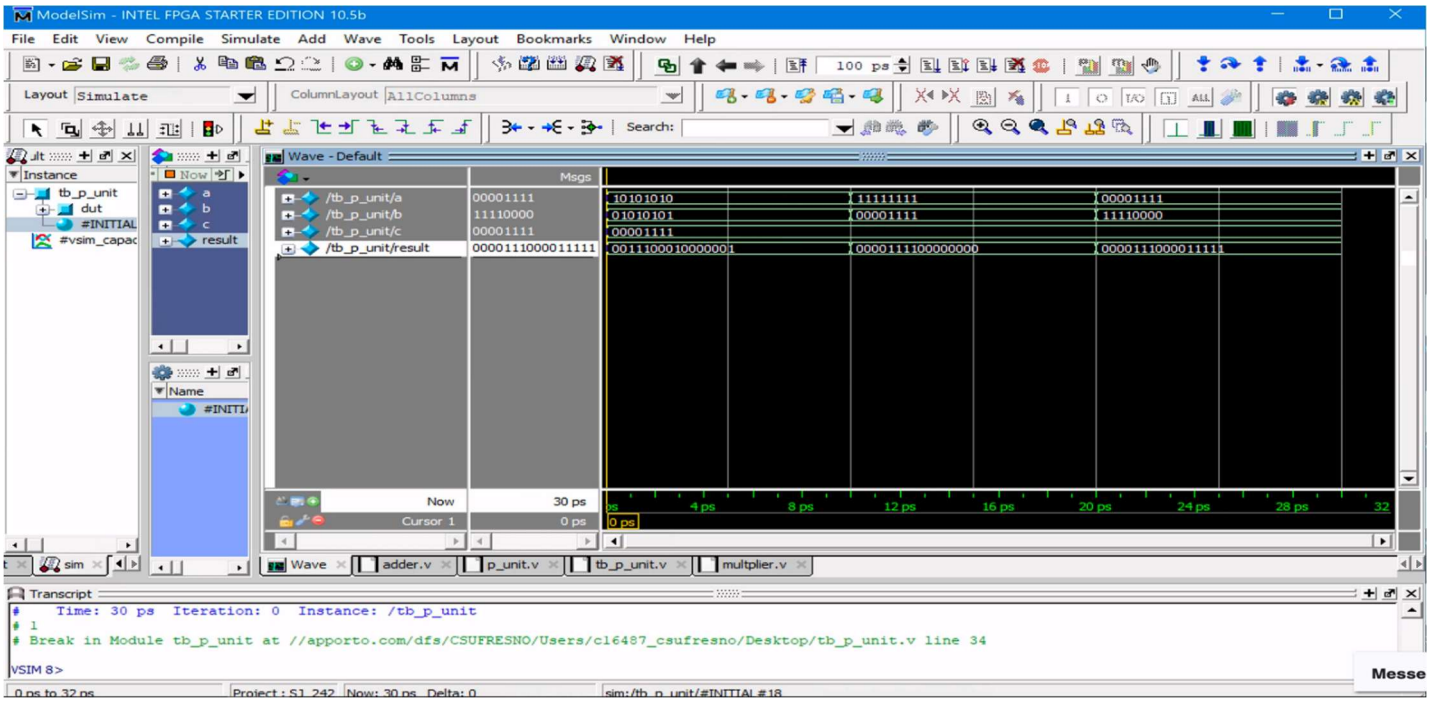


Fig: Verification of Parallel Arithmetic Unit through waveforms

9 Synthesis Using Synopsys Design Compiler

For the synthesis of the functionally verified Verilog HDL designs, the Synopsys Design Compiler (DC) is employed, which is available in a Linux Operating System VirtualBox environment provided by Dr. Aaron Stillmaker. Here are the detailed steps to follow in the synthesis process:

Setup for Synthesis

9.1 Design Compiler Interfaces:

The Synopsys Design Compiler offers two user interfaces: **dc_shell** (command-line interface) and **Design Vision** (GUI). For this project, we primarily use **dc_shell** due to its efficiency and the availability of familiar commands and scripts, though the GUI is also suitable for beginners.

9.2 Preparing the Environment:

Access the virtual system and navigate to the **DCFilesandExamples** folder located in the **Documents** directory. This folder contains essential files and scripts used as a foundation for synthesis.

9.3 Creating a Project Directory:

Create a new project folder named according to your specific project needs (for this example, it could be **SJ MAC**). Copy all necessary contents from **DCFilesandExamples** into this new directory. This setup will host all the modified

and additional files for your synthesis process.

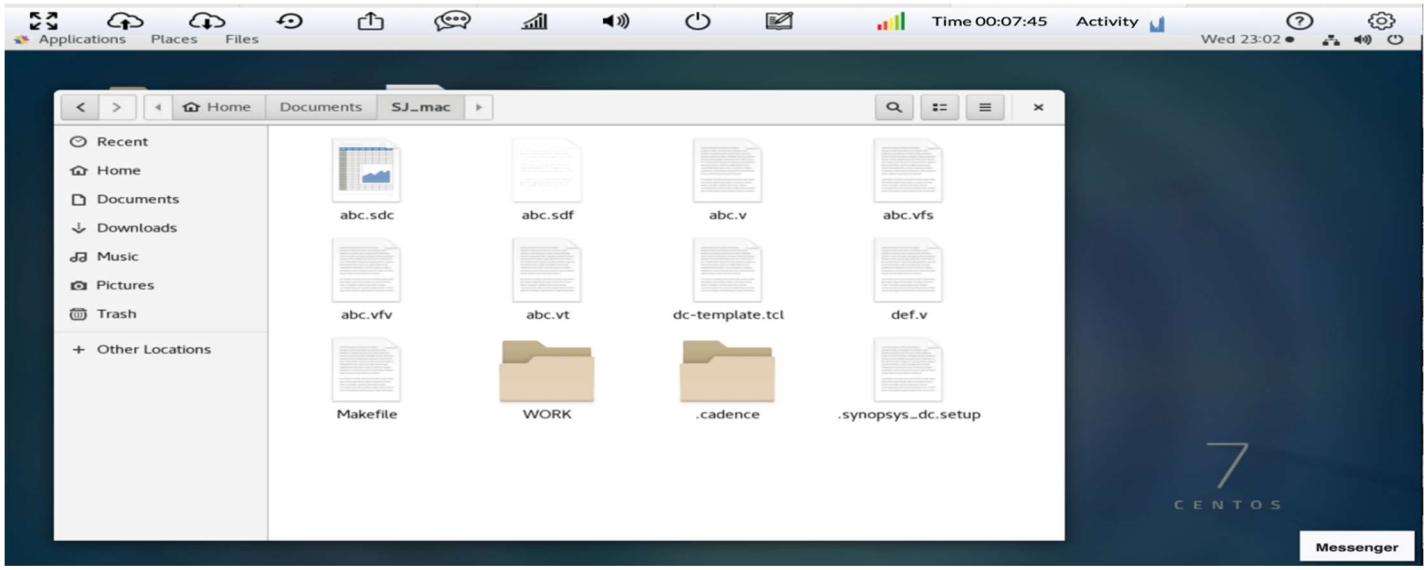


Fig: Contents present in DCFilesandExamples that is to be copied to a project folder

9.4. **Modifying Verilog and Support Files:**

- Replace the existing code in the **abc.v** file with your own Verilog HDL, which should be the contents of your **p_unit.v**. Save this file under the name **p_unit.v** to match our project specifications.
- Replace the existing code of abc.vt with our testbench code and name it as p_unit.vt.
- Similarly, update the **abc.vfs** file to only include your primary module file (**p_unit.v**). Save these files using names that align with your module to ensure clarity and consistency.

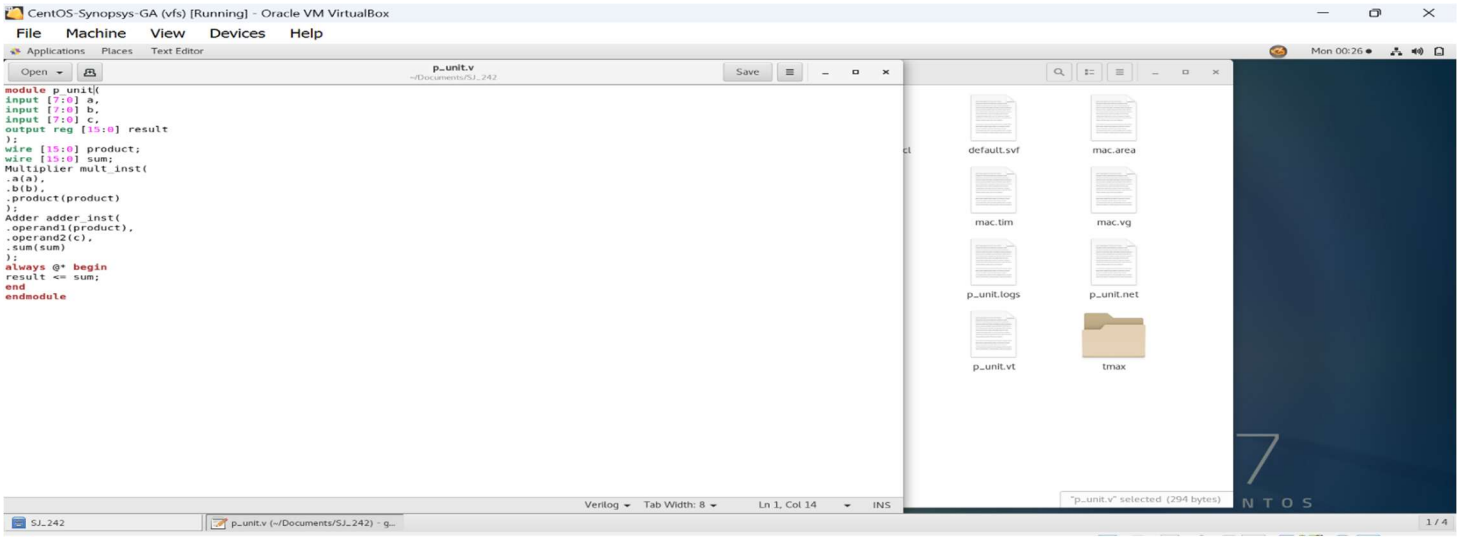


Fig: Replace our Verilog code with the original Verilog code in abc.v

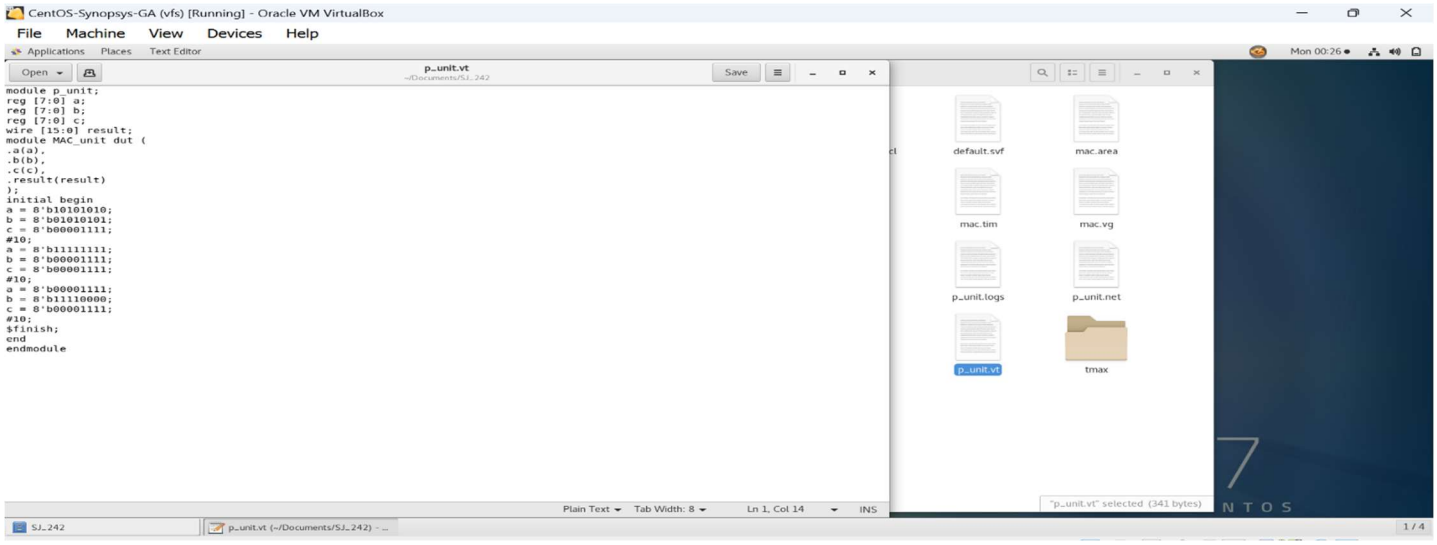


Fig: Replace our Verilog code with the original Verilog code in abc.vt

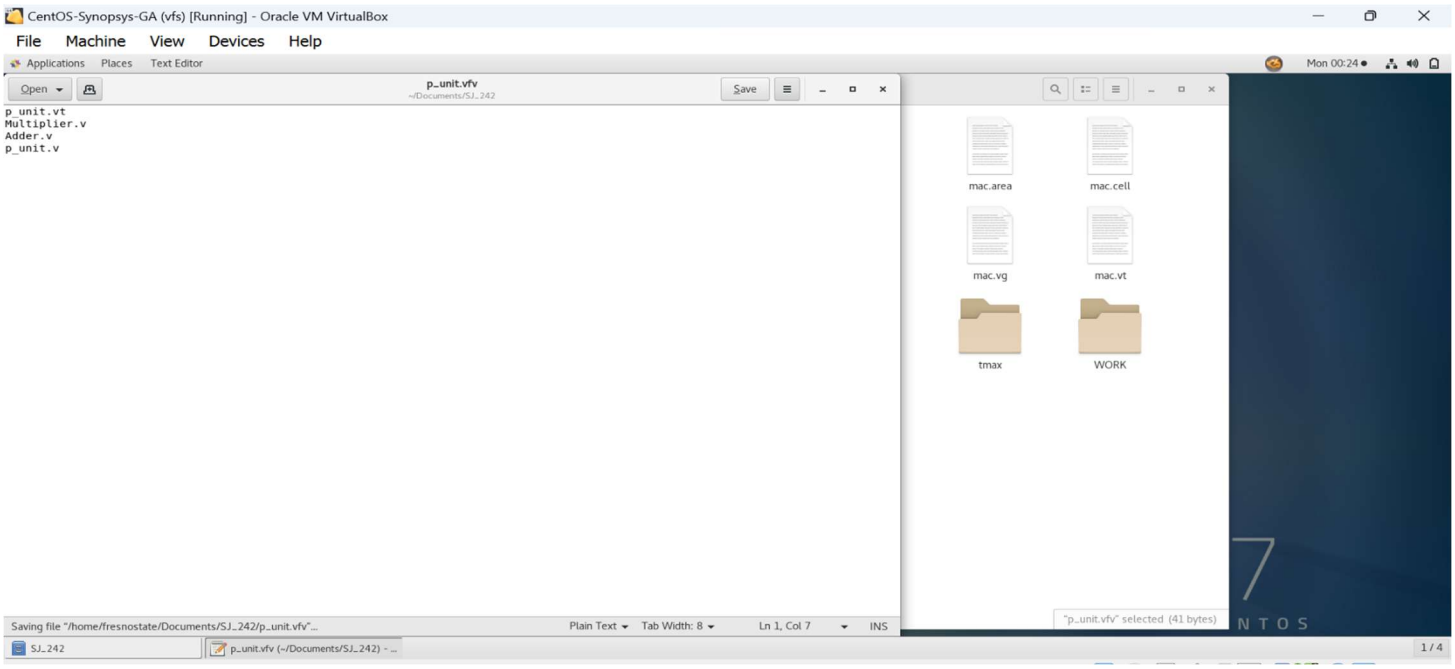


Fig: Replace our Verilog code with the original Verilog code in abc.vfv

Similarly, update the **abc.vfs** file to only include your primary module file (**p_unit.v**). Save these files using names that align with your module to ensure clarity and consistency.

9.5. Configuring the Makefile:

- Open the **Makefile** located in your project directory. Update the top module name setting to **p_unit**, reflecting your main Verilog module.
- Ensure the Makefile has appropriate targets set up for compiling and synthesizing your HDL code, including clean-up commands and synthesis procedures.

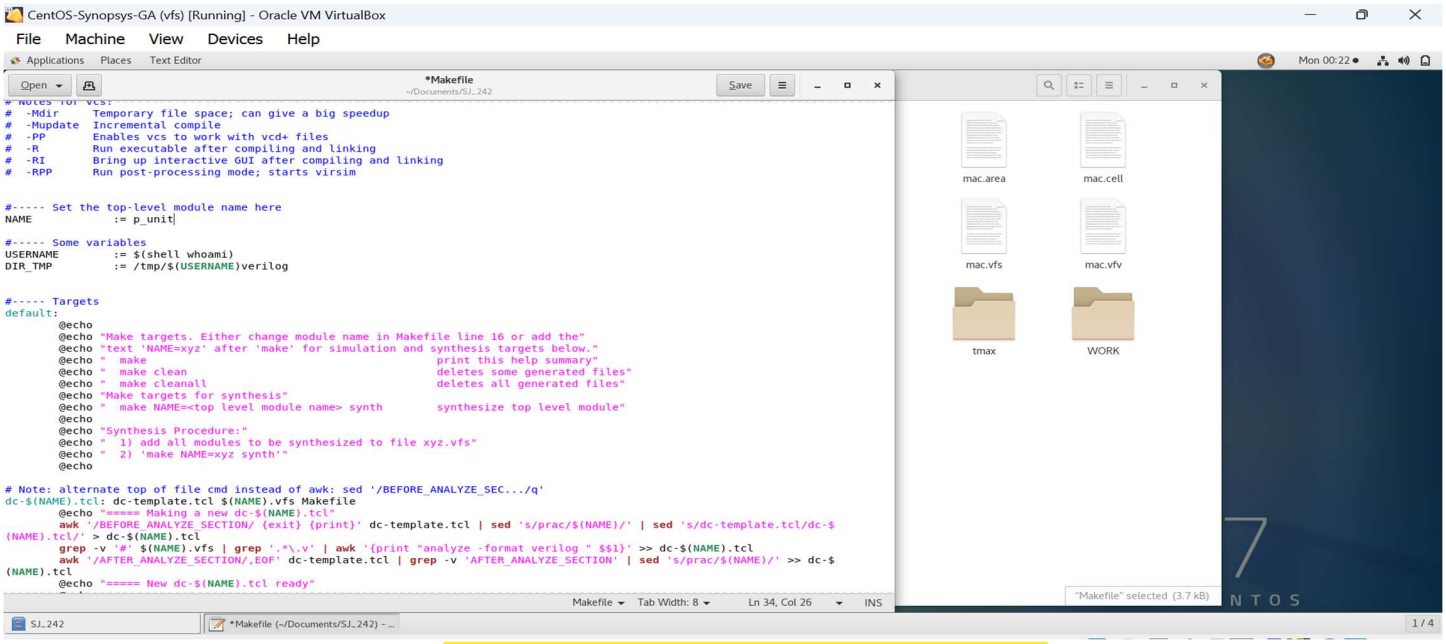


Fig: Update the top module in Makefile

9.6. Once your project setup and files are properly configured, you can proceed with the synthesis of our module. Here are detailed steps to open a terminal, execute the synthesis command, and obtain the results:

Step 1: Opening the Terminal Window

- **Access Terminal:** To open a terminal window in your VirtualBox environment, right-click on the desktop or within the folder where your project is located. Choose 'Open Terminal Here' or a similar option as indicated in the context menu.

Step 2: Preparing for Synthesis

- **Launch Terminal Window:** Once the terminal window is open, ensure you are in the correct directory where your project files (**p_unit.v** and the Makefile) are located. This can typically be checked by entering **pwd** (print working directory) and **ls** (list directory contents) commands to confirm your current path and visible files.

Step 3: Executing the Synthesis Command

- **Enter Synthesis Command:** In the terminal, type the specific command provided in your project documentation or the Makefile to start the synthesis process. This command usually begins with a call to make followed by a synthesis-specific argument, like make synth, which engages the Design Compiler to translate your Verilog code into a gate-level netlist.
- **Execute the Command:** Press Enter to execute the command.

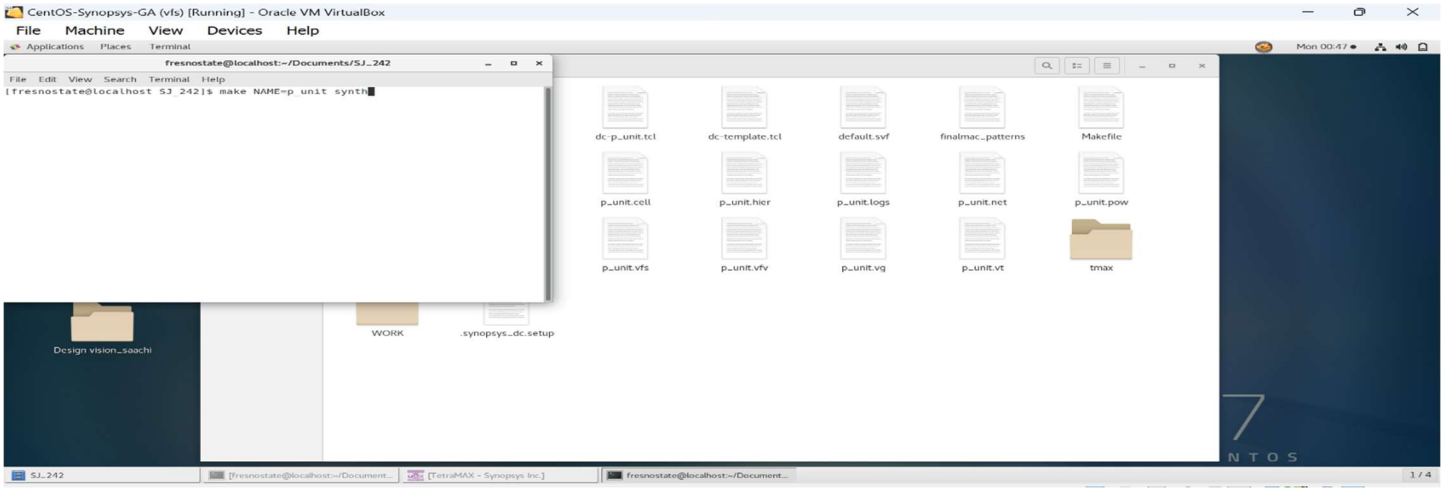


Fig: Executing synthesis command

9.7 Note on Synthesis Execution and Results:

- **Execution:** Press **Enter** to start the synthesis, transforming your Verilog design into a gate-level netlist with accompanying area and power reports.
- **Monitoring:** Synthesis time varies based on design complexity and computer performance; watch the terminal for status updates.
- **Results:** After synthesis, find the gate netlist and reports in the current directory, accessible by minimizing or exiting the terminal.

Step 4. Accessing Synthesis Reports:

Report Files: After synthesis, the reports will be saved in the specified folder:
.vg file: This is your gate-level netlist.
.area file: Contains the area results of your synthesized design.
.pow file: Provides the power analysis results.

These files can be seen in the figure below.

