TERNARY CONTENT ADDRESSABLE MEMORY

Detailed report about the project for the course VLSI PHYSICAL DESIGN

Submitted by

GHANTASALA NAGENDRA BABU – AP22110020017
CHEBROLU NANDA GOPAL – AP22110020018

Under the guidance of

Dr. Saswat Kumar Ram

Assistant Professor, ECE

Department of Electronics and Communication engineering,
SRM University AP,
Neerukonda – 522240, Guntur (district), Andhra Pradesh.

CERTIFICATE

This is to certify that Mr.Ghantasala Nagendra Babu, Mr.Chebrolu Nanda Gopal have successfully completed the project titled "**TERNARY CONTENT ADDRESABLE MEMORY**" is being submitted for the award of degree of **BACHELOR OF TECHNOLOGY** in the department of Electronics and Communication Engineering at SRM University AP. Their work was conducted under my guidance and supervision, and it reflects their dedication, hard work.

Date:

Place: SRM University AP

Dr. Saswat Kumar Ram
Department of Electronics and Communication Engineering,
SRM University AP.

ACKNOWLEDGEMENT

We want to express our heartfelt gratitude and appreciation to all the individuals who have contributed to the successful completion of our college group project. We would also like to thank our professor, **Dr. Saswat Kumar Ram** for their guidance and support. Their expertise and mentorship have been instrumental in shaping our project and pushing us toward excellence. His depth of