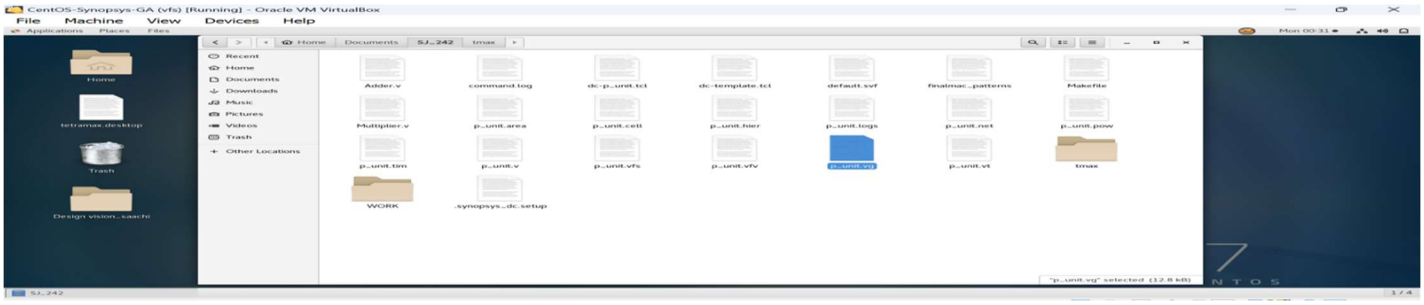


Fig: Synthesis Reports saved in the specified folder

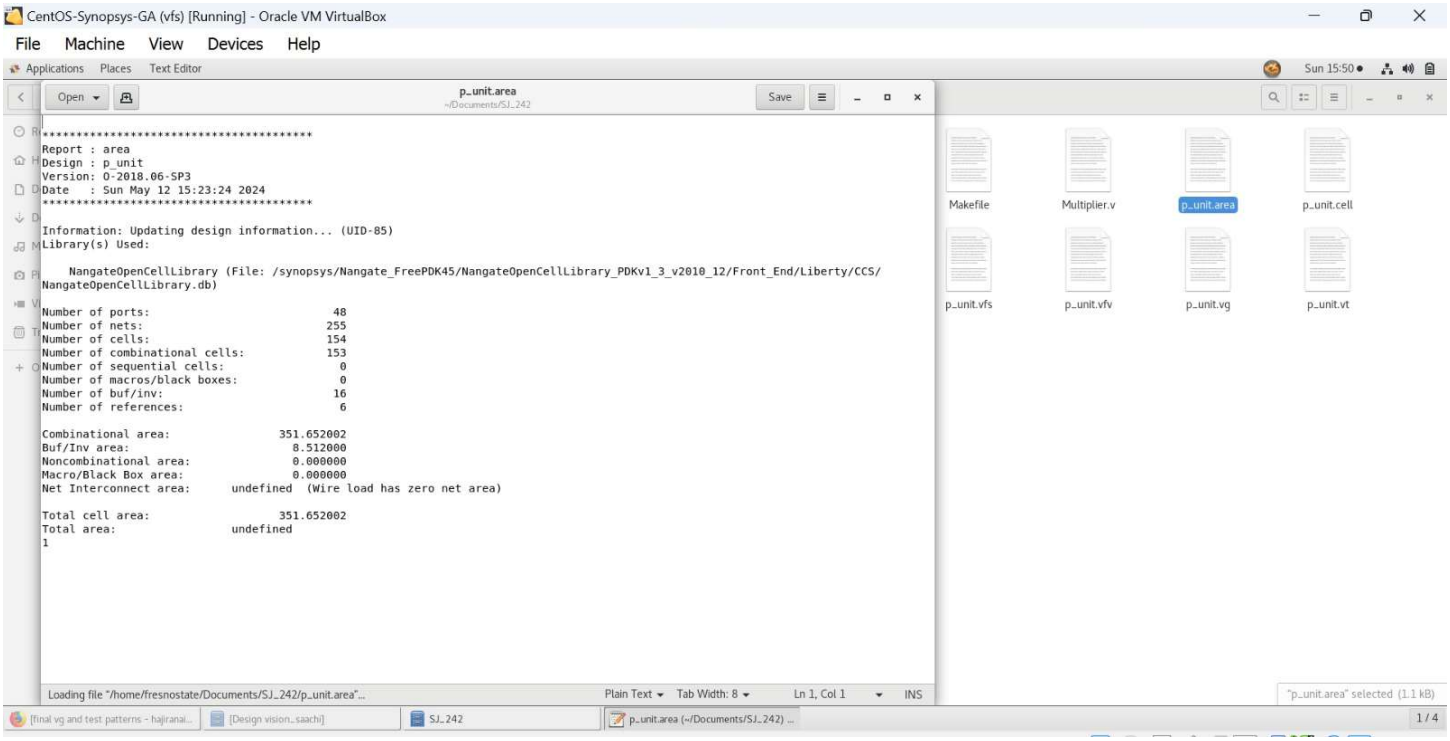


Fig: Area Results

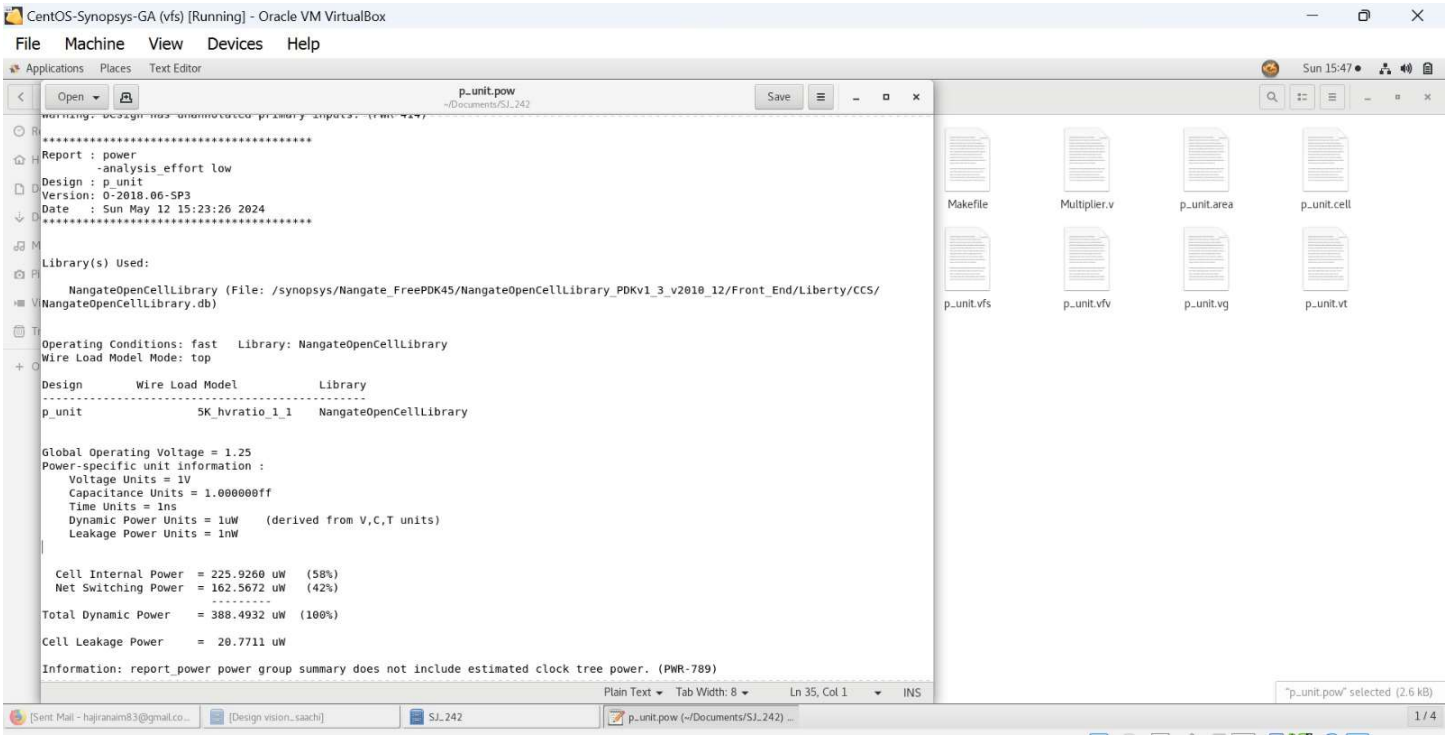


Fig: Power Results

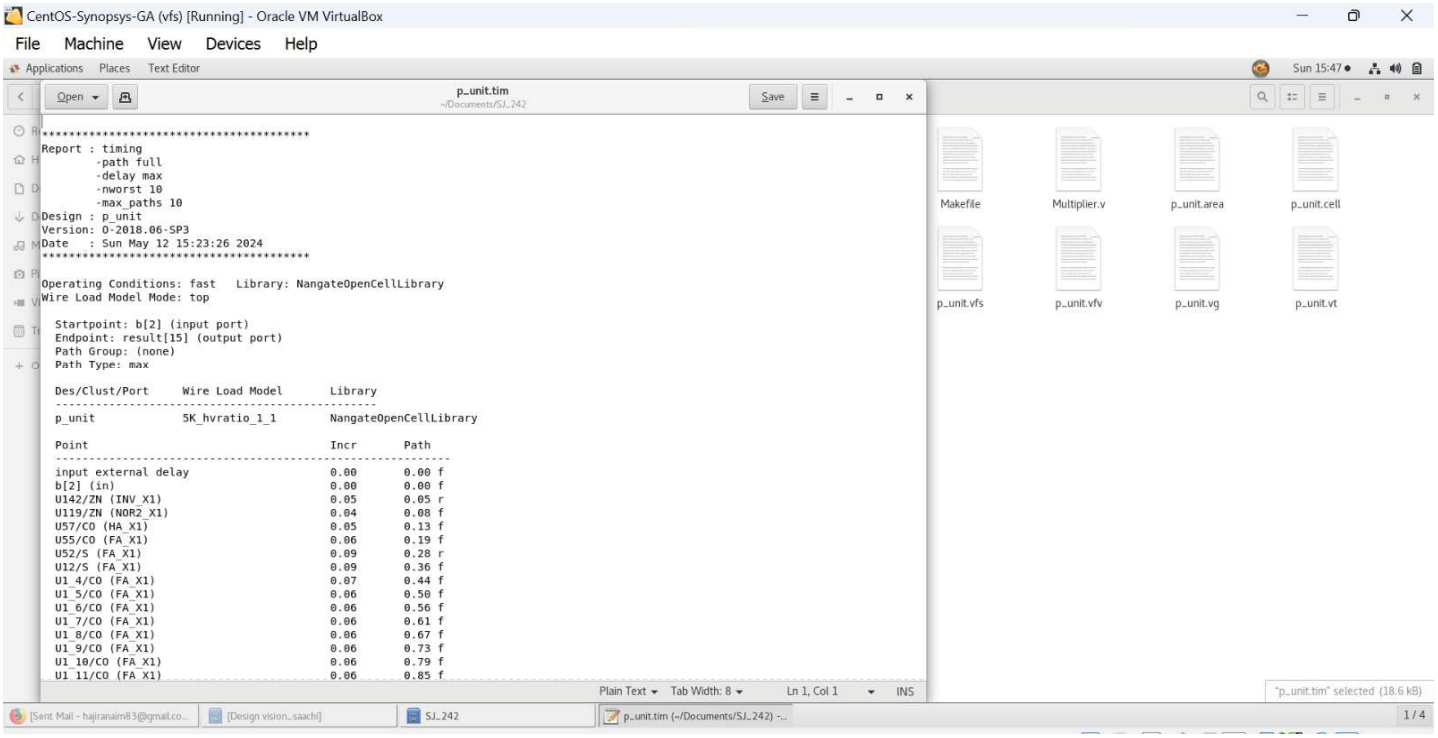


Fig: Timing Report

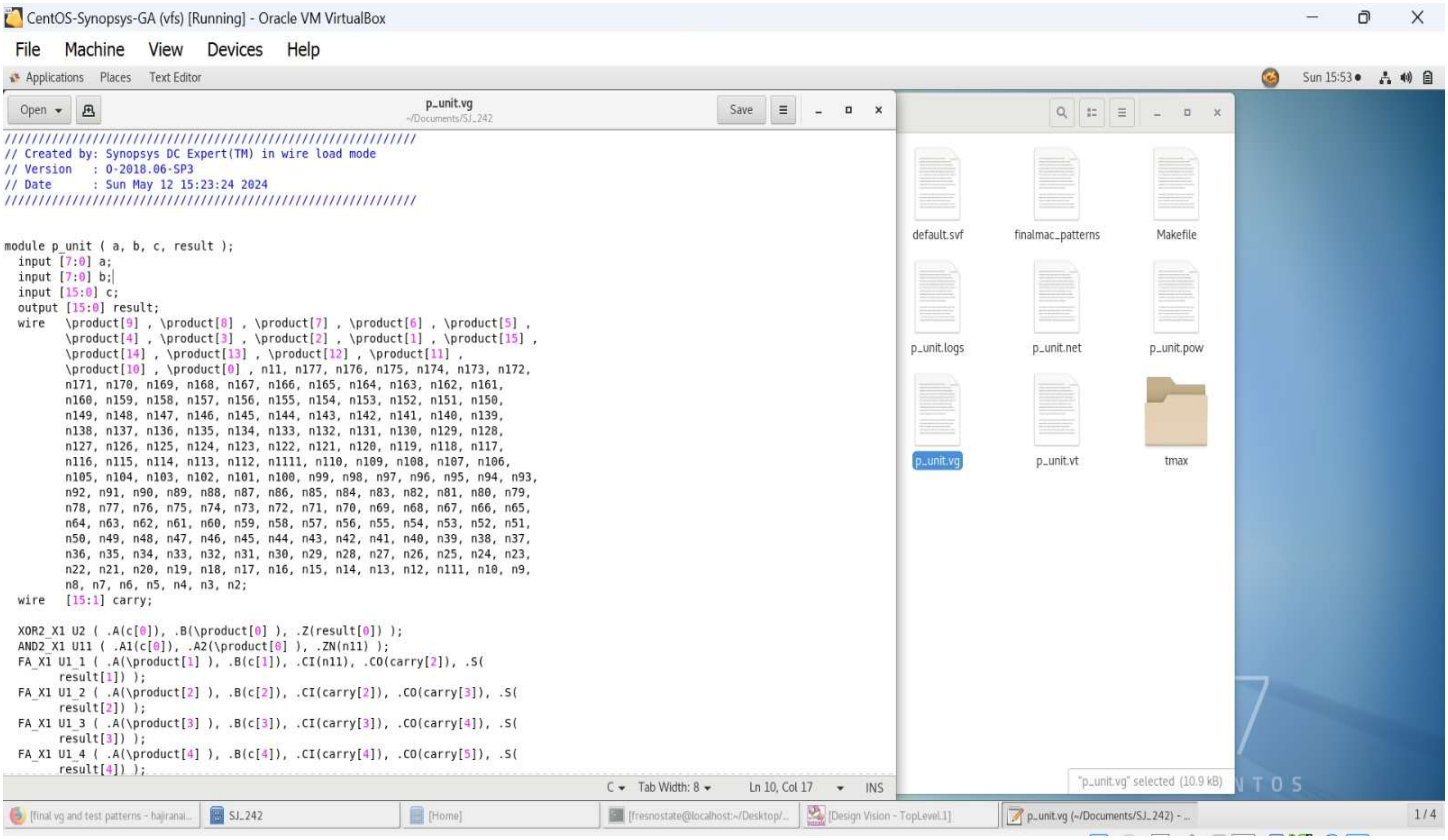


Fig: Netlist Results

10. Post-synthesis Verification:

After obtaining the gate netlist, it's crucial to verify the synthesized netlist's functionality to ensure it adheres to the intended technology standards. This process is referred to as "post-synthesis verification."

10.1: Setting Up for Post-Synthesis Simulation

- **Prepare for Simulation:** Follow the same initial steps as in the pre-synthesis simulation, up to the point of creating a project, creating a **.**v file.
- **Saving the File:** Instead of using the original file name, save the new file with the name **p_unit_vg_ps.v**, indicating that this is the post-synthesis verification file. This step is demonstrated in the below figure.

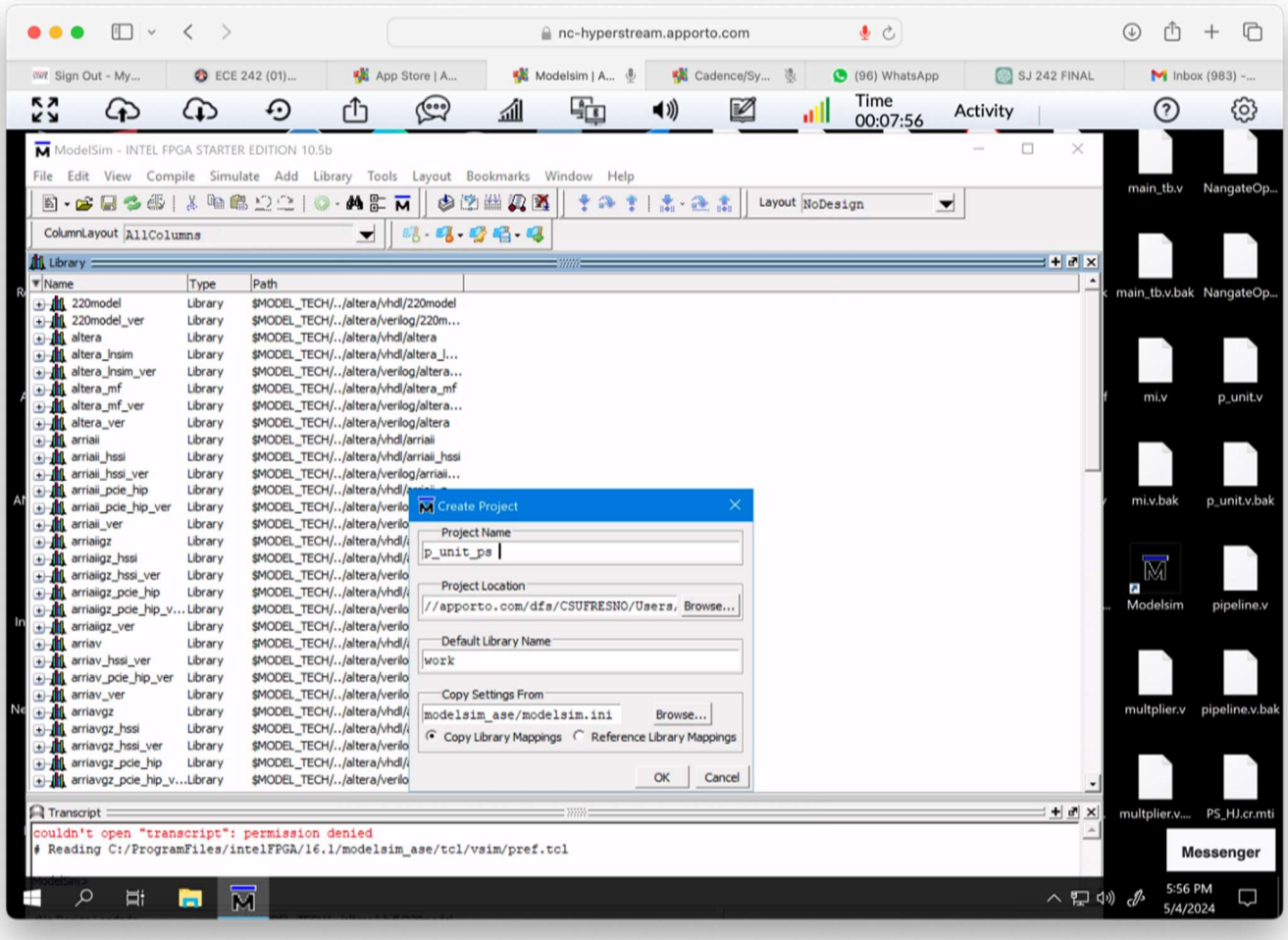


Fig: Creating a new project

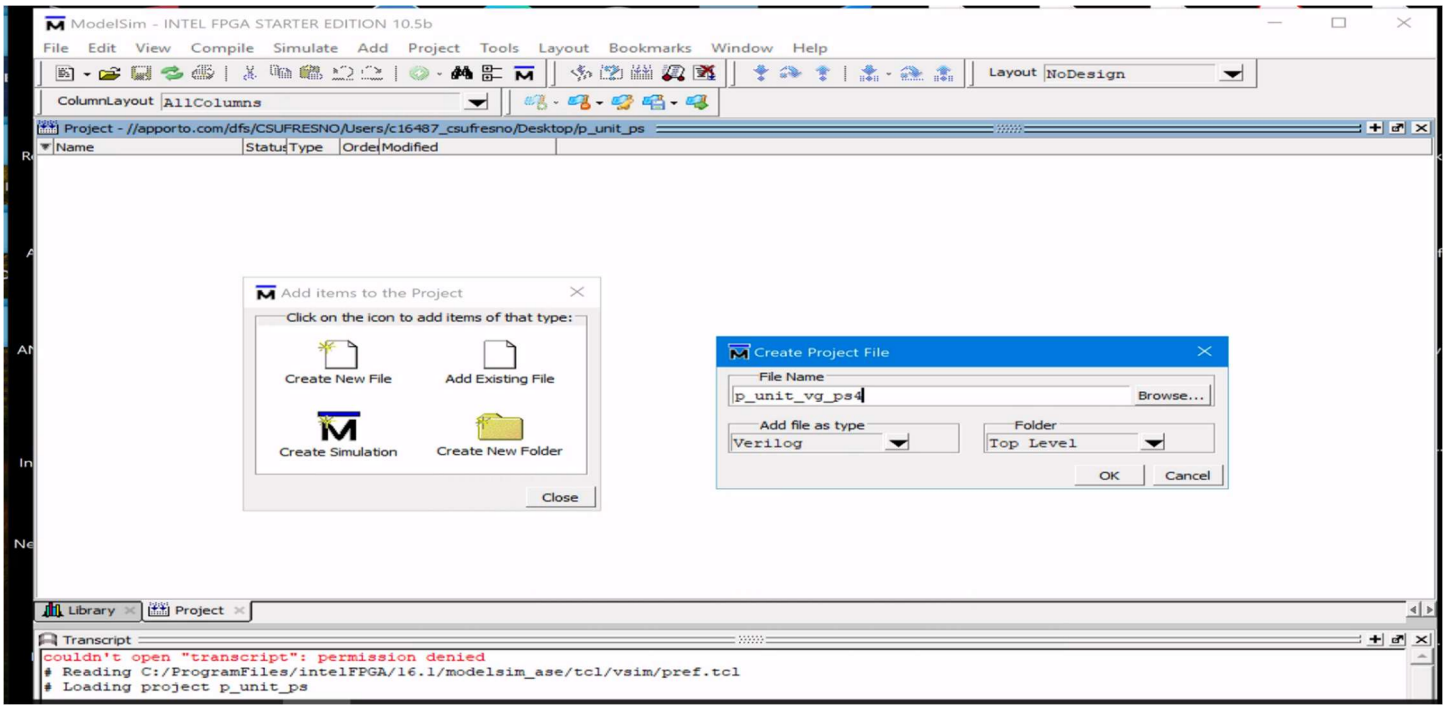


Fig: Creating a p_unit_vg_ps.v file to simulate the netlist synthesized from the synthesis step

Copy and paste the gate netlist synthesized from the Synthesis step and the testbench from the previous steps into the new files p_unit_vg_ps.v and P_unit_ps_tb.vt

10.2: Compilation of Post-Synthesis Files

- **Compile Files:** Load both the **p_unit_vg_ps.v** and **P_unit_ps_tb.vt** files into your Verilog compiler.
- **Error Handling:** If the compiler identifies errors due to unrecognized gates, you'll need to include the gate definitions from your synthesis library. Add the **NangateOpenCellLibrary.v** file to your project and include it in your top module by adding the line: **include "NangateOpenCellLibrary.v"**.
- **Recompile:** After adding the necessary library file, recompile all files in the project. This should resolve any previous compilation errors.

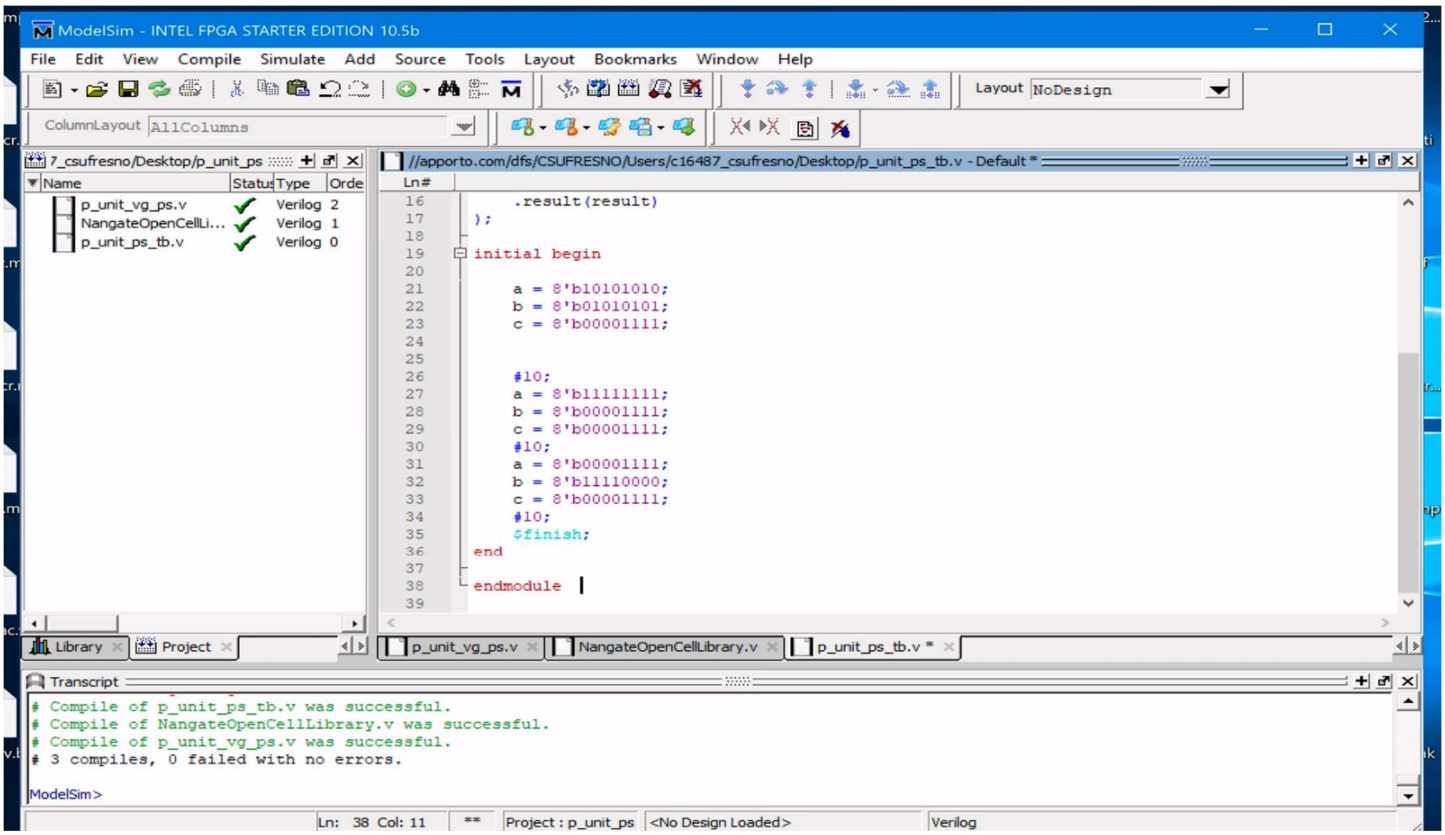


Fig: Compile all the files

10.3: Simulation of Post-Synthesis Design

- **Start Simulation:** Run the simulation using the **P_unit_ps_tb.vt** file to test the post-synthesis functionality of your design.
- **Troubleshooting Simulation Errors:** If you encounter an error such as "Error loading design", it might be due to missing time unit or precision in the library file. To fix this, add the directive **timescale 1ns/100ps** at the beginning of your Verilog files and the library file if needed. This specifies the simulation time unit and time precision, ensuring that the simulator processes delays and transitions correctly.

10.4: Final Post-Synthesis Simulation and Verification

- **Simulate Again:** Conduct a final simulation using the **P_unit_ps_tb.vt** test bench to confirm error-free functionality, including the effects of synthesis such as gate-level delays.
- **Waveform Analysis:** Add and analyze waveforms for critical signals to verify the design's behavior post-synthesis, similar to pre-synthesis verification. The resultant waveform in given in the below figure

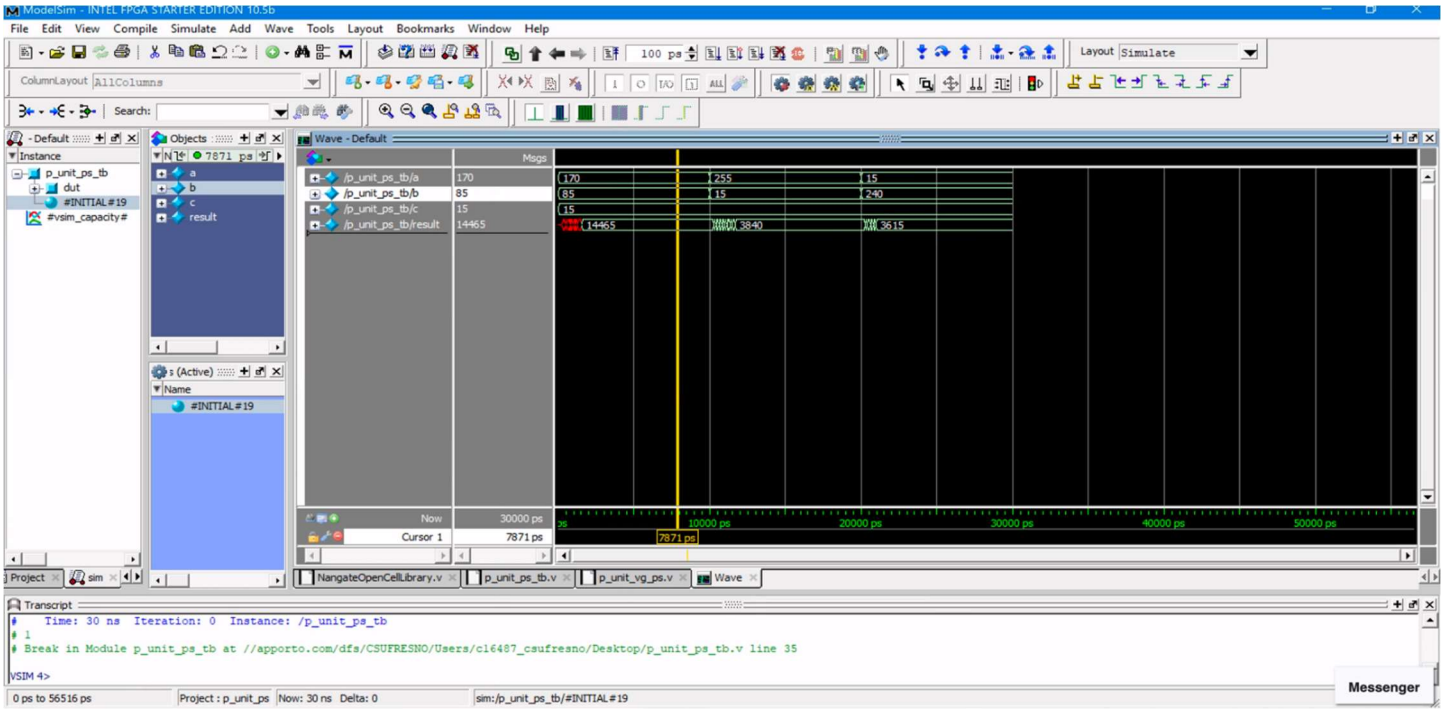


Fig: Post-Synthesis Waveform Analysis

Now we should evaluate the design's testability using ATPG tools like Synopsys TetraMax.

11.Test Pattern Generation using TetraMAX

11.1: Create and Organize Test Files

- Create a Folder: Navigate to your home directory, right-click on an empty space, and choose to create a new folder.
- Add Files: Place the synthesized netlist file and library files into this folder.

Accessing the Technology Library File for TetraMAX

To retrieve the necessary technology library file, follow these streamlined steps:

Open System Locations:

Click Places and select Computer to view your system directories.

Find Synopsys Folder:

Navigate through the directories to locate the Synopsys folder.

Access Nangate_FreePDK45:

Within the Synopsys folder, find and open the Nangate_FreePDK45 directory.

Locate Library File:

Go to the Front_End folder, then enter the Verilog subfolder where the library file is stored.

Select the File:

Identify and select the necessary Verilog library file, essential for your test pattern generation with TetraMAX.

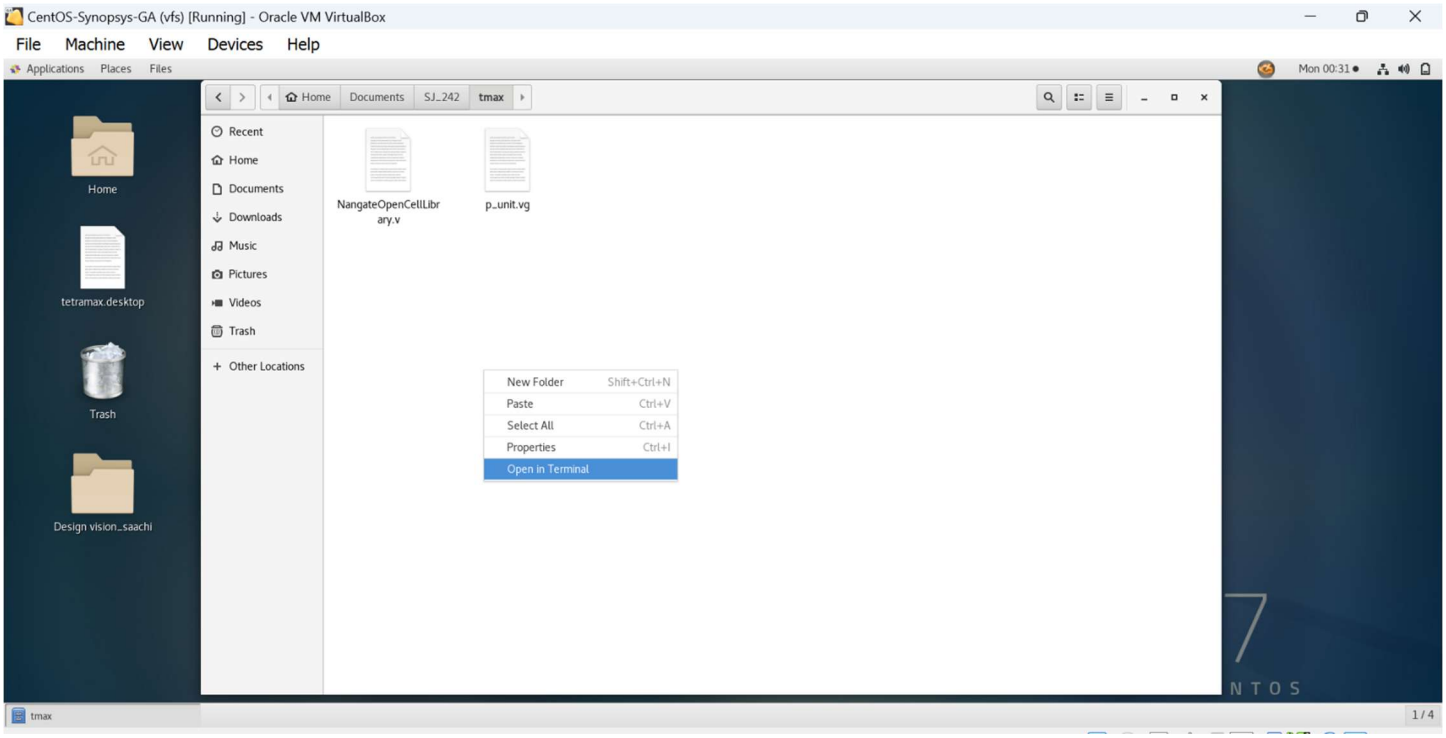


Fig: Open Terminal

11.2: Writing Test Patterns Using Synopsys TetraMax

Test pattern generation with Synopsys TetraMax involves several critical steps to effectively detect faults within a design. Here's a breakdown of the process:
Steps in TetraMax Test Pattern Generation:

1. Netlist: TetraMax starts with the netlist which outlines all logic gates and their connections within your design.
2. Build: The tool builds the model of your circuit based on the netlist.
3. DRC (Design Rule Checking): TetraMax performs a design rule check to ensure the netlist conforms to specified design rules.
4. ATPG (Automatic Test Pattern Generation): TetraMax generates test patterns specifically designed to detect single stuck-at faults.
5. Writing Patterns: These patterns are then written out, and ready to be applied to the circuit either in simulation or on actual hardware.
6. Schematic View: Allows viewing the schematic representation of the netlist and the faults targeted by the test patterns.

11.2.1 Invoking TetraMax:

- Open Terminal: Start by opening a terminal window.
- Launch Command: Input the command to invoke TetraMax. The specific command can be seen in the below figure.

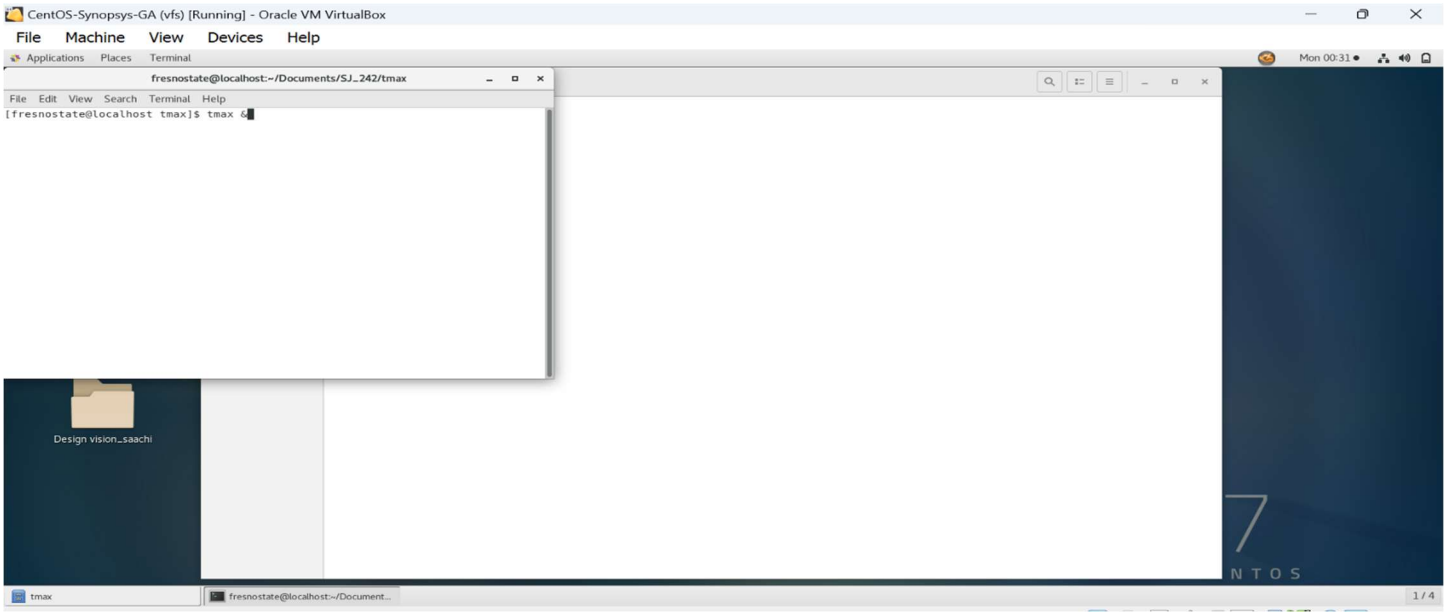


Fig: Invoking Tetramax

Access TetraMax Interface: Refer to the below figure to view the TetraMax home screen.

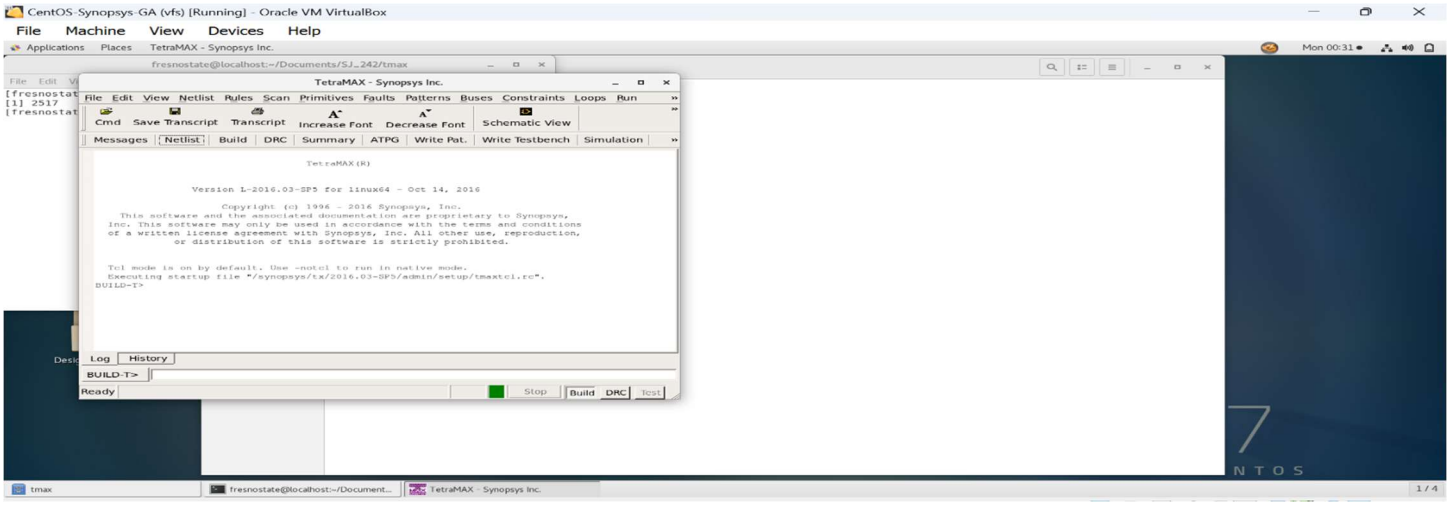


Fig: Tetramax home screen

11.2.2: Import Netlist and Library Files:

- Navigate to the option to open the Netlist (as shown in above figure), where you can import the netlist and corresponding library files. These files are critical as they contain the structural details needed for generating effective test patterns.

11.2.3: Load Library File:

- Navigate to the option for opening library files as shown in the below Figure.
- Ensure to check the box labeled “library modules” when opening the library file. This setting tells TetraMax to treat the file as containing definitions of logic gates and other components used in the design.
- Click on "run" to load the library file into TetraMax.

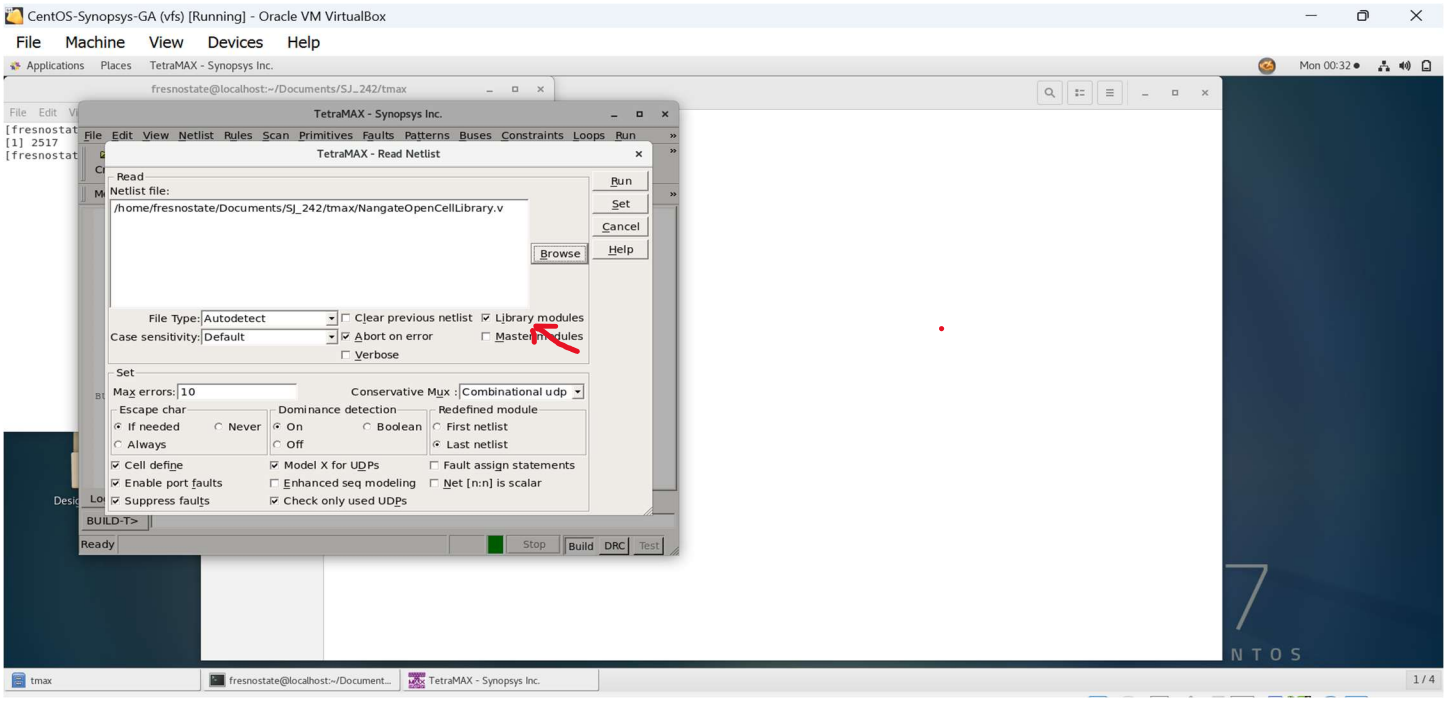


Fig: Library modules are selected while opening the library file

1. **Reading the Netlist File:**
    - Repeat the Process for the Netlist File:
    - Using the same command toolbar, open the netlist file which in our project is p_unit.vg.

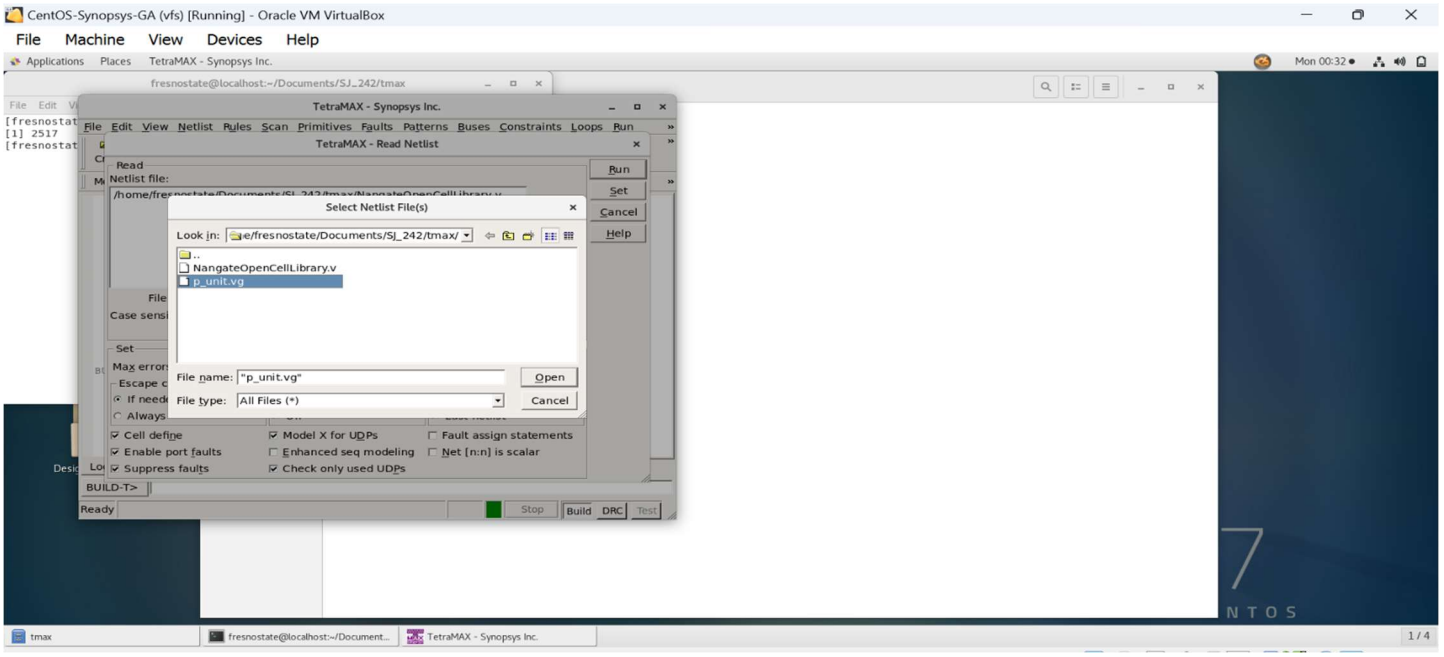


Fig: Repeat the Process for the Netlist File

- This time, ensure that the “library modules” box is **NOT** checked. This setting is crucial because the netlist file should be read as the actual design description, not as library components.
- Execute the command to load the netlist file by clicking "run."

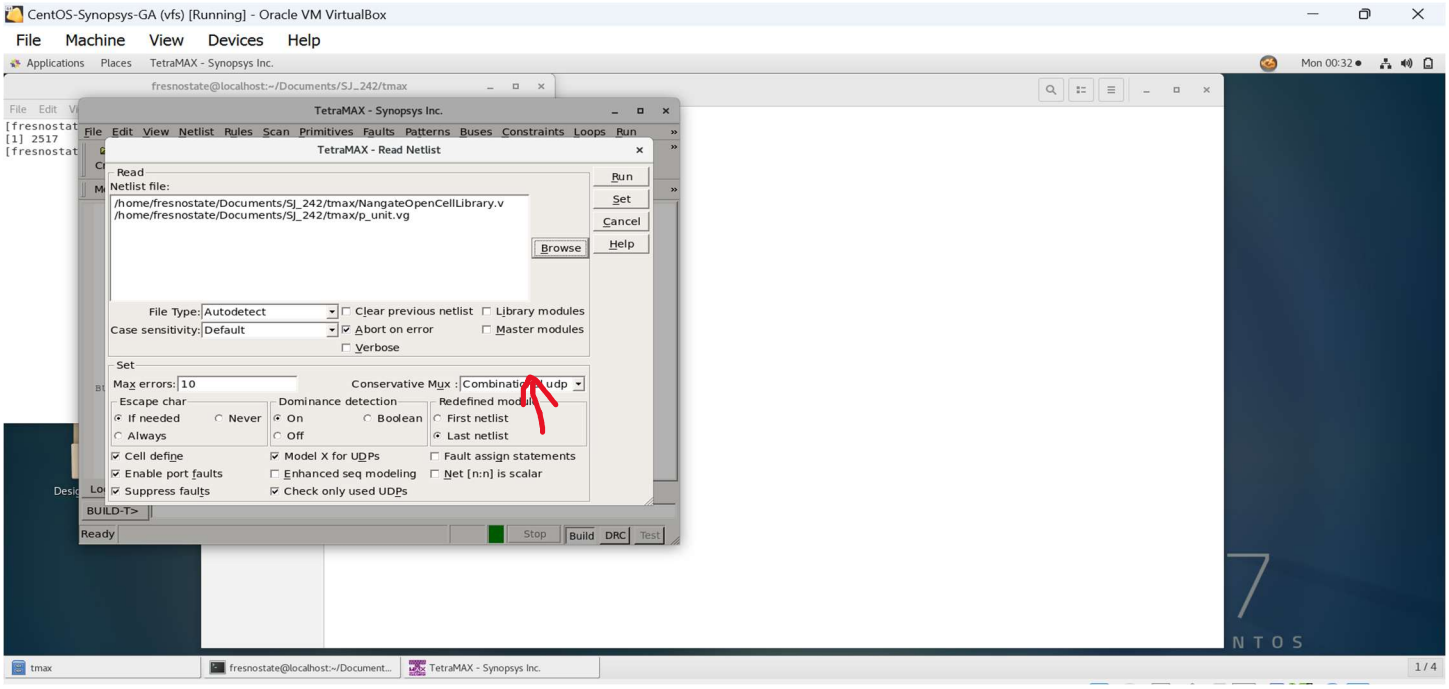


Fig: The “library modules” box is NOT checked

11.2.4Check for Errors:

- After running both the library and netlist files, TetraMax will process the information and display a summary of any errors encountered.
- Review the error summary displayed on the screen to determine if any issues need to be addressed. This step is crucial for ensuring that TetraMax has all the necessary information to generate accurate test patterns.

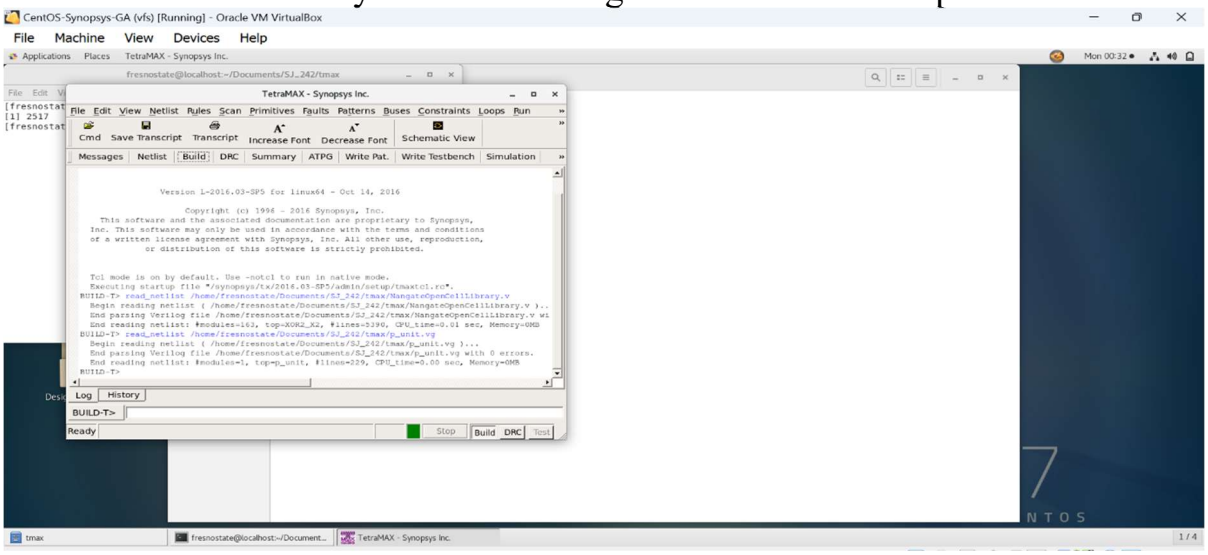


Fig: Displayed error summary

11.2.5 Setting Up the Build in TetraMax

- Select Top-Level Module:
- Verify the top-level module name matches your design's module name.
- Use the dropdown menu in TetraMax to select the appropriate module recognized from the netlist.
- Initiate the Build:
- Click ‘Run’ to start building the test patterns. This triggers the Automatic Test Pattern Generation (ATPG) process.

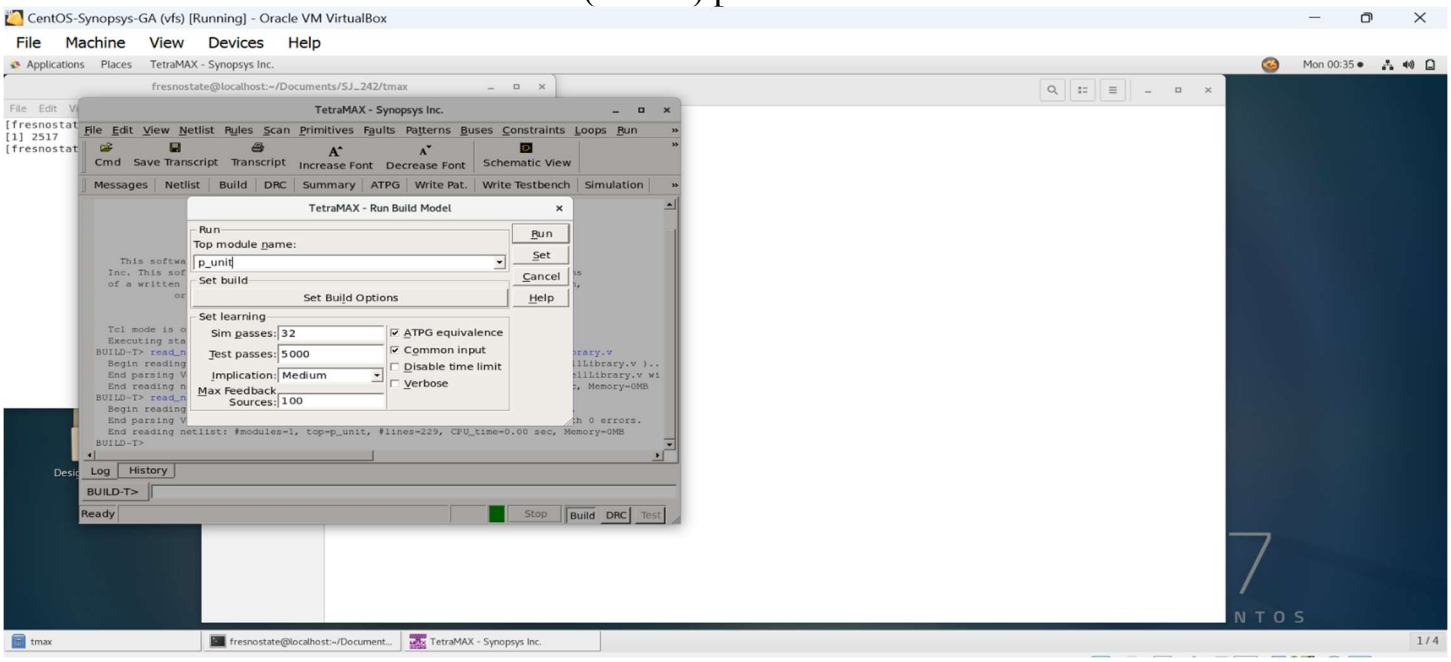


Fig: Run build model

Monitor Progress:

- Observe the Transcript window for a summary and any important messages or errors during the build process, as shown in the below figure.

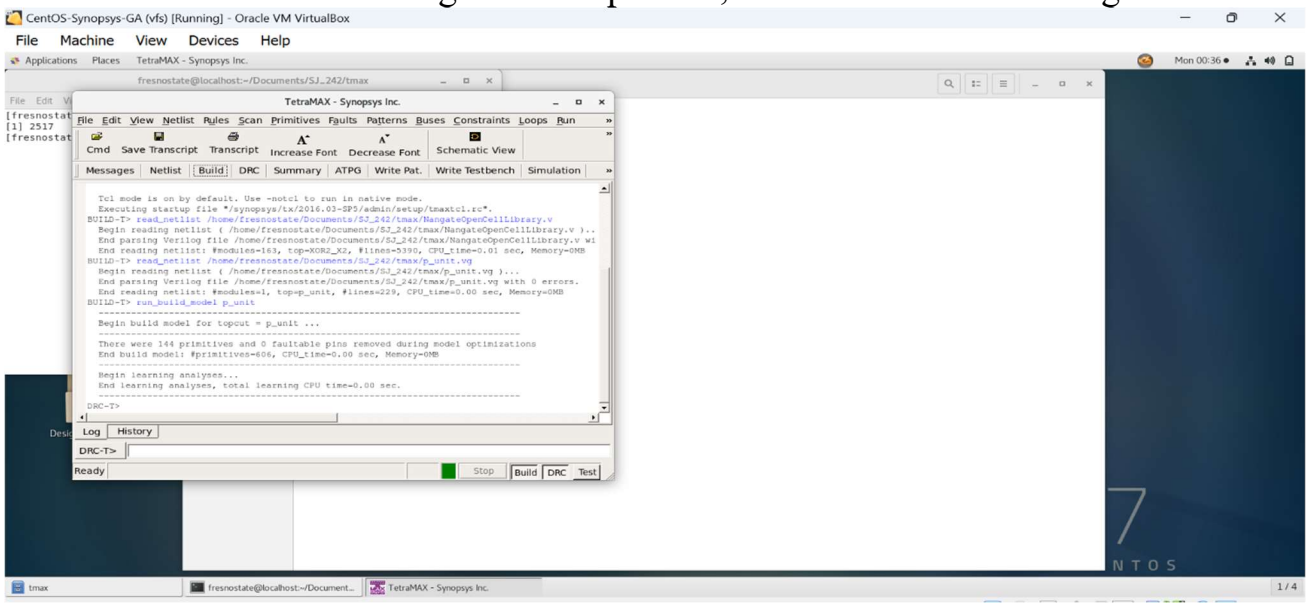


Fig: Transcript displayed when we run build model

11.2.6 Conducting Design Rule Check (DRC) in TetraMax

To ensure that your design conforms to the required design rules before proceeding to test pattern generation, perform a Design Rule Check (DRC) in TetraMax using the following steps:

- **Initiate DRC Process:**

Open the command toolbar in the main window of TetraMax.
Click on the **DRC** button to open the DRC dialog box.

- **Run DRC:**

In the DRC dialog box, typically, no changes are needed for a basic non-sequential circuit without a STIL file. Accept the default settings provided.
Click the **run** tab to start the DRC process. This is shown in the below Figure.

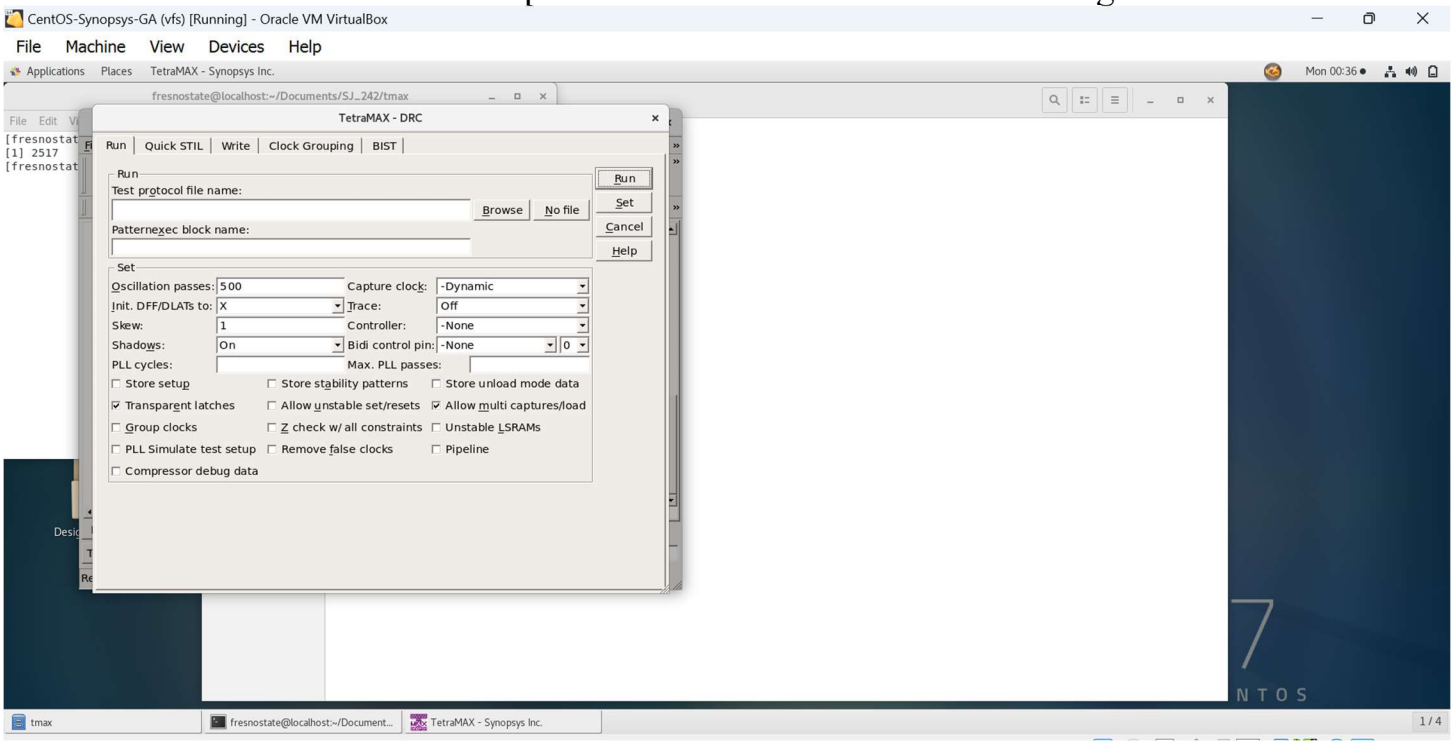


Fig: Run DRC

- **Handle DRC Violations:**

If the DRC identifies any rule breaches, these need to be addressed. Corrections can be made through the Graphic State Viewer (GSV) window.
Continue adjusting your design until all violations are resolved

- **Check for Compliance:**

After correcting violations, the transcript window will display whether there are any remaining DRC violations.
Only proceed to the TEST mode in TetraMax once all DRC violations are cleared, ensuring your design is compliant with all necessary rules.

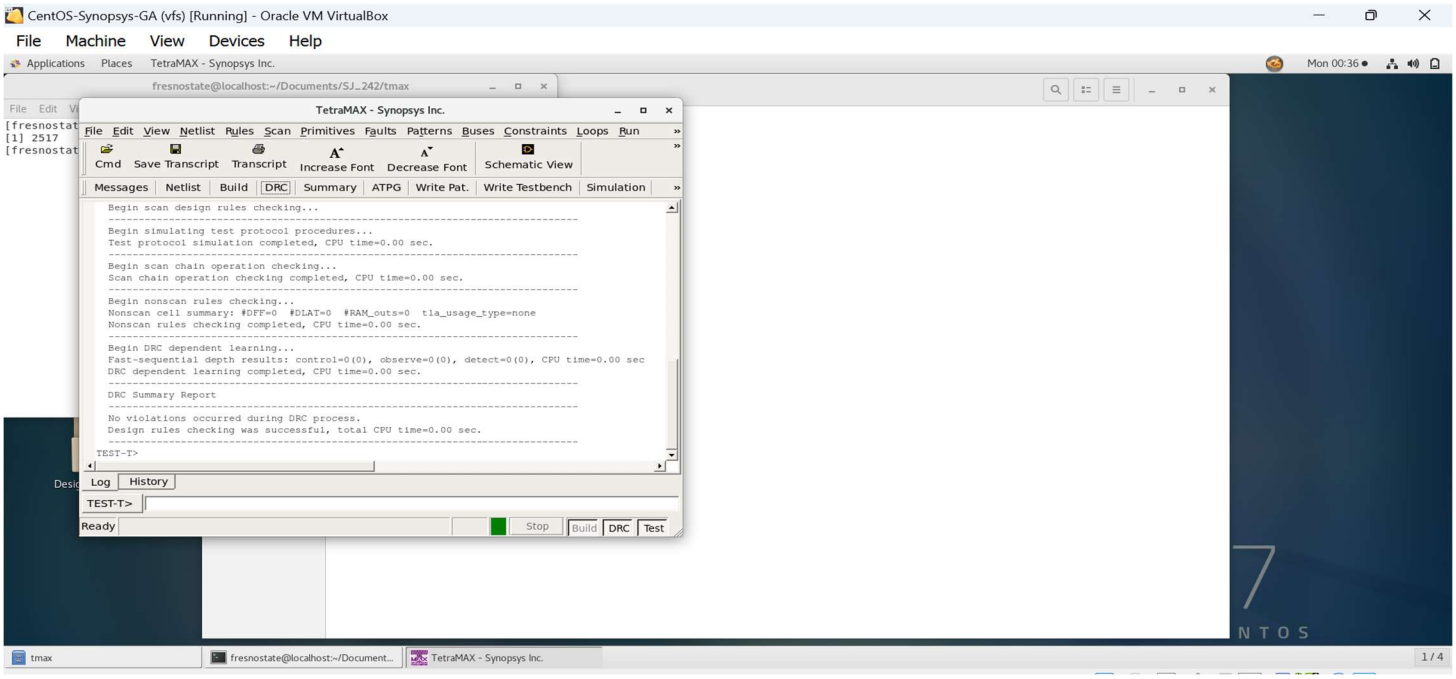


Fig: DRC Transcript

11.2.7: Setting Up and Running ATPG in TetraMax

Automated Test Pattern Generation (ATPG) in TetraMax involves several steps to ensure that faults are accurately detected and tested in your design. Follow these steps to configure and execute ATPG effectively:

- **Accessing ATPG Configuration:**

Click on the **ATPG** option in the TetraMax interface to open the ATPG configuration window.

- **Configuring ATPG Settings:**

Set Coverage Goal: In the ATPG window, specify the fault coverage percentage. For comprehensive testing, set the "coverage %" to 100.

Fault Source: Click on "Add all faults" to include all possible faults in the test. This ensures a thorough examination of the design.
Fault Model: Select the "Stuck fault model" to simulate conditions where signals are incorrectly fixed at high or low states.
ATPG Type: Choose between "Basic scan," "Fast sequential," or "Full sequential," depending on the complexity and requirements of your design. This determines the method and speed of the test pattern generation.

- **Running ATPG:**

After setting all necessary configurations, click the **Auto** button to start the ATPG process. This will automatically generate the test patterns based on the selected settings.

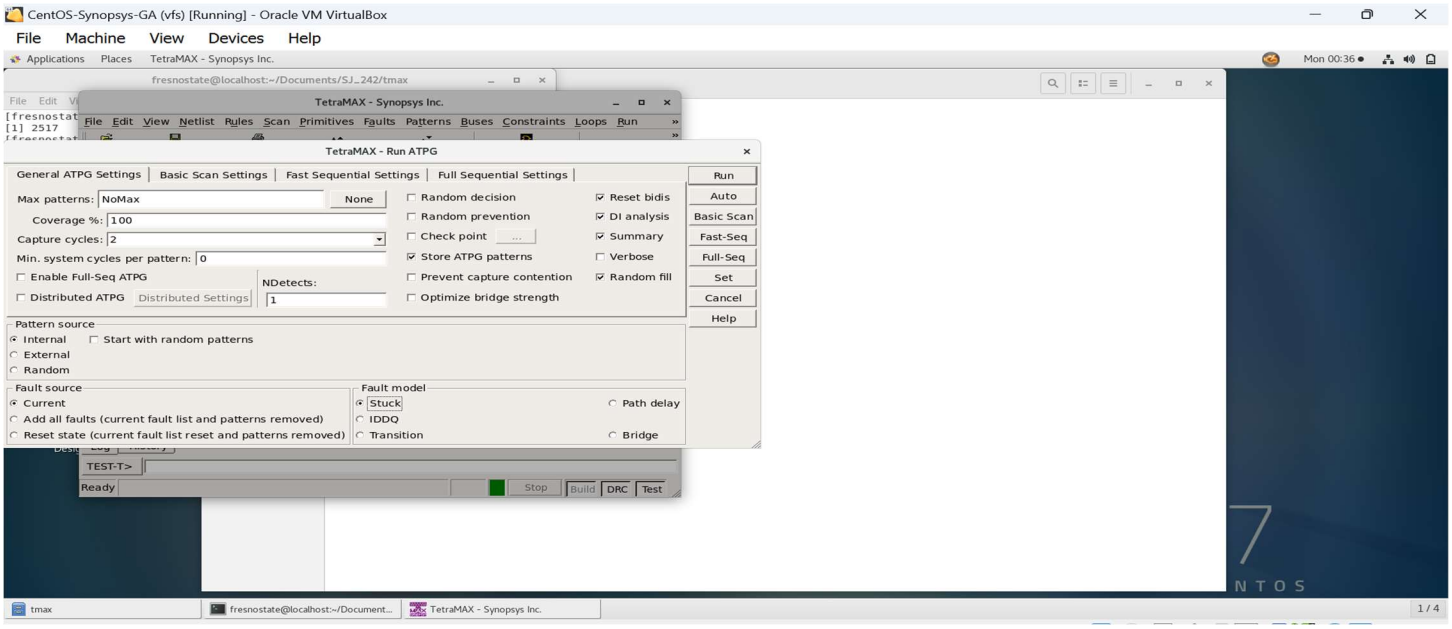


Fig: Run ATPG

- **Observing Results:**

Monitor the progress and results in the transcript window. This window displays critical information including the number of faults detected, the number of test patterns generated, and CPU usage data, as shown in the below figure.

The transcript window provides insights into the efficiency and effectiveness of the generated test patterns and the overall test process.

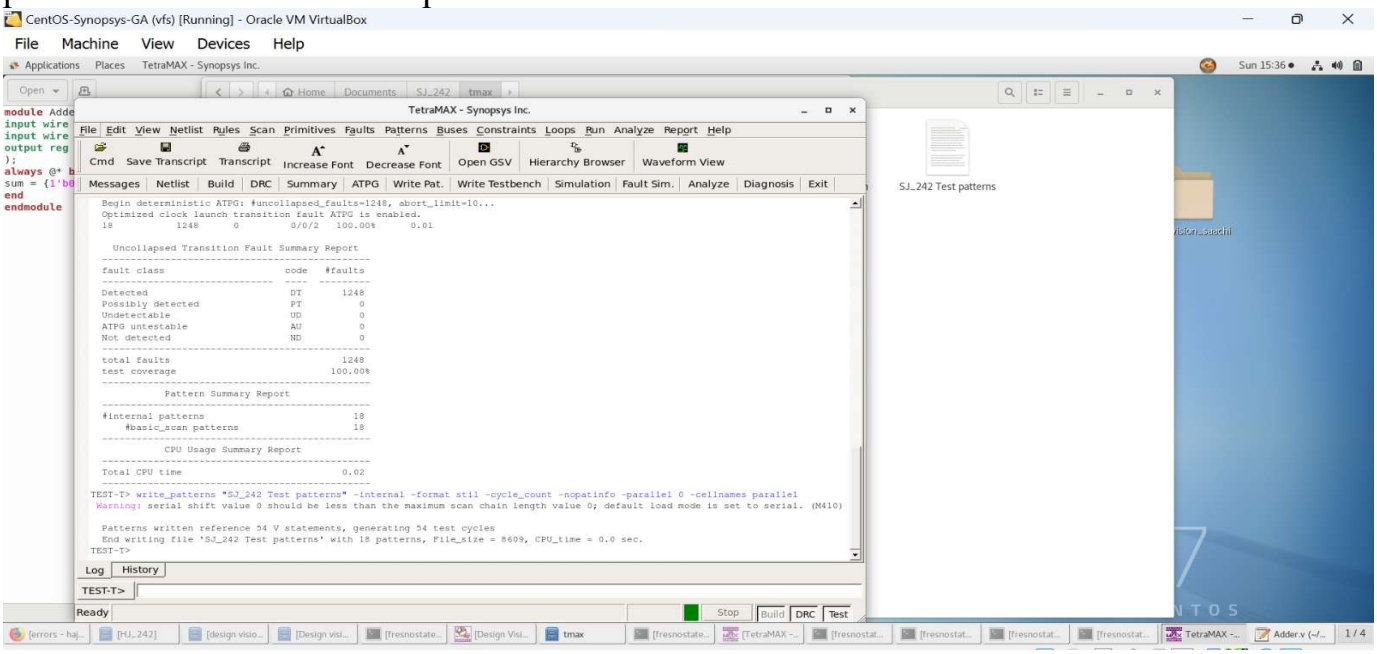


Fig: Transcript of generated patterns

11.2.8 Writing Test Patterns in TetraMax

After generating test patterns using ATPG in TetraMax, the next step is to save these patterns into a file for further analysis or practical testing.

Follow these steps to write the test patterns:

- **Initiate Pattern Writing:**

Click on "Write Patterns" from the command toolbar in the main window of TetraMax. This opens the pattern writing dialog box.

- **Configure Pattern File:**

Pattern File Name: Enter the name of the pattern file where the test patterns will be saved. Choose a name that helps you easily identify the purpose of the file.

Format Selection: Select the format in which the test patterns should be written. Choose a format that matches your testing requirements or the specifications of the testing hardware.

The configuration window, as shown in the below Figure, allows you to set these parameters.

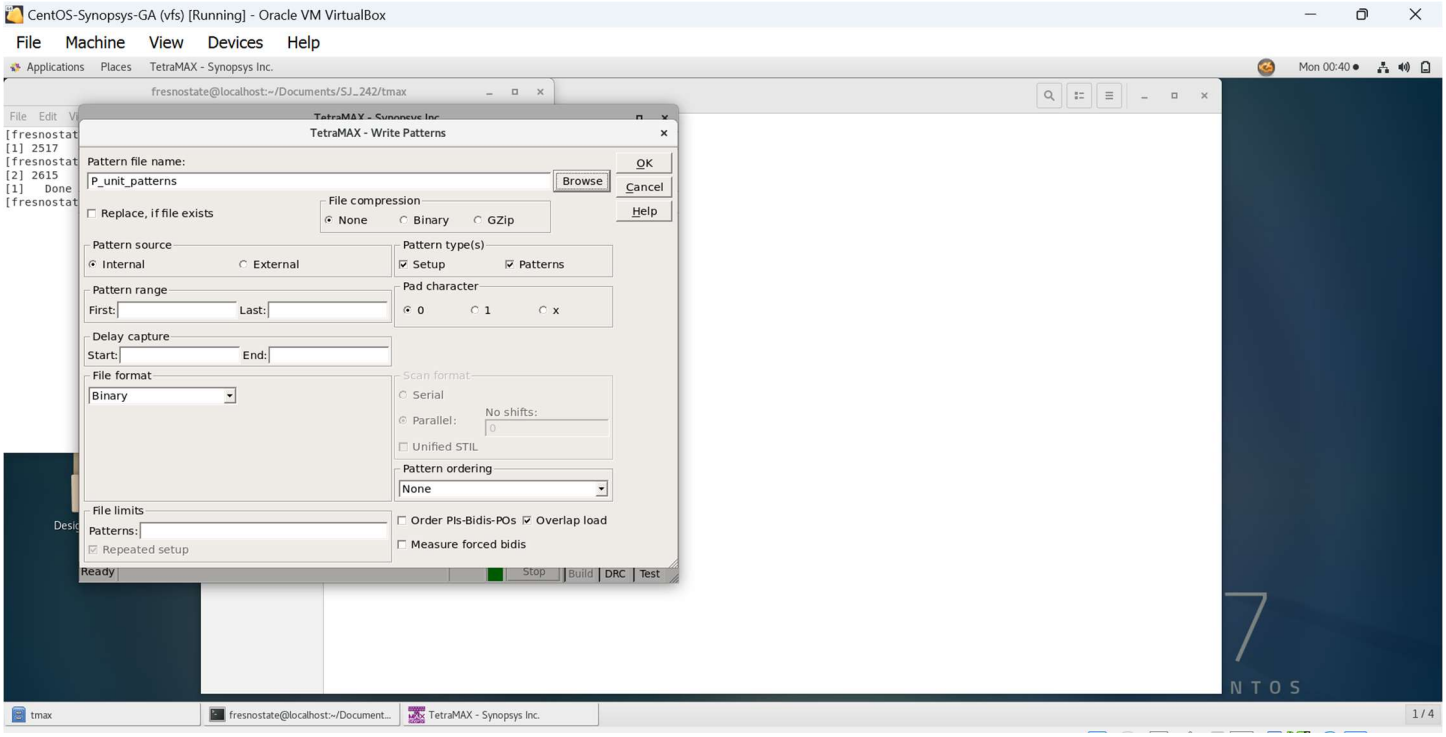


Fig: Configure pattern file.

- **Generate and Save Patterns:**

Click **OK** to confirm the settings and start the process of writing the test patterns to the specified file. The test patterns will be written into the file located in the same folder as your library and netlist files, ensuring all relevant files are organized together.

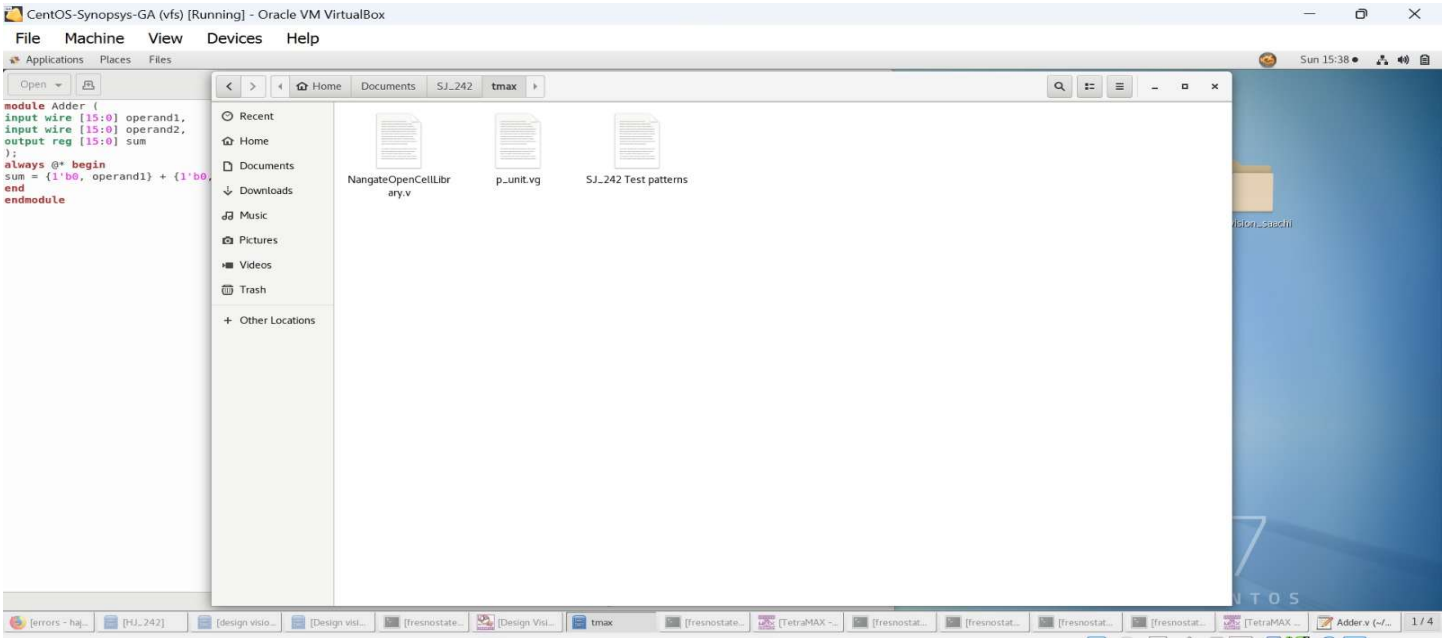


Fig: Pattern file displayed in the folder

- **Review the Output:**

After the test patterns are written, you can view the file to verify the patterns, detected faults, fault coverage, and other relevant data. This helps in assessing the effectiveness of the test patterns in covering the potential faults in the design.

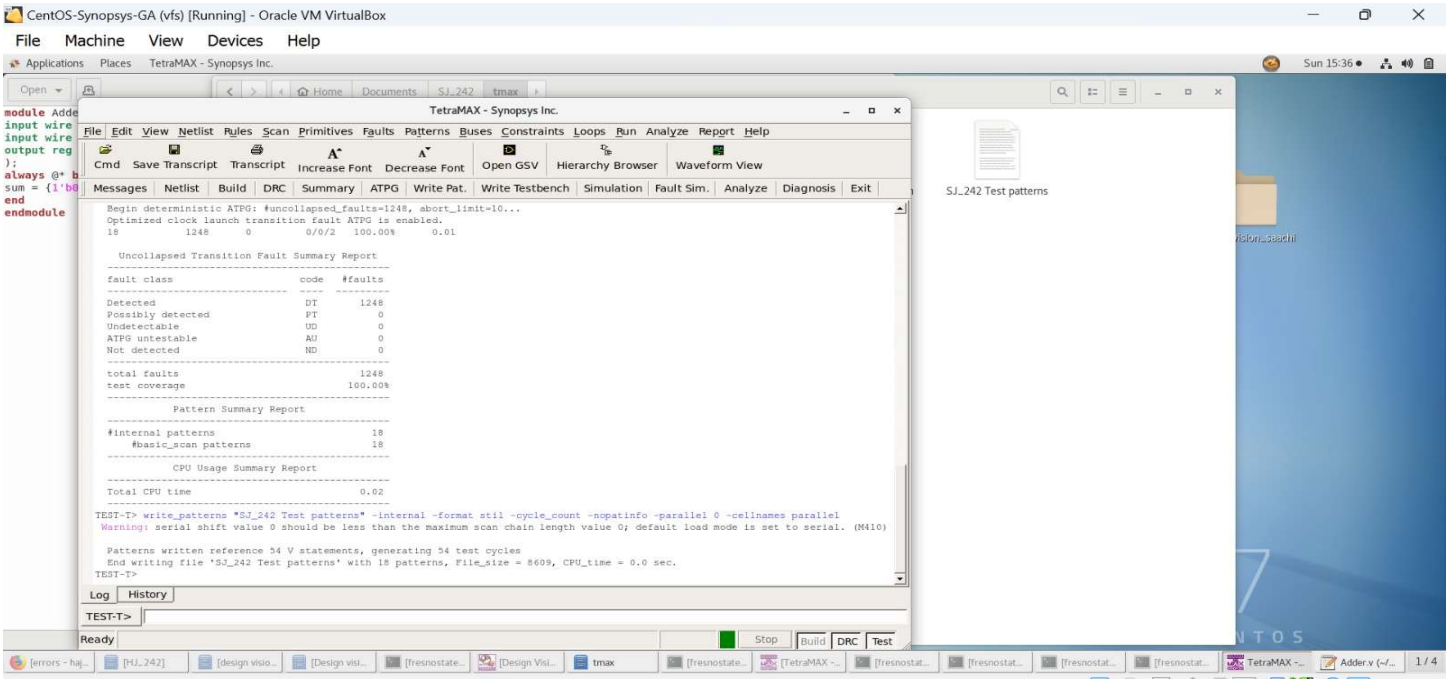


Fig: Review of patterns

11.2.9: Schematic View:

To get the schematic view of our parallel arithmetic unit we should follow the below steps:

1. We should make a new folder where we put our required Verilog codes and required library files.

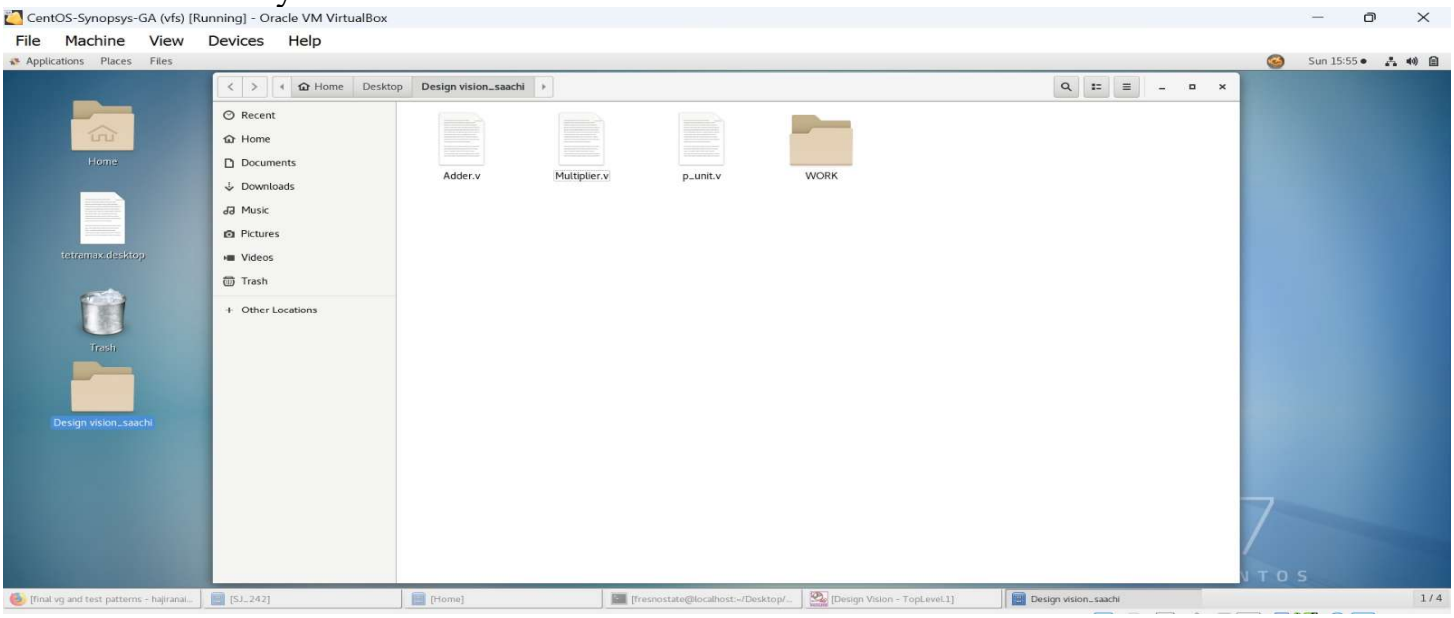


Fig: Create a new folder

2. Now you should open the terminal and type design_vision

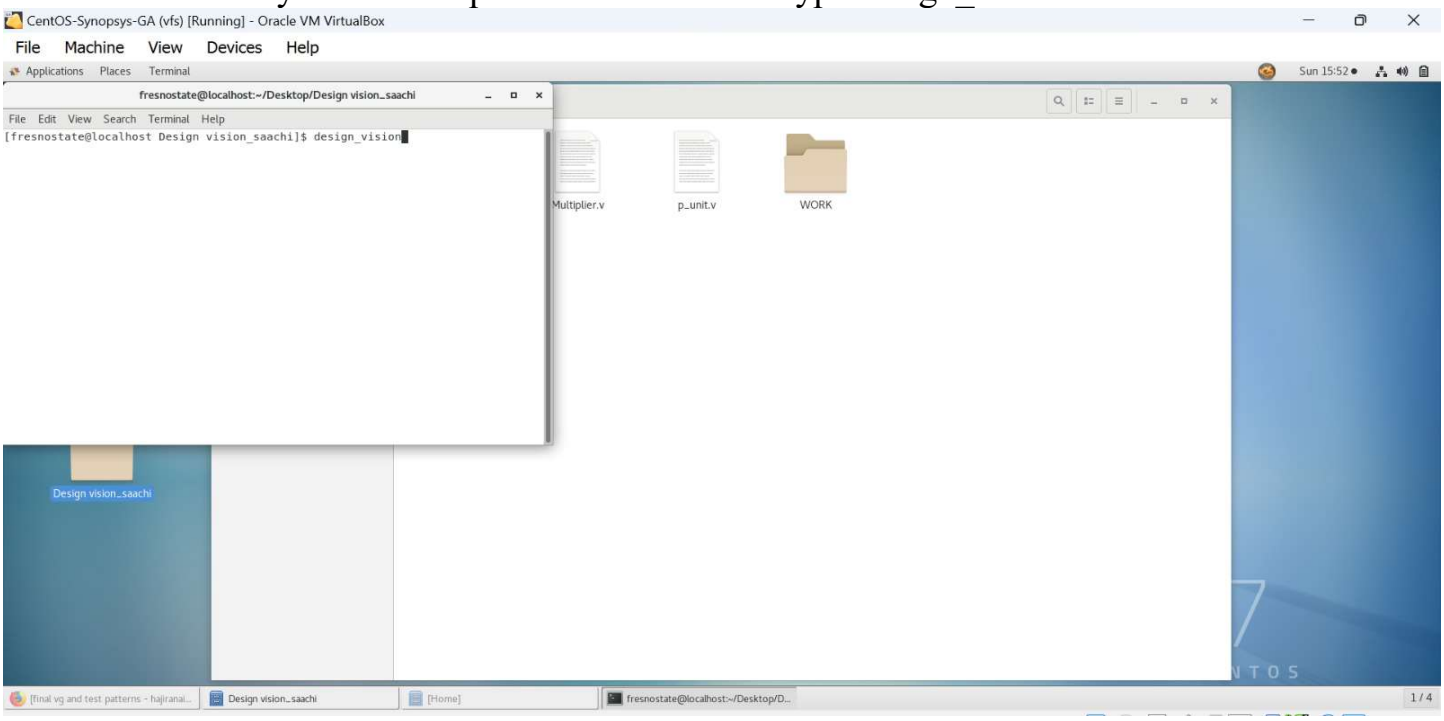


Fig: Type design_vision on terminal

3. Now we should click on **file>setup**

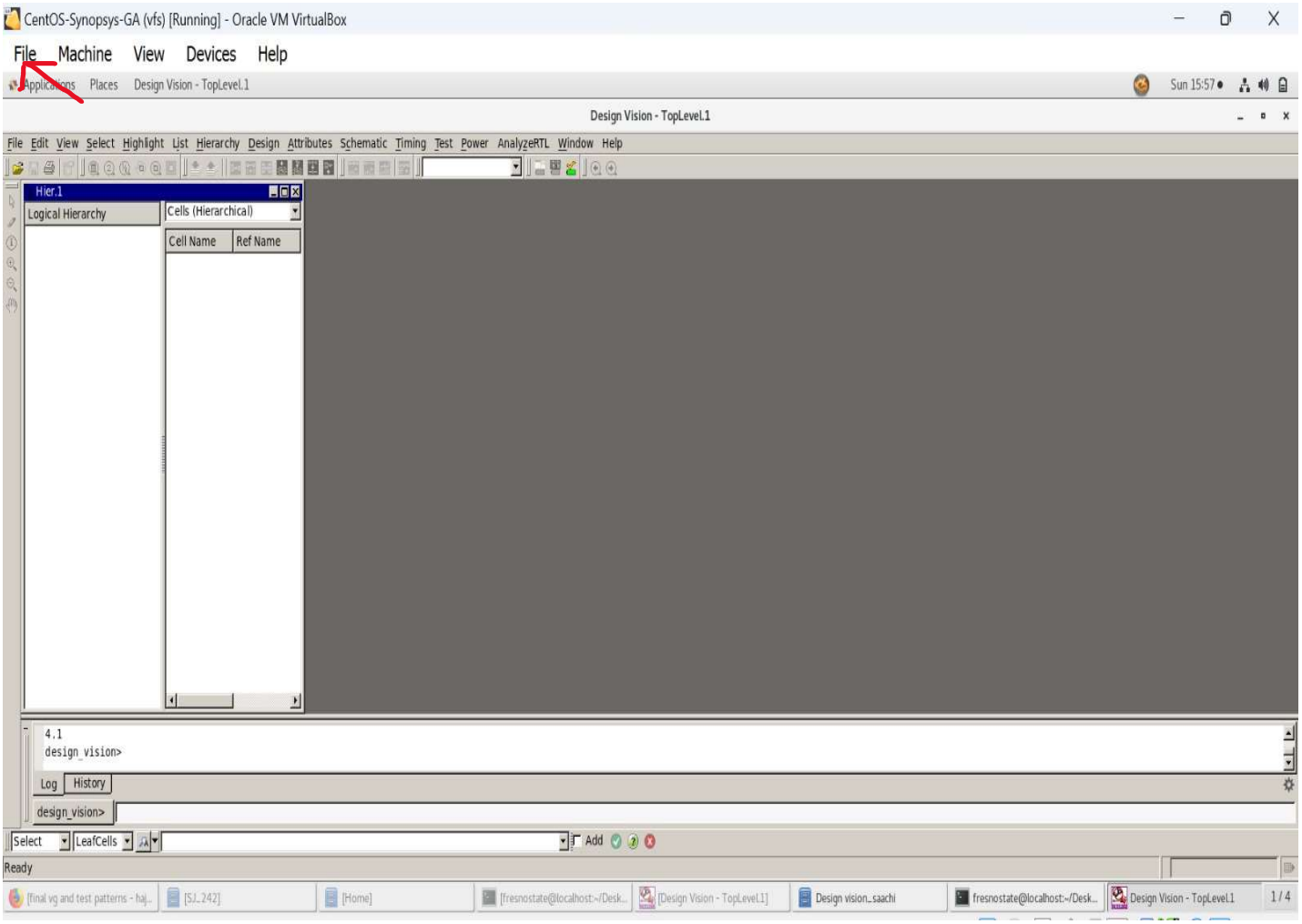


Fig: File setup

4. Setting Up Libraries in the Design Environment

1. **Link Library Setup:**
- Go to the library settings and choose 'Add'.
- Find and select **NangateOpenCellLibrary**, and click 'OK'.

2. **Target Library Configuration:**
- In the library settings, add the **NangateOpenCellLibrary.db** for synthesis targeting.

3. **Symbol Library Setup:'**
- Instead of a **.db** file, choose the **NangateOpenCellLibrary.sdb** file for the symbol library. The **.sdb** file typically contains symbolic representations of the components in the library, useful for schematic generation and visualization.
- Add **NangateOpenCellLibrary.sdb** for the symbol library, ensuring it is used for schematic views.

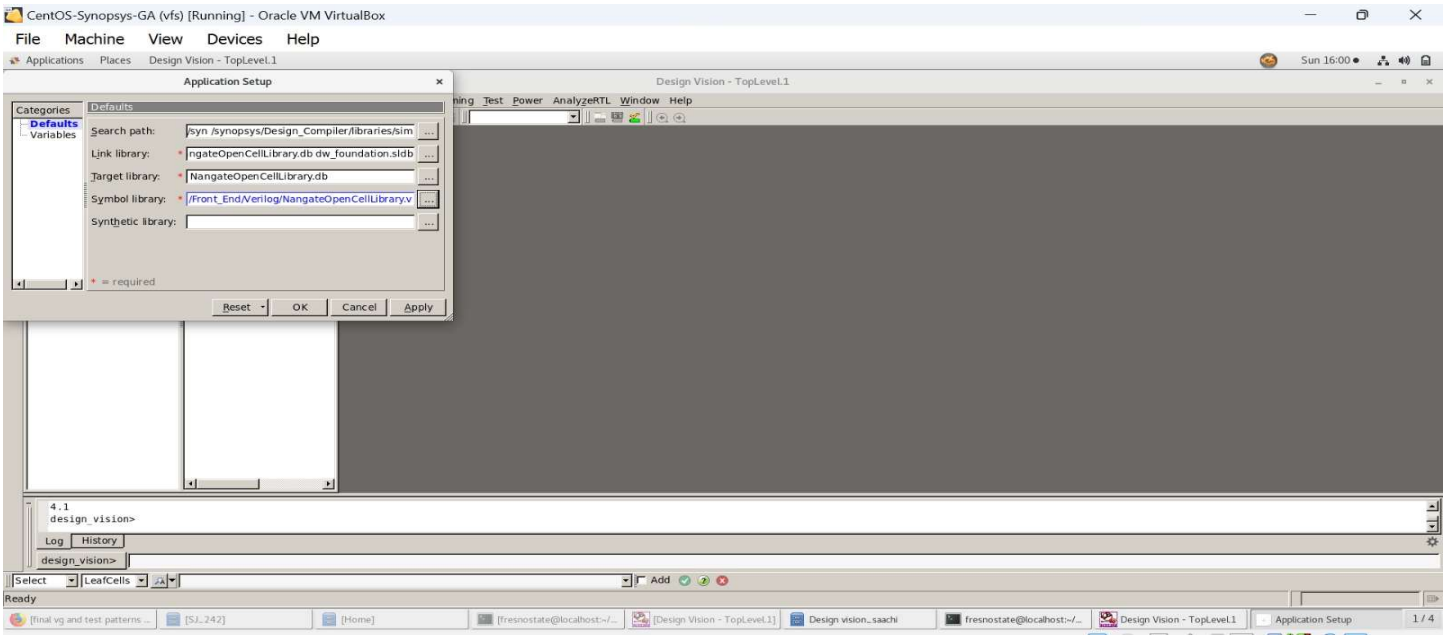


Fig: Setting Up Libraries in the Design Environment

5. To add files for analysis, go to **File** > **Analyze**

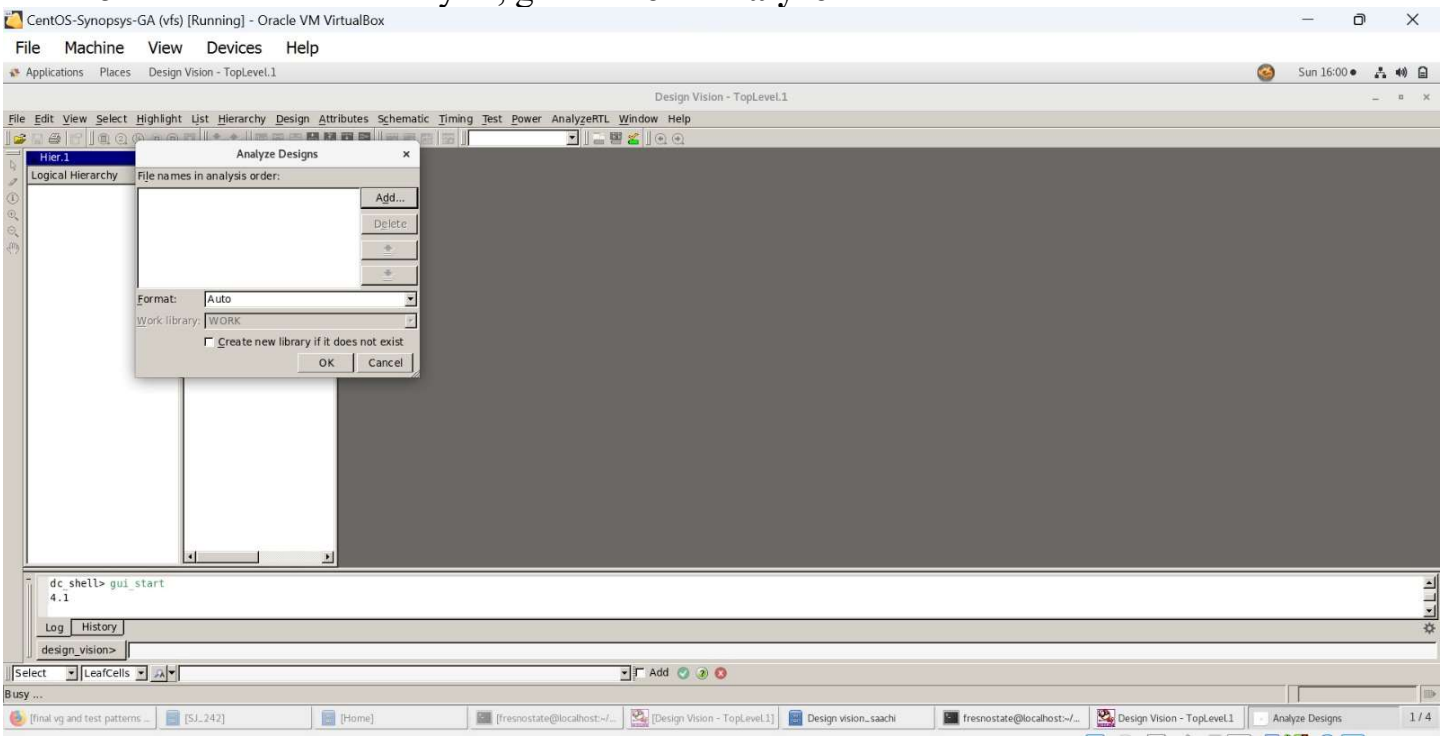


Fig: Add files

6. Click **Add** to include your code files, ensure they are stored in the **WORK** library path, and then click **OK** to load the files successfully.

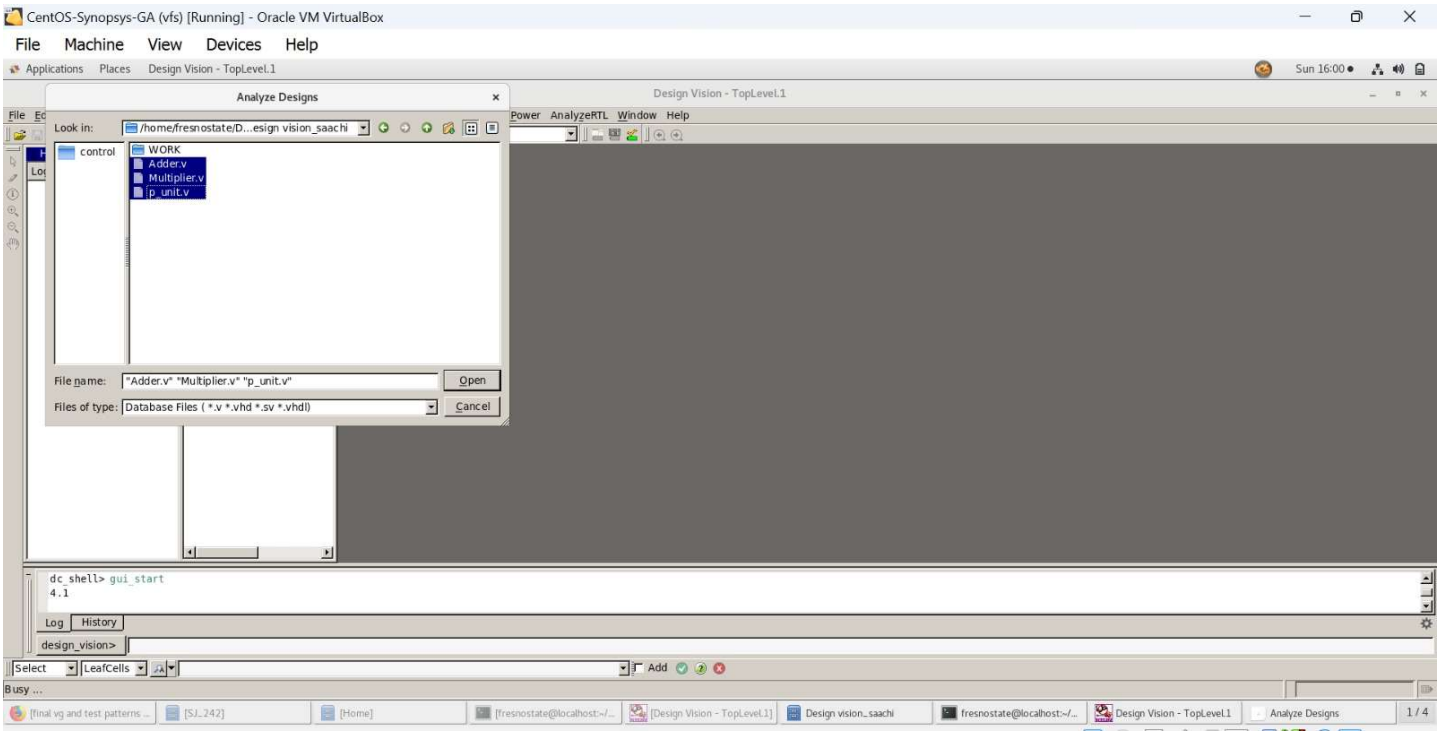


Fig: Add code files

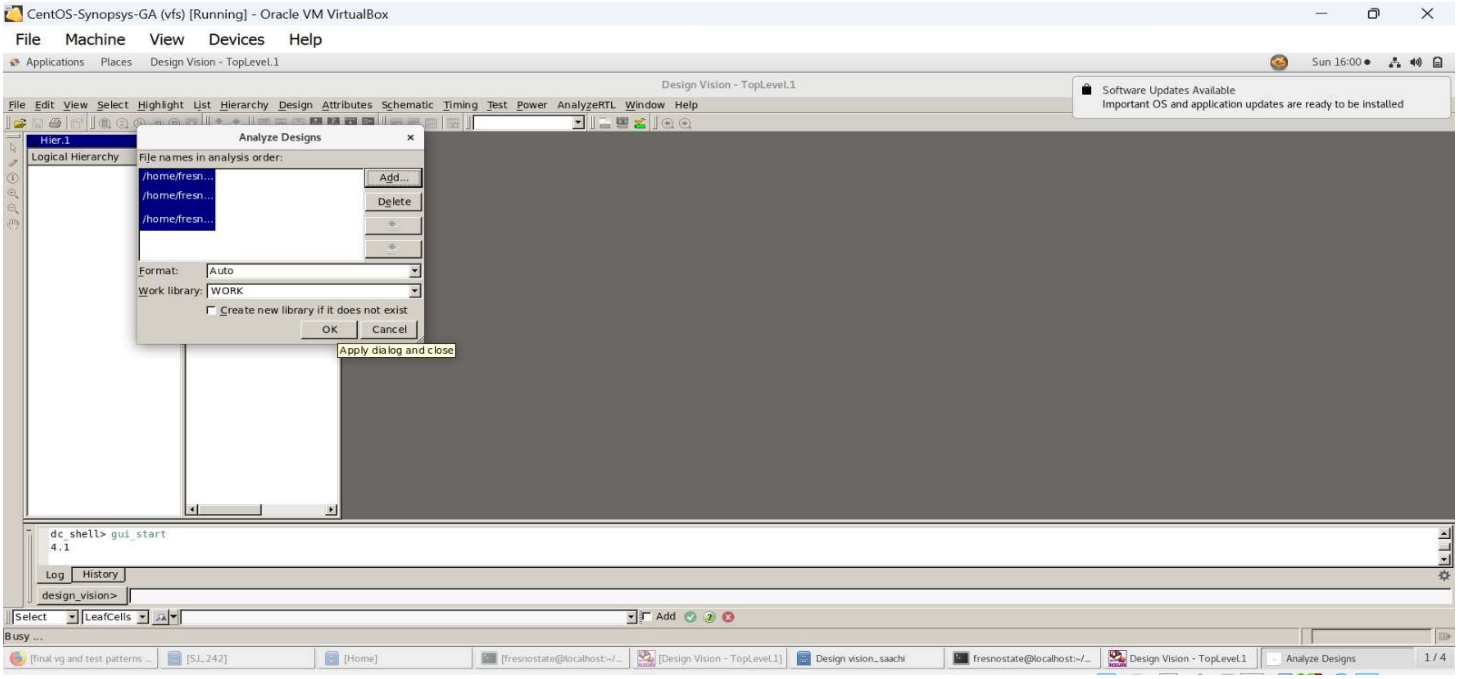


Fig: Click ok and make sure the library name is work.

7. Elaborate the Design:

- Go to **File** > **Elaborate**.
- Ensure the **Work** library file is used.
- Set your top module Verilog code, which is **p_unit.v**, as the design to elaborate.
- Click **OK** to start the elaboration process, which checks if your code is synthesizable.

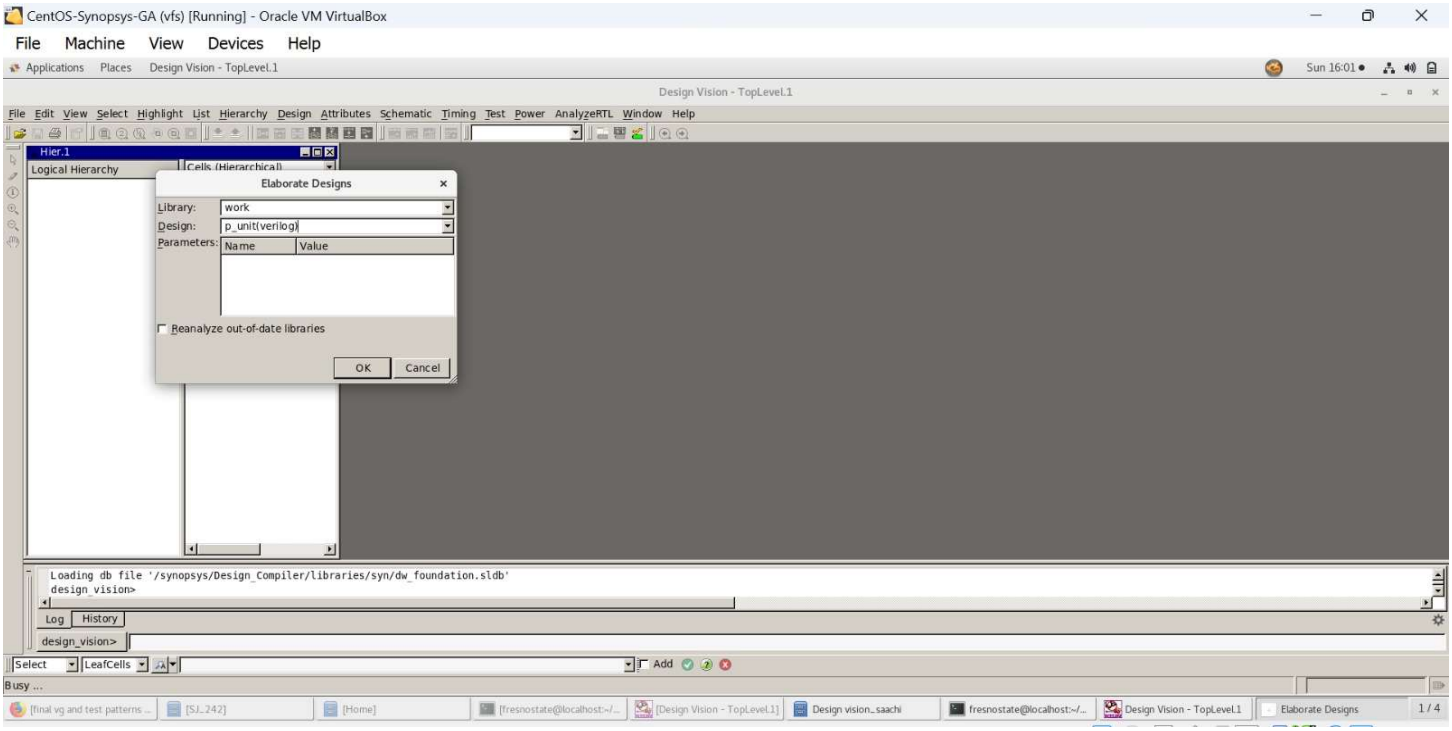


Fig: Elaborate the design

8. After the compilation runs successfully, right-click and select "Schematic View" to visually inspect the schematic diagram of your design.

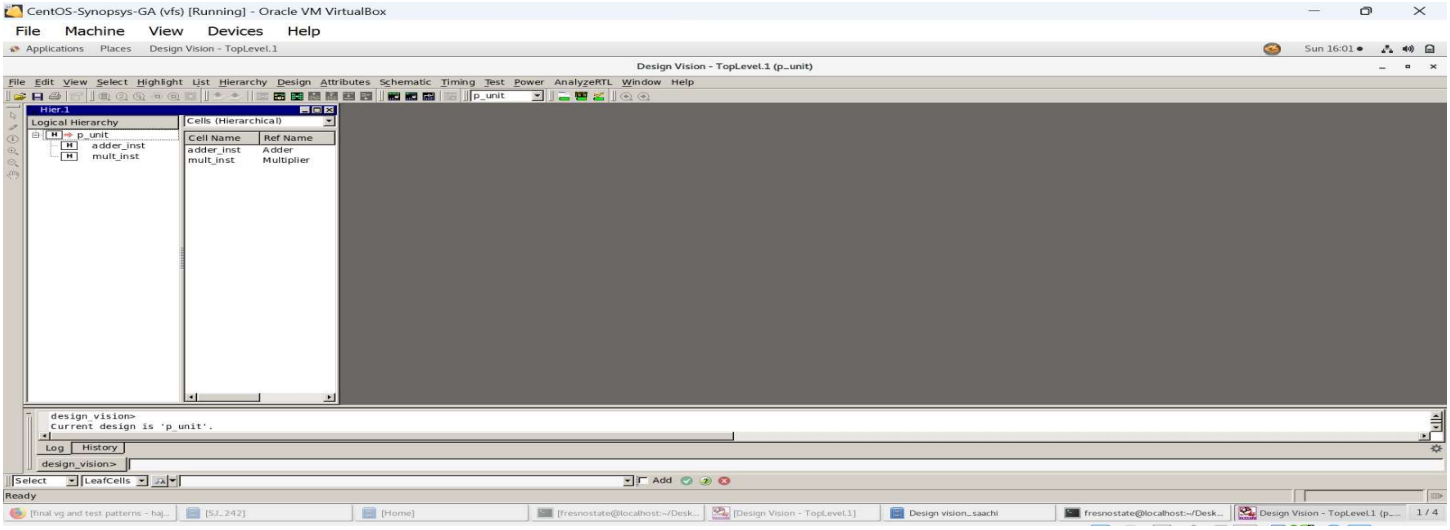


Fig: right-click and select Schematic View

9. Once you open the Schematic View, you'll see a visual representation of your design's schematic.

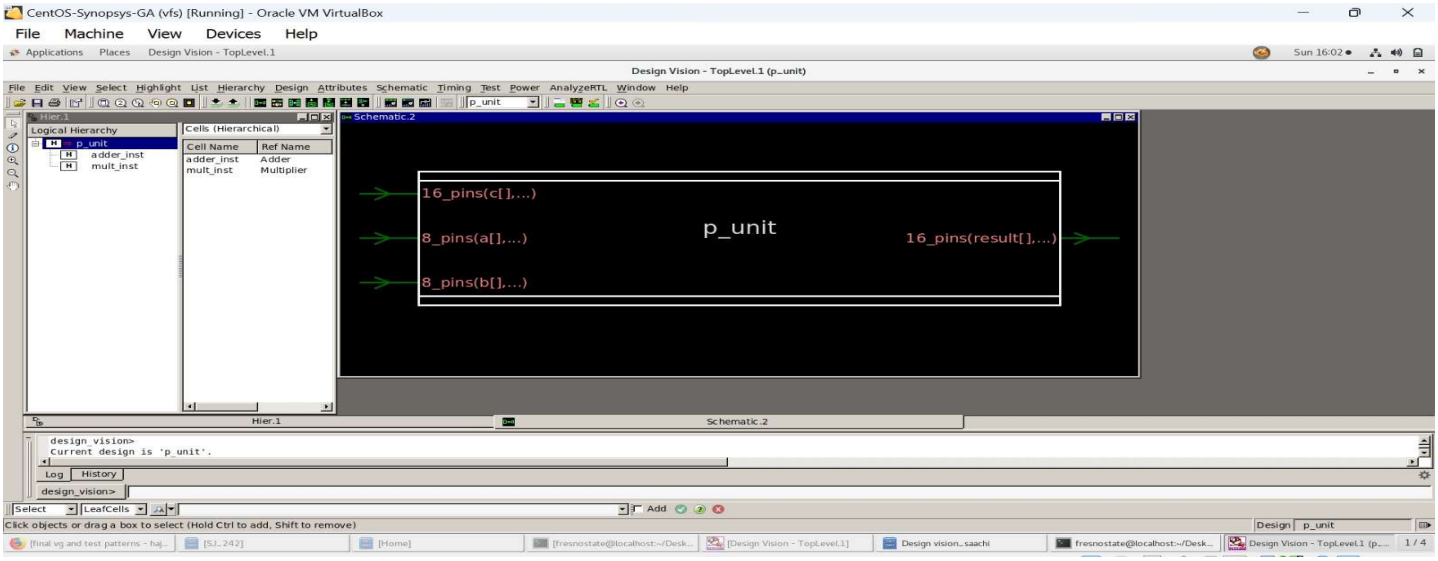


Fig: Schematic

If you double-click on any component or connection within the schematic, it will provide a more detailed view or further information about that specific part of the design. This feature is useful for closely inspecting the architecture and understanding the interconnections and functionality of individual elements within your design.

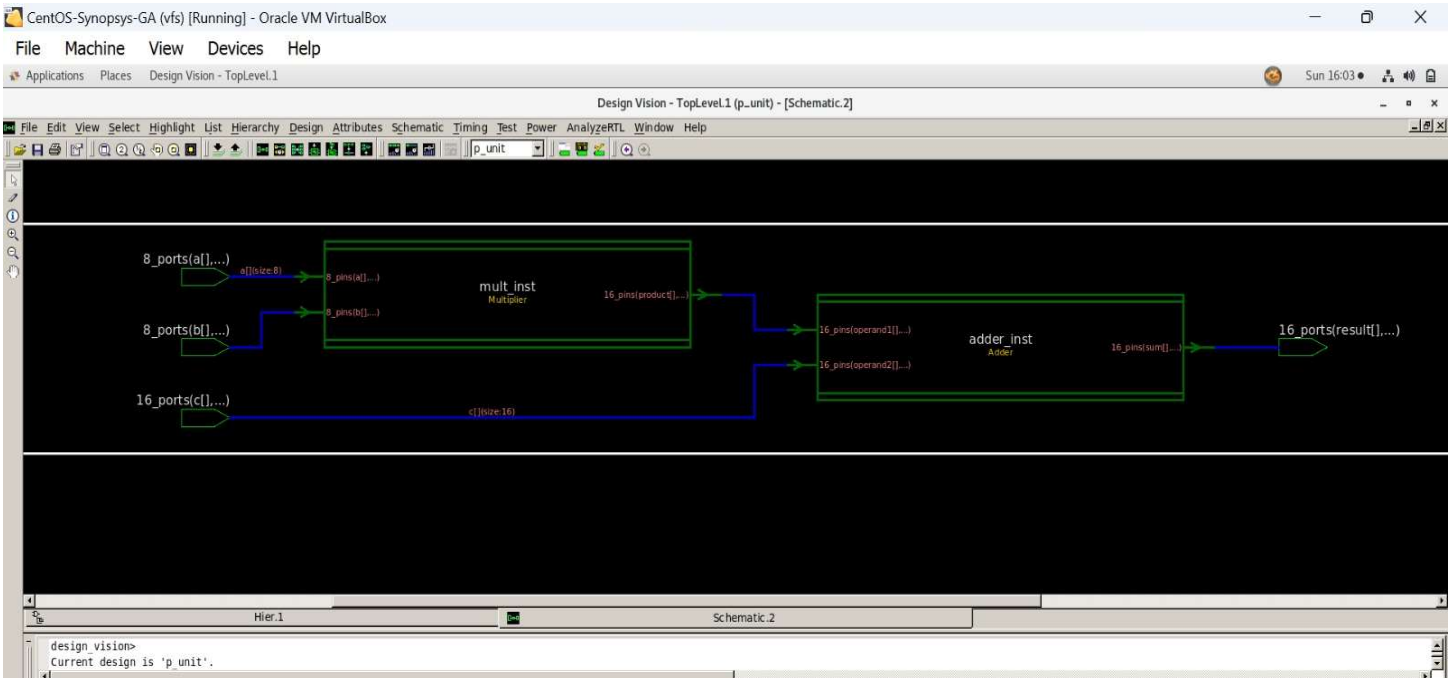


Fig: Detailed Schematic 1

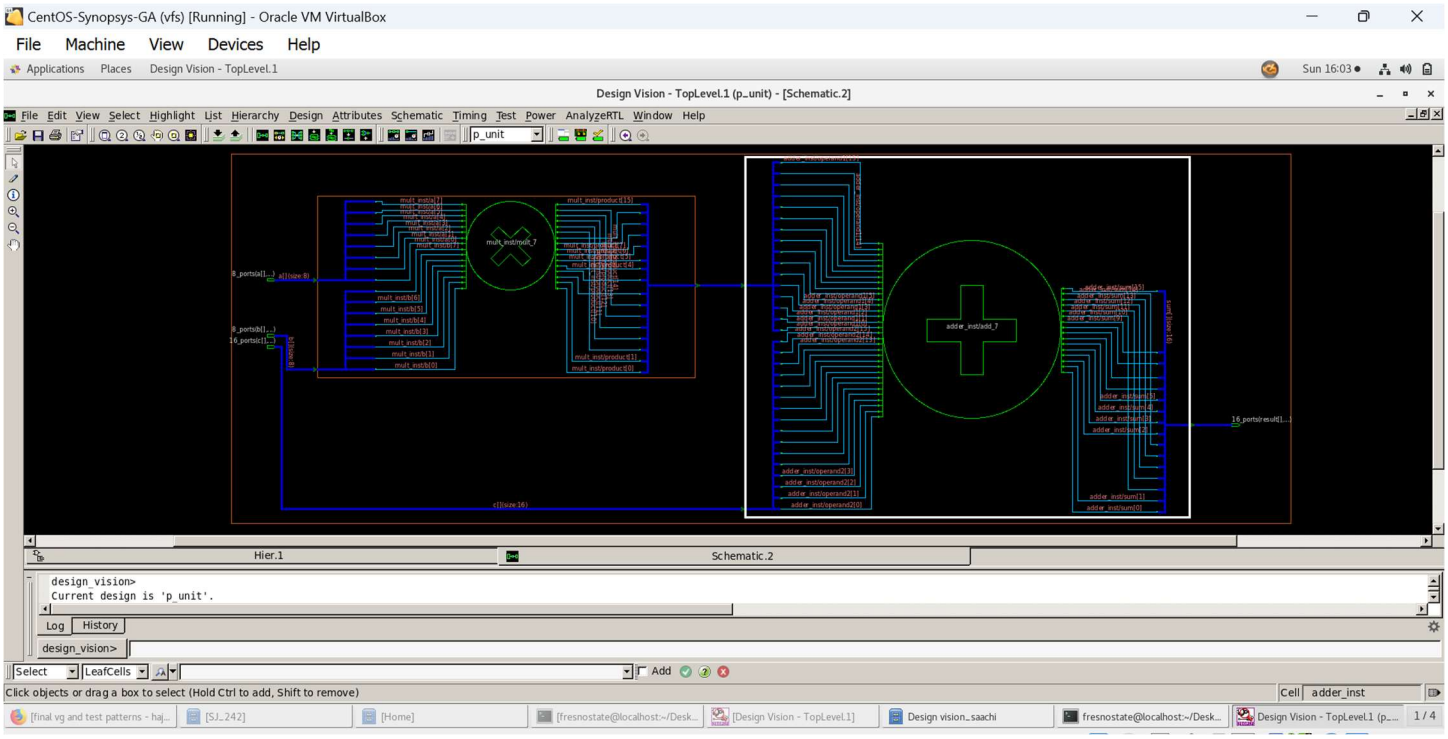


Fig: Detailed Schematic 2

12. Conclusion

The completion of this project marks a significant advancement in the field of digital circuit design, specifically in the implementation and verification of a parallel arithmetic unit using combinational logic. Throughout the project, we successfully designed, simulated, synthesized, and tested a parallel arithmetic unit capable of performing high-speed arithmetic operations efficiently.

Key achievements of the project include:

- Design and Implementation: We developed a highly efficient parallel architecture for performing simultaneous arithmetic operations using Verilog.
- Simulation and Verification: Through rigorous simulations in ModelSim, the functionality and performance of each component were validated, ensuring that the design met all specified requirements before and after synthesis.
- Synthesis Optimization: The Synopsys Design Compiler was utilized to convert RTL designs into optimized gate-level netlists, effectively balancing area, speed, and power consumption.
- Test Pattern Generation and Fault Analysis: Using TetraMax, we generated and analyzed test patterns that ensured the design was robust against potential manufacturing defects, achieving comprehensive fault coverage.

13. Appendix

Verilog Code

Adder Module:

```
module Adder (
input wire [15:0] operand1,
input wire [15:0] operand2,
output reg [15:0] sum
);
always @* begin
sum = {1'b0, operand1} + {1'b0,operand2};
end
endmodule
```

Multiplier Module:

```
module Multiplier (
input wire [7:0] a,
input wire [7:0] b,
output reg [15:0] product
);
always @* begin
product = a * b;
end
endmodule
```

PAU unit module

```
module p_unit (
  input [7:0] a,
  input [7:0] b,
  input [7:0] c,
  output reg [15:0] result
);
wire [15:0] product;
wire [15:0] sum;
```

```
Multiplier mult_inst(.a(a), .b(b), .product(product));
Adder adder_inst(.operand1(product), .operand2(c), .sum(sum));

always @* begin
  result = sum;
end
endmodule
```

Testbench

```
module tb_p_unit
reg [7:0] a;
reg [7:0] b;
reg [7:0] c;
wire [15:0] result;

PAU_unit dut(.a(a), .b(b), .c(c), .result(result));

initial begin
  a = 8'b10101010; b = 8'b01010101; c = 8'b00001111;
  #10;
  a = 8'b11111111; b = 8'b00001111; c = 8'b00001111;
  #10;
  $finish;
end
endmodule
```

Synthesized GATE NETLIST

```
////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version : O-2018.06-SP3
// Date : Sun May 12 15:23:24 2024
////////////////////////////////////////////////////////////
module p_unit ( a, b, c, result );
input [7:0] a;
input [7:0] b;
input [15:0] c;
output [15:0] result;
wire \product[9] , \product[8] , \product[7] , \product[6] , \product[5] ,
\product[4] , \product[3] , \product[2] , \product[1] , \product[15] ,
\product[14] , \product[13] , \product[12] , \product[11] ,
\product[10] , \product[0] , n11, n177, n176, n175, n174, n173, n172,
n171, n170, n169, n168, n167, n166, n165, n164, n163, n162, n161,
n160, n159, n158, n157, n156, n155, n154, n153, n152, n151, n150,
n149, n148, n147, n146, n145, n144, n143, n142, n141, n140, n139,
n138, n137, n136, n135, n134, n133, n132, n131, n130, n129, n128,
n127, n126, n125, n124, n123, n122, n121, n120, n119, n118, n117,
n116, n115, n114, n113, n112, n1111, n110, n109, n108, n107, n106,
n105, n104, n103, n102, n101, n100, n99, n98, n97, n96, n95, n94, n93,
n92, n91, n90, n89, n88, n87, n86, n85, n84, n83, n82, n81, n80, n79,
n78, n77, n76, n75, n74, n73, n72, n71, n70, n69, n68, n67, n66, n65,
n64, n63, n62, n61, n60, n59, n58, n57, n56, n55, n54, n53, n52, n51,
n50, n49, n48, n47, n46, n45, n44, n43, n42, n41, n40, n39, n38, n37,
n36, n35, n34, n33, n32, n31, n30, n29, n28, n27, n26, n25, n24, n23,
n22, n21, n20, n19, n18, n17, n16, n15, n14, n13, n12, n111, n10, n9,
n8, n7, n6, n5, n4, n3, n2;
wire [15:1] carry;
XOR2_X1 U2 ( .A(c[0]), .B(\product[0] ), .Z(result[0]) );
AND2_X1 U11 ( .A1(c[0]), .A2(\product[0] ), .ZN(n11) );
FA_X1 U1_1 ( .A(\product[1] ), .B(c[1]), .CI(n11), .CO(carry[2]), .S(
result[1]) );
FA_X1 U1_2 ( .A(\product[2] ), .B(c[2]), .CI(carry[2]), .CO(carry[3]), .S(
result[2]) );
FA_X1 U1_3 ( .A(\product[3] ), .B(c[3]), .CI(carry[3]), .CO(carry[4]), .S(
result[3]) );
FA_X1 U1_4 ( .A(\product[4] ), .B(c[4]), .CI(carry[4]), .CO(carry[5]), .S(
result[4]) );
FA_X1 U1_5 ( .A(\product[5] ), .B(c[5]), .CI(carry[5]), .CO(carry[6]), .S(
result[5]) );
```

```
FA_X1 U1_6 ( .A(\product[6] ), .B(c[6]), .CI(carry[6]), .CO(carry[7]), .S(
result[6]) );
FA_X1 U1_7 ( .A(\product[7] ), .B(c[7]), .CI(carry[7]), .CO(carry[8]), .S(
result[7]) );
FA_X1 U1_8 ( .A(\product[8] ), .B(c[8]), .CI(carry[8]), .CO(carry[9]), .S(
result[8]) );
FA_X1 U1_9 ( .A(\product[9] ), .B(c[9]), .CI(carry[9]), .CO(carry[10]), .S(
result[9]) );
FA_X1 U1_10 ( .A(\product[10] ), .B(c[10]), .CI(carry[10]), .CO(carry[11]),
.S(result[10]) );
FA_X1 U1_11 ( .A(\product[11] ), .B(c[11]), .CI(carry[11]), .CO(carry[12]),
.S(result[11]) );
FA_X1 U1_12 ( .A(\product[12] ), .B(c[12]), .CI(carry[12]), .CO(carry[13]),
.S(result[12]) );
FA_X1 U1_13 ( .A(\product[13] ), .B(c[13]), .CI(carry[13]), .CO(carry[14]),
.S(result[13]) );
FA_X1 U1_14 ( .A(\product[14] ), .B(c[14]), .CI(carry[14]), .CO(carry[15]),
.S(result[14]) );
FA_X1 U1_15 ( .A(\product[15] ), .B(c[15]), .CI(carry[15]), .S(result[15])
);
INV_X1 U155 ( .A(a[7]), .ZN(n170) );
INV_X1 U154 ( .A(a[6]), .ZN(n171) );
INV_X1 U153 ( .A(a[5]), .ZN(n172) );
INV_X1 U152 ( .A(a[4]), .ZN(n173) );
INV_X1 U151 ( .A(a[3]), .ZN(n174) );
INV_X1 U150 ( .A(a[2]), .ZN(n175) );
INV_X1 U149 ( .A(a[1]), .ZN(n176) );
INV_X1 U148 ( .A(a[0]), .ZN(n177) );
INV_X1 U147 ( .A(b[7]), .ZN(n162) );
INV_X1 U146 ( .A(b[6]), .ZN(n163) );
INV_X1 U145 ( .A(b[5]), .ZN(n164) );
INV_X1 U144 ( .A(b[4]), .ZN(n165) );
INV_X1 U143 ( .A(b[3]), .ZN(n166) );
INV_X1 U142 ( .A(b[2]), .ZN(n167) );
INV_X1 U141 ( .A(b[1]), .ZN(n168) );
INV_X1 U140 ( .A(b[0]), .ZN(n169) );
NOR2_X1 U121 ( .A1(n169), .A2(n177), .ZN(\product[0] ) );
NOR2_X1 U120 ( .A1(n168), .A2(n177), .ZN(n161) );
NOR2_X1 U119 ( .A1(n167), .A2(n177), .ZN(n160) );
NOR2_X1 U118 ( .A1(n166), .A2(n177), .ZN(n159) );
NOR2_X1 U117 ( .A1(n165), .A2(n177), .ZN(n158) );
NOR2_X1 U116 ( .A1(n164), .A2(n177), .ZN(n157) );
NOR2_X1 U115 ( .A1(n163), .A2(n177), .ZN(n156) );
```

```
NOR2_X1 U114 ( .A1(n162), .A2(n177), .ZN(n155) );
NOR2_X1 U113 ( .A1(n169), .A2(n176), .ZN(n154) );
NOR2_X1 U112 ( .A1(n168), .A2(n176), .ZN(n153) );
NOR2_X1 U1111 ( .A1(n167), .A2(n176), .ZN(n152) );
NOR2_X1 U110 ( .A1(n166), .A2(n176), .ZN(n151) );
NOR2_X1 U109 ( .A1(n165), .A2(n176), .ZN(n150) );
NOR2_X1 U108 ( .A1(n164), .A2(n176), .ZN(n149) );
NOR2_X1 U107 ( .A1(n163), .A2(n176), .ZN(n148) );
NOR2_X1 U106 ( .A1(n162), .A2(n176), .ZN(n147) );
NOR2_X1 U105 ( .A1(n169), .A2(n175), .ZN(n146) );
NOR2_X1 U104 ( .A1(n168), .A2(n175), .ZN(n145) );
NOR2_X1 U103 ( .A1(n167), .A2(n175), .ZN(n144) );
NOR2_X1 U102 ( .A1(n166), .A2(n175), .ZN(n143) );
NOR2_X1 U101 ( .A1(n165), .A2(n175), .ZN(n142) );
NOR2_X1 U100 ( .A1(n164), .A2(n175), .ZN(n141) );
NOR2_X1 U99 ( .A1(n163), .A2(n175), .ZN(n140) );
NOR2_X1 U98 ( .A1(n162), .A2(n175), .ZN(n139) );
NOR2_X1 U97 ( .A1(n169), .A2(n174), .ZN(n138) );
NOR2_X1 U96 ( .A1(n168), .A2(n174), .ZN(n137) );
NOR2_X1 U95 ( .A1(n167), .A2(n174), .ZN(n136) );
NOR2_X1 U94 ( .A1(n166), .A2(n174), .ZN(n135) );
NOR2_X1 U93 ( .A1(n165), .A2(n174), .ZN(n134) );
NOR2_X1 U92 ( .A1(n164), .A2(n174), .ZN(n133) );
NOR2_X1 U91 ( .A1(n163), .A2(n174), .ZN(n132) );
NOR2_X1 U90 ( .A1(n162), .A2(n174), .ZN(n131) );
NOR2_X1 U89 ( .A1(n169), .A2(n173), .ZN(n130) );
NOR2_X1 U88 ( .A1(n168), .A2(n173), .ZN(n129) );
NOR2_X1 U87 ( .A1(n167), .A2(n173), .ZN(n128) );
NOR2_X1 U86 ( .A1(n166), .A2(n173), .ZN(n127) );
NOR2_X1 U85 ( .A1(n165), .A2(n173), .ZN(n126) );
NOR2_X1 U84 ( .A1(n164), .A2(n173), .ZN(n125) );
NOR2_X1 U83 ( .A1(n163), .A2(n173), .ZN(n124) );
NOR2_X1 U82 ( .A1(n162), .A2(n173), .ZN(n123) );
NOR2_X1 U81 ( .A1(n169), .A2(n172), .ZN(n122) );
NOR2_X1 U80 ( .A1(n168), .A2(n172), .ZN(n121) );
NOR2_X1 U79 ( .A1(n167), .A2(n172), .ZN(n120) );
NOR2_X1 U78 ( .A1(n166), .A2(n172), .ZN(n119) );
NOR2_X1 U77 ( .A1(n165), .A2(n172), .ZN(n118) );
NOR2_X1 U76 ( .A1(n164), .A2(n172), .ZN(n117) );
NOR2_X1 U75 ( .A1(n163), .A2(n172), .ZN(n116) );
NOR2_X1 U74 ( .A1(n162), .A2(n172), .ZN(n115) );
NOR2_X1 U73 ( .A1(n169), .A2(n171), .ZN(n114) );
NOR2_X1 U72 ( .A1(n168), .A2(n171), .ZN(n113) );
```

```
NOR2_X1 U71 ( .A1(n167), .A2(n171), .ZN(n112) );
NOR2_X1 U70 ( .A1(n166), .A2(n171), .ZN(n1111) );
NOR2_X1 U69 ( .A1(n165), .A2(n171), .ZN(n110) );
NOR2_X1 U68 ( .A1(n164), .A2(n171), .ZN(n109) );
NOR2_X1 U67 ( .A1(n163), .A2(n171), .ZN(n108) );
NOR2_X1 U66 ( .A1(n162), .A2(n171), .ZN(n107) );
NOR2_X1 U65 ( .A1(n169), .A2(n170), .ZN(n106) );
NOR2_X1 U64 ( .A1(n168), .A2(n170), .ZN(n105) );
NOR2_X1 U63 ( .A1(n167), .A2(n170), .ZN(n104) );
NOR2_X1 U62 ( .A1(n166), .A2(n170), .ZN(n103) );
NOR2_X1 U61 ( .A1(n165), .A2(n170), .ZN(n102) );
NOR2_X1 U60 ( .A1(n164), .A2(n170), .ZN(n101) );
NOR2_X1 U59 ( .A1(n163), .A2(n170), .ZN(n100) );
NOR2_X1 U58 ( .A1(n162), .A2(n170), .ZN(n99) );
HA_X1 U57 ( .A(n153), .B(n160), .CO(n97), .S(n98) );
HA_X1 U56 ( .A(n138), .B(n145), .CO(n95), .S(n96) );
FA_X1 U55 ( .A(n152), .B(n159), .CI(n97), .CO(n93), .S(n94) );
HA_X1 U54 ( .A(n130), .B(n137), .CO(n91), .S(n92) );
FA_X1 U53 ( .A(n144), .B(n158), .CI(n151), .CO(n89), .S(n90) );
FA_X1 U52 ( .A(n92), .B(n95), .CI(n93), .CO(n87), .S(n88) );
HA_X1 U51 ( .A(n122), .B(n129), .CO(n85), .S(n86) );
FA_X1 U50 ( .A(n136), .B(n157), .CI(n143), .CO(n83), .S(n84) );
FA_X1 U49 ( .A(n91), .B(n150), .CI(n86), .CO(n81), .S(n82) );
FA_X1 U48 ( .A(n84), .B(n89), .CI(n82), .CO(n79), .S(n80) );
HA_X1 U47 ( .A(n114), .B(n121), .CO(n77), .S(n78) );
FA_X1 U46 ( .A(n128), .B(n156), .CI(n149), .CO(n75), .S(n76) );
FA_X1 U45 ( .A(n135), .B(n142), .CI(n85), .CO(n73), .S(n74) );
FA_X1 U44 ( .A(n83), .B(n78), .CI(n76), .CO(n71), .S(n72) );
FA_X1 U43 ( .A(n74), .B(n81), .CI(n72), .CO(n69), .S(n70) );
HA_X1 U42 ( .A(n106), .B(n113), .CO(n67), .S(n68) );
FA_X1 U41 ( .A(n120), .B(n155), .CI(n148), .CO(n65), .S(n66) );
FA_X1 U40 ( .A(n134), .B(n127), .CI(n141), .CO(n63), .S(n64) );
FA_X1 U39 ( .A(n68), .B(n77), .CI(n75), .CO(n61), .S(n62) );
FA_X1 U38 ( .A(n64), .B(n73), .CI(n66), .CO(n59), .S(n60) );
FA_X1 U37 ( .A(n62), .B(n71), .CI(n60), .CO(n57), .S(n58) );
HA_X1 U36 ( .A(n105), .B(n112), .CO(n55), .S(n56) );
FA_X1 U35 ( .A(n119), .B(n126), .CI(n147), .CO(n53), .S(n54) );
FA_X1 U34 ( .A(n133), .B(n140), .CI(n67), .CO(n51), .S(n52) );
FA_X1 U33 ( .A(n65), .B(n56), .CI(n63), .CO(n49), .S(n50) );
FA_X1 U32 ( .A(n61), .B(n54), .CI(n52), .CO(n47), .S(n48) );
FA_X1 U31 ( .A(n59), .B(n50), .CI(n48), .CO(n45), .S(n46) );
FA_X1 U30 ( .A(n104), .B(n139), .CI(n1111), .CO(n43), .S(n44) );
FA_X1 U29 ( .A(n118), .B(n132), .CI(n125), .CO(n41), .S(n42) );
```

```
FA_X1 U28 ( .A(n53), .B(n55), .CI(n51), .CO(n39), .S(n40) );
FA_X1 U27 ( .A(n44), .B(n42), .CI(n49), .CO(n37), .S(n38) );
FA_X1 U26 ( .A(n47), .B(n40), .CI(n38), .CO(n35), .S(n36) );
FA_X1 U25 ( .A(n103), .B(n131), .CI(n110), .CO(n33), .S(n34) );
FA_X1 U24 ( .A(n117), .B(n124), .CI(n43), .CO(n31), .S(n32) );
FA_X1 U23 ( .A(n34), .B(n41), .CI(n39), .CO(n29), .S(n30) );
FA_X1 U22 ( .A(n37), .B(n32), .CI(n30), .CO(n27), .S(n28) );
FA_X1 U211 ( .A(n102), .B(n123), .CI(n109), .CO(n25), .S(n26) );
FA_X1 U20 ( .A(n33), .B(n116), .CI(n26), .CO(n23), .S(n24) );
FA_X1 U19 ( .A(n24), .B(n31), .CI(n29), .CO(n21), .S(n22) );
FA_X1 U18 ( .A(n101), .B(n115), .CI(n108), .CO(n19), .S(n20) );
FA_X1 U17 ( .A(n20), .B(n25), .CI(n23), .CO(n17), .S(n18) );
FA_X1 U16 ( .A(n100), .B(n107), .CI(n19), .CO(n15), .S(n16) );
HA_X1 U15 ( .A(n161), .B(n154), .CO(n14), .S(\product[1] ) );
FA_X1 U14 ( .A(n14), .B(n146), .CI(n98), .CO(n13), .S(\product[2] ) );
FA_X1 U13 ( .A(n13), .B(n96), .CI(n94), .CO(n12), .S(\product[3] ) );
FA_X1 U12 ( .A(n88), .B(n90), .CI(n12), .CO(n111), .S(\product[4] ) );
FA_X1 U111 ( .A(n80), .B(n87), .CI(n111), .CO(n10), .S(\product[5] ) );
FA_X1 U10 ( .A(n70), .B(n79), .CI(n10), .CO(n9), .S(\product[6] ) );
FA_X1 U9 ( .A(n58), .B(n69), .CI(n9), .CO(n8), .S(\product[7] ) );
FA_X1 U8 ( .A(n46), .B(n57), .CI(n8), .CO(n7), .S(\product[8] ) );
FA_X1 U7 ( .A(n36), .B(n45), .CI(n7), .CO(n6), .S(\product[9] ) );
FA_X1 U6 ( .A(n28), .B(n35), .CI(n6), .CO(n5), .S(\product[10] ) );
FA_X1 U5 ( .A(n22), .B(n27), .CI(n5), .CO(n4), .S(\product[11] ) );
FA_X1 U4 ( .A(n21), .B(n18), .CI(n4), .CO(n3), .S(\product[12] ) );
FA_X1 U3 ( .A(n17), .B(n16), .CI(n3), .CO(n2), .S(\product[13] ) );
FA_X1 U21 ( .A(n15), .B(n99), .CI(n2), .CO(\product[15] ), .S(\product[14] )
);
endmodule
```

Tetramax Test Patterns

```
STIL 1.0 { Design 2005; }
Header {
Title " TetraMAX(R) L-2016.03-SP5-i161014_180153 STIL output";
Date "Sun May 12 15:34:43 2024";
History {
Ann {* Uncollapsed Transition Fault Summary Report *}
Ann {* ----------------------------------------------- *}
Ann {* fault class code #faults *}
Ann {* ------------------------------ ---- --------- *}
Ann {* Detected DT 1248 *}
Ann {* Possibly detected PT 0 *}
Ann {* Undetectable UD 0 *}
Ann {* ATPG untestable AU 0 *}
Ann {* Not detected ND 0 *}
Ann {* ----------------------------------------------- *}
Ann {* total faults 1248 *}
Ann {* test coverage 100.00% *}
Ann {* ----------------------------------------------- *}
Ann {* *}
Ann {* Pattern Summary Report *}
Ann {* ----------------------------------------------- *}
Ann {* #internal patterns 18 *}
Ann {* #basic_scan patterns 18 *}
Ann {* ----------------------------------------------- *}
Ann {* *}
Ann {* rule severity #fails description *}
Ann {* ---- -------- ------ ------------------------------- *}
Ann {* B9 warning 1 undriven module internal net *}
Ann {* B10 warning 1 unconnected module internal net *}
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 = p_unit *}
Ann {* Unified STIL Flow *}
Ann {* min_n_shifts = 1 *}
Ann {* serial_flag = 1 *}
}
}
Signals {
```

```
"a[7]" In; "a[6]" In; "a[5]" In; "a[4]" In; "a[3]" In; "a[2]" In; "a[1]" In;
"a[0]" In;
"b[7]" In; "b[6]" In; "b[5]" In; "b[4]" In; "b[3]" In; "b[2]" In; "b[1]" In;
"b[0]" In;
"c[15]" In; "c[14]" In; "c[13]" In; "c[12]" In; "c[11]" In; "c[10]" In; "c[9]" In;
"c[8]" In; "c[7]" In; "c[6]" In; "c[5]" In; "c[4]" In; "c[3]" In; "c[2]" In;
"c[1]" In;
"c[0]" In; "result[15]" Out; "result[14]" Out; "result[13]" Out; "result[12]" Out;
"result[11]" Out; "result[10]" Out; "result[9]" Out; "result[8]" Out; "result[7]"
Out;
"result[6]" Out; "result[5]" Out; "result[4]" Out; "result[3]" Out; "result[2]"
Out;
"result[1]" Out; "result[0]" Out;
}
SignalGroups {
"_default_In_Timing_" = '"a[7]" + "a[6]" + "a[5]" + "a[4]" + "a[3]" + "a[2]" +
"a[1]" + "a[0]" + "b[7]" + "b[6]" + "b[5]" + "b[4]" + "b[3]" + "b[2]" +
"b[1]" + "b[0]" + "c[15]" + "c[14]" + "c[13]" + "c[12]" + "c[11]" + "c[10]" +
"c[9]" + "c[8]" + "c[7]" + "c[6]" + "c[5]" + "c[4]" + "c[3]" + "c[2]" +
"c[1]" + "c[0]"'; // #signals=32
"_pi" = '"a[7]" + "a[6]" + "a[5]" + "a[4]" + "a[3]" + "a[2]" + "a[1]" +
"a[0]" + "b[7]" + "b[6]" + "b[5]" + "b[4]" + "b[3]" + "b[2]" + "b[1]" +
"b[0]" + "c[15]" + "c[14]" + "c[13]" + "c[12]" + "c[11]" + "c[10]" + "c[9]" +
"c[8]" + "c[7]" + "c[6]" + "c[5]" + "c[4]" + "c[3]" + "c[2]" + "c[1]" +
"c[0]"'; // #signals=32
"_in" = '"a[7]" + "a[6]" + "a[5]" + "a[4]" + "a[3]" + "a[2]" + "a[1]" +
"a[0]" + "b[7]" + "b[6]" + "b[5]" + "b[4]" + "b[3]" + "b[2]" + "b[1]" +
"b[0]" + "c[15]" + "c[14]" + "c[13]" + "c[12]" + "c[11]" + "c[10]" + "c[9]" +
"c[8]" + "c[7]" + "c[6]" + "c[5]" + "c[4]" + "c[3]" + "c[2]" + "c[1]" +
"c[0]"'; // #signals=32
"_default_Out_Timing_" = '"result[15]" + "result[14]" + "result[13]" +
"result[12]" + "result[11]" + "result[10]" + "result[9]" + "result[8]" +
"result[7]" + "result[6]" + "result[5]" + "result[4]" + "result[3]" +
"result[2]" + "result[1]" + "result[0]"'; // #signals=16
"_po" = '"result[15]" + "result[14]" + "result[13]" + "result[12]" +
"result[11]" + "result[10]" + "result[9]" + "result[8]" + "result[7]" +
"result[6]" + "result[5]" + "result[4]" + "result[3]" + "result[2]" +
"result[1]" + "result[0]"'; // #signals=16
"_out" = '"result[15]" + "result[14]" + "result[13]" + "result[12]" +
"result[11]" + "result[10]" + "result[9]" + "result[8]" + "result[7]" +
"result[6]" + "result[5]" + "result[4]" + "result[3]" + "result[2]" +
"result[1]" + "result[0]"'; // #signals=16
}
```

```
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"=\r16 X ; }
"forcePI": V { "_pi"=\r32 # ; }
"measurePO": V { "_po"=\r16 # ; }
}
// 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"=\r32 0 ; "_po"=\r16 X ; }
"pattern 0": V { "_pi"=0010001010000000011100000000000; }
Call "capture" {
"_pi"=1101111110111101101111111111111111; "_po"=LHHLLHLLHLHLLLHL; }
"pattern 1": V { "_pi"=1000101010100011101111111111111111; }
Call "capture" {
"_pi"=1111100011011000000000000000000000; "_po"=HHLHLLLHLHLLLLLL; }
"pattern 2": V { "_pi"=0000110100010100000000001001100011; }
Call "capture" {
"_pi"=1111001100101111011111010001110; "_po"=HLHLHLHHLLHLHLHH; }
"pattern 3": V { "_pi"=1000001100000111110110010100 1101; }
Call "capture" {
"_pi"=0000110011100111011011010110101; "_po"=LHHHHLLHHLLLHLLH; }
"pattern 4": V { "_pi"=0010100111011010010001001 1011; }
Call "capture" {
"_pi"=1000110100000001000110101010 1000; "_po"=LLLHHLHHLLHHLHLH; }
"pattern 5": V { "_pi"=1111010011000000101000100 1001100; }
Call "capture" {
"_pi"=0101000010100001100011010101010 1; "_po"=HLHHHHHHHLHLLLHLH; }
"pattern 6": V { "_pi"=0000111100000011001101001010 0101; }
Call "capture" {
"_pi"=1010111000000110111101011010 1100; "_po"=HHHHHLLHHHLLLLLL; }
"pattern 7": V { "_pi"=0001011011100111001101100 0101111; }
Call "capture" {
"_pi"=1100110111010011110101110010 0101; "_po"=HLLLLLLLLLLHHHLL; }
"pattern 8": V { "_pi"=1010100101110011100110110 0010111; }
Call "capture" {
"_pi"=0011101010011000111010111001 0010; "_po"=LLLLHHHLLLLLLLHL; }
"pattern 9": V { "_pi"=0101001100111110010011011 0001011; }
Call "capture" {
"_pi"=1001110111110000111101011100 1001; "_po"=HLLLHLLLHHHHHLLH; }
"pattern 10": V { "_pi"=0111110100000111000011101 0111000; }
Call "capture" {
"_pi"=0010001101111001010101110 0010111; "_po"=LHHLLHHHHLHLLLHL; }
"pattern 11": V { "_pi"=1111110111111111101011110 0100001; }
Call "capture" {
"_pi"=0010101011001001000001100 1111000; "_po"=LLHLLHHHLHHHLLHL; }
"pattern 12": V { "_pi"=1011101010101000010101111 0010000; }
Call "capture" {
"_pi"=1111111011001101000001100 111100; "_po"=HHHLHLLLHHLHLHHL; }
"pattern 13": V { "_pi"=1011111101100011000000111 0110101; }
Call "capture" {
```

"_pi"=1100100011011011111101100011011O1; "_po"=HLLHLHHHHLLLLHLH; }
"pattern 14": V { "_pi"=0101111111110001100000011101l010; }
Call "capture" {
"_pi"=1010010011111101111101l000110110; "_po"=HLLHHLLLLHLLHLHL; }
"pattern 15": V { "_pi"=0100110101001011111010001001O000; }
Call "capture" {
"_pi"=1010001111001000010111101l101000; "_po"=HHLHHHHLLHLLLLLL; }
"pattern 16": V { "_pi"=1111001000000101101101l100001101; }
Call "capture" {
"_pi"=0000000110001011011000011010O001; "_po"=LHHLLLHLLLHLHHLL; }
"pattern 17": V { "_pi"=1101100101101010001011011111O010; }
Call "capture" {
"_pi"=1001101110101011110011011O111101; "_po"=LLHHLHLHLHLLLHHL; }
}
// Patterns reference 54 V statements, generating 54 test cycles

```
****************************************
Report : area
Design : p_unit
Version: O-2018.06-SP3
Date   : Sun May 12 15:23:24 2024
****************************************

Information: Updating design information... (UID-85)
Library(s) Used:

    NangateOpenCellLibrary (File: /synopsys/Nangate_FreePDK45/
NangateOpenCellLibrary_PDKv1_3_v2010_12/Front_End/Liberty/CCS/
NangateOpenCellLibrary.db)

Number of ports:                          48
Number of nets:                          255
Number of cells:                         154
Number of combinational cells:           153
Number of sequential cells:                0
Number of macros/black boxes:              0
Number of buf/inv:                        16
Number of references:                      6

Combinational area:               351.652002
Buf/Inv area:                       8.512000
Noncombinational area:              0.000000
Macro/Black Box area:               0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:                  351.652002
Total area:                 undefined
1
```

```
Loading db file '/synopsys/Nangate_FreePDK45/NangateOpenCellLibrary_PDKv1_3_v2010_12/
Front_End/Liberty/CCS/NangateOpenCellLibrary.db'
Information: Propagating switching activity (low effort zero delay simulation).
(PWR-6)
Warning: There is no defined clock in the design. (PWR-80)
Warning: Design has unannotated primary inputs. (PWR-414)

****************************************
Report : power
        -analysis_effort low
Design : p_unit
Version: O-2018.06-SP3
Date   : Sun May 12 15:23:26 2024
****************************************


Library(s) Used:

    NangateOpenCellLibrary (File: /synopsys/Nangate_FreePDK45/
NangateOpenCellLibrary_PDKv1_3_v2010_12/Front_End/Liberty/CCS/
NangateOpenCellLibrary.db)


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

Design        Wire Load Model            Library
------------------------------------------------
p_unit                5K_hvratio_1_1    NangateOpenCellLibrary


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


  Cell Internal Power  = 225.9260 uW   (58%)
  Net Switching Power  = 162.5672 uW   (42%)
                         ---------
Total Dynamic Power    = 388.4932 uW  (100%)

Cell Leakage Power     =  20.7711 uW

Information: report_power power group summary does not include estimated clock tree
power. (PWR-789)

                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power
(   %    )  Attrs
---------------------------------------------------------------------------------------------------
io_pad             0.0000            0.0000            0.0000            0.0000
(   0.00%)
memory             0.0000            0.0000            0.0000            0.0000
(   0.00%)
black_box          0.0000            0.0000            0.0000            0.0000
(   0.00%)
clock_network      0.0000            0.0000            0.0000            0.0000
(   0.00%)
```

```
register           0.0000            0.0000            0.0000            0.0000
(   0.00%)
sequential         0.0000            0.0000            0.0000            0.0000
(   0.00%)
combinational    225.9260          162.5672        2.0771e+04          409.2642
( 100.00%)
---------------------------------------------------------------------------------------------------
Total            225.9260 uW       162.5672 uW     2.0771e+04 nW       409.2642 uW
1
```

```
****************************************
Report : timing
        -path full
        -delay max
        -nworst 10
        -max_paths 10
Design : p_unit
Version: O-2018.06-SP3
Date   : Sun May 12 15:23:26 2024
****************************************

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

  Startpoint: b[2] (input port)
  Endpoint: result[15] (output port)
  Path Group: (none)
  Path Type: max

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

  Point                                    Incr       Path
  -----------------------------------------------------------
  input external delay                     0.00       0.00 f
  b[2] (in)                                0.00       0.00 f
  U142/ZN (INV_X1)                         0.05       0.05 r
  U119/ZN (NOR2_X1)                        0.04       0.08 f
  U57/CO (HA_X1)                           0.05       0.13 f
  U55/CO (FA_X1)                           0.06       0.19 f
  U52/S (FA_X1)                            0.09       0.28 r
  U12/S (FA_X1)                            0.09       0.36 f
  U1_4/CO (FA_X1)                          0.07       0.44 f
  U1_5/CO (FA_X1)                          0.06       0.50 f
  U1_6/CO (FA_X1)                          0.06       0.56 f
  U1_7/CO (FA_X1)                          0.06       0.61 f
  U1_8/CO (FA_X1)                          0.06       0.67 f
  U1_9/CO (FA_X1)                          0.06       0.73 f
  U1_10/CO (FA_X1)                         0.06       0.79 f
  U1_11/CO (FA_X1)                         0.06       0.85 f
  U1_12/CO (FA_X1)                         0.06       0.91 f
  U1_13/CO (FA_X1)                         0.06       0.97 f
  U1_14/CO (FA_X1)                         0.06       1.03 f
  U1_15/S (FA_X1)                          0.08       1.12 r
  result[15] (out)                         0.01       1.13 r
  data arrival time                                   1.13
  -----------------------------------------------------------
  (Path is unconstrained)


  Startpoint: a[0] (input port)
  Endpoint: result[15] (output port)
  Path Group: (none)
  Path Type: max

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

  Point                                    Incr       Path
```

```
------------------------------------------------------------
input external delay                      0.00       0.00 f
a[0] (in)                                 0.00       0.00 f
U148/ZN (INV_X1)                          0.04       0.04 r
U119/ZN (NOR2_X1)                         0.04       0.08 f
U57/CO (HA_X1)                            0.05       0.13 f
U55/CO (FA_X1)                            0.06       0.19 f
U52/S (FA_X1)                             0.09       0.28 r
U12/S (FA_X1)                             0.09       0.36 f
U1_4/CO (FA_X1)                           0.07       0.44 f
U1_5/CO (FA_X1)                           0.06       0.50 f
U1_6/CO (FA_X1)                           0.06       0.56 f
U1_7/CO (FA_X1)                           0.06       0.61 f
U1_8/CO (FA_X1)                           0.06       0.67 f
U1_9/CO (FA_X1)                           0.06       0.73 f
U1_10/CO (FA_X1)                          0.06       0.79 f
U1_11/CO (FA_X1)                          0.06       0.85 f
U1_12/CO (FA_X1)                          0.06       0.91 f
U1_13/CO (FA_X1)                          0.06       0.97 f
U1_14/CO (FA_X1)                          0.06       1.03 f
U1_15/S (FA_X1)                           0.08       1.12 r
result[15] (out)                          0.01       1.13 r
data arrival time                                    1.13
------------------------------------------------------------
(Path is unconstrained)


Startpoint: b[2] (input port)
Endpoint: result[15] (output port)
Path Group: (none)
Path Type: max

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

Point                                     Incr       Path
------------------------------------------------------------
input external delay                      0.00       0.00 f
b[2] (in)                                 0.00       0.00 f
U142/ZN (INV_X1)                          0.05       0.05 r
U119/ZN (NOR2_X1)                         0.04       0.08 f
U57/CO (HA_X1)                            0.05       0.13 f
U55/CO (FA_X1)                            0.06       0.19 f
U52/S (FA_X1)                             0.09       0.28 r
U12/S (FA_X1)                             0.09       0.36 f
U1_4/CO (FA_X1)                           0.07       0.44 f
U1_5/CO (FA_X1)                           0.06       0.50 f
U1_6/CO (FA_X1)                           0.06       0.56 f
U1_7/CO (FA_X1)                           0.06       0.61 f
U1_8/CO (FA_X1)                           0.06       0.67 f
U1_9/CO (FA_X1)                           0.06       0.73 f
U1_10/CO (FA_X1)                          0.06       0.79 f
U1_11/CO (FA_X1)                          0.06       0.85 f
U1_12/CO (FA_X1)                          0.06       0.91 f
U1_13/CO (FA_X1)                          0.06       0.97 f
U1_14/CO (FA_X1)                          0.06       1.03 f
U1_15/S (FA_X1)                           0.08       1.11 r
result[15] (out)                          0.01       1.13 r
data arrival time                                    1.13
------------------------------------------------------------
(Path is unconstrained)
```

```
Startpoint: b[2] (input port)
Endpoint: result[15] (output port)
Path Group: (none)
Path Type: max

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

Point                                    Incr       Path
-----------------------------------------------------------
input external delay                     0.00       0.00 f
b[2] (in)                                0.00       0.00 f
U142/ZN (INV_X1)                         0.05       0.05 r
U119/ZN (NOR2_X1)                        0.04       0.08 f
U57/CO (HA_X1)                           0.05       0.13 f
U55/CO (FA_X1)                           0.06       0.19 f
U52/S (FA_X1)                            0.09       0.28 r
U12/S (FA_X1)                            0.09       0.36 f
U1_4/CO (FA_X1)                          0.07       0.44 f
U1_5/CO (FA_X1)                          0.06       0.50 f
U1_6/CO (FA_X1)                          0.06       0.56 f
U1_7/CO (FA_X1)                          0.06       0.61 f
U1_8/CO (FA_X1)                          0.06       0.67 f
U1_9/CO (FA_X1)                          0.06       0.73 f
U1_10/CO (FA_X1)                         0.06       0.79 f
U1_11/CO (FA_X1)                         0.06       0.85 f
U1_12/CO (FA_X1)                         0.06       0.91 f
U1_13/CO (FA_X1)                         0.06       0.97 f
U1_14/CO (FA_X1)                         0.06       1.03 f
U1_15/S (FA_X1)                          0.08       1.11 r
result[15] (out)                         0.01       1.13 r
data arrival time                                   1.13
-----------------------------------------------------------
(Path is unconstrained)


Startpoint: b[2] (input port)
Endpoint: result[15] (output port)
Path Group: (none)
Path Type: max

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

Point                                    Incr       Path
-----------------------------------------------------------
input external delay                     0.00       0.00 f
b[2] (in)                                0.00       0.00 f
U142/ZN (INV_X1)                         0.05       0.05 r
U119/ZN (NOR2_X1)                        0.04       0.08 f
U57/CO (HA_X1)                           0.05       0.13 f
U55/CO (FA_X1)                           0.06       0.19 f
U52/S (FA_X1)                            0.09       0.28 r
U12/S (FA_X1)                            0.09       0.36 f
U1_4/CO (FA_X1)                          0.07       0.44 f
U1_5/CO (FA_X1)                          0.06       0.50 f
U1_6/CO (FA_X1)                          0.06       0.56 f
U1_7/CO (FA_X1)                          0.06       0.61 f
```

```
U1_8/CO (FA_X1)                          0.06       0.67 f
U1_9/CO (FA_X1)                          0.06       0.73 f
U1_10/CO (FA_X1)                         0.06       0.79 f
U1_11/CO (FA_X1)                         0.06       0.85 f
U1_12/CO (FA_X1)                         0.06       0.91 f
U1_13/CO (FA_X1)                         0.06       0.97 f
U1_14/CO (FA_X1)                         0.06       1.03 f
U1_15/S (FA_X1)                          0.08       1.11 r
result[15] (out)                         0.01       1.13 r
data arrival time                                   1.13
---------------------------------------------------------------
(Path is unconstrained)


Startpoint: b[2] (input port)
Endpoint: result[15] (output port)
Path Group: (none)
Path Type: max

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

Point                                    Incr       Path
---------------------------------------------------------------
input external delay                     0.00       0.00 f
b[2] (in)                                0.00       0.00 f
U142/ZN (INV_X1)                         0.05       0.05 r
U119/ZN (NOR2_X1)                        0.04       0.08 f
U57/CO (HA_X1)                           0.05       0.13 f
U55/CO (FA_X1)                           0.06       0.19 f
U52/S (FA_X1)                            0.09       0.28 r
U12/S (FA_X1)                            0.09       0.36 f
U1_4/CO (FA_X1)                          0.07       0.44 f
U1_5/CO (FA_X1)                          0.06       0.50 f
U1_6/CO (FA_X1)                          0.06       0.56 f
U1_7/CO (FA_X1)                          0.06       0.61 f
U1_8/CO (FA_X1)                          0.06       0.67 f
U1_9/CO (FA_X1)                          0.06       0.73 f
U1_10/CO (FA_X1)                         0.06       0.79 f
U1_11/CO (FA_X1)                         0.06       0.85 f
U1_12/CO (FA_X1)                         0.06       0.91 f
U1_13/CO (FA_X1)                         0.06       0.97 f
U1_14/CO (FA_X1)                         0.06       1.03 f
U1_15/S (FA_X1)                          0.08       1.11 r
result[15] (out)                         0.01       1.13 r
data arrival time                                   1.13
---------------------------------------------------------------
(Path is unconstrained)


Startpoint: b[2] (input port)
Endpoint: result[15] (output port)
Path Group: (none)
Path Type: max

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

Point                                    Incr       Path
---------------------------------------------------------------
```

```
input external delay                      0.00       0.00 f
b[2] (in)                                 0.00       0.00 f
U142/ZN (INV_X1)                          0.05       0.05 r
U119/ZN (NOR2_X1)                         0.04       0.08 f
U57/CO (HA_X1)                            0.05       0.13 f
U55/CO (FA_X1)                            0.06       0.19 f
U52/S (FA_X1)                             0.09       0.28 r
U12/S (FA_X1)                             0.09       0.36 f
U1_4/CO (FA_X1)                           0.07       0.44 f
U1_5/CO (FA_X1)                           0.06       0.50 f
U1_6/CO (FA_X1)                           0.06       0.56 f
U1_7/CO (FA_X1)                           0.06       0.61 f
U1_8/CO (FA_X1)                           0.06       0.67 f
U1_9/CO (FA_X1)                           0.06       0.73 f
U1_10/CO (FA_X1)                          0.06       0.79 f
U1_11/CO (FA_X1)                          0.06       0.85 f
U1_12/CO (FA_X1)                          0.06       0.91 f
U1_13/CO (FA_X1)                          0.06       0.97 f
U1_14/CO (FA_X1)                          0.06       1.03 f
U1_15/S (FA_X1)                           0.08       1.11 r
result[15] (out)                          0.01       1.13 r
data arrival time                                    1.13
-----------------------------------------------------------
(Path is unconstrained)


Startpoint: b[2] (input port)
Endpoint: result[15] (output port)
Path Group: (none)
Path Type: max

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

Point                                     Incr       Path
-----------------------------------------------------------
input external delay                      0.00       0.00 f
b[2] (in)                                 0.00       0.00 f
U142/ZN (INV_X1)                          0.05       0.05 r
U119/ZN (NOR2_X1)                         0.04       0.08 f
U57/CO (HA_X1)                            0.05       0.13 f
U55/CO (FA_X1)                            0.06       0.19 f
U52/S (FA_X1)                             0.09       0.28 r
U12/S (FA_X1)                             0.09       0.36 f
U1_4/CO (FA_X1)                           0.07       0.44 f
U1_5/CO (FA_X1)                           0.06       0.49 f
U1_6/CO (FA_X1)                           0.06       0.55 f
U1_7/CO (FA_X1)                           0.06       0.61 f
U1_8/CO (FA_X1)                           0.06       0.67 f
U1_9/CO (FA_X1)                           0.06       0.73 f
U1_10/CO (FA_X1)                          0.06       0.79 f
U1_11/CO (FA_X1)                          0.06       0.85 f
U1_12/CO (FA_X1)                          0.06       0.91 f
U1_13/CO (FA_X1)                          0.06       0.97 f
U1_14/CO (FA_X1)                          0.06       1.03 f
U1_15/S (FA_X1)                           0.08       1.11 r
result[15] (out)                          0.01       1.13 r
data arrival time                                    1.13
-----------------------------------------------------------
(Path is unconstrained)
```

```
Startpoint: b[2] (input port)
Endpoint: result[15] (output port)
Path Group: (none)
Path Type: max

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

Point                                    Incr       Path
-----------------------------------------------------------
input external delay                     0.00       0.00 f
b[2] (in)                                0.00       0.00 f
U142/ZN (INV_X1)                         0.05       0.05 r
U119/ZN (NOR2_X1)                        0.04       0.08 f
U57/CO (HA_X1)                           0.05       0.13 f
U55/CO (FA_X1)                           0.06       0.19 f
U52/S (FA_X1)                            0.09       0.28 r
U12/S (FA_X1)                            0.09       0.36 f
U1_4/CO (FA_X1)                          0.07       0.44 f
U1_5/CO (FA_X1)                          0.06       0.50 f
U1_6/CO (FA_X1)                          0.06       0.56 f
U1_7/CO (FA_X1)                          0.06       0.61 f
U1_8/CO (FA_X1)                          0.06       0.67 f
U1_9/CO (FA_X1)                          0.06       0.73 f
U1_10/CO (FA_X1)                         0.06       0.79 f
U1_11/CO (FA_X1)                         0.06       0.85 f
U1_12/CO (FA_X1)                         0.06       0.91 f
U1_13/CO (FA_X1)                         0.06       0.97 f
U1_14/CO (FA_X1)                         0.06       1.03 f
U1_15/S (FA_X1)                          0.08       1.11 r
result[15] (out)                         0.01       1.13 r
data arrival time                                   1.13
-----------------------------------------------------------
(Path is unconstrained)


Startpoint: b[2] (input port)
Endpoint: result[15] (output port)
Path Group: (none)
Path Type: max

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

Point                                    Incr       Path
-----------------------------------------------------------
input external delay                     0.00       0.00 f
b[2] (in)                                0.00       0.00 f
U142/ZN (INV_X1)                         0.05       0.05 r
U119/ZN (NOR2_X1)                        0.04       0.08 f
U57/CO (HA_X1)                           0.05       0.13 f
U55/CO (FA_X1)                           0.06       0.19 f
U52/S (FA_X1)                            0.09       0.28 r
U12/S (FA_X1)                            0.09       0.36 f
U1_4/CO (FA_X1)                          0.07       0.44 f
U1_5/CO (FA_X1)                          0.06       0.50 f
U1_6/CO (FA_X1)                          0.06       0.56 f
U1_7/CO (FA_X1)                          0.06       0.61 f
U1_8/CO (FA_X1)                          0.06       0.67 f
```

```
  U1_9/CO (FA_X1)                          0.06       0.73 f
  U1_10/CO (FA_X1)                         0.06       0.79 f
  U1_11/CO (FA_X1)                         0.06       0.85 f
  U1_12/CO (FA_X1)                         0.06       0.91 f
  U1_13/CO (FA_X1)                         0.06       0.97 f
  U1_14/CO (FA_X1)                         0.06       1.03 f
  U1_15/S (FA_X1)                          0.08       1.11 r
  result[15] (out)                         0.01       1.13 r
  data arrival time                                   1.13
  ---------------------------------------------------------------
  (Path is unconstrained)


1
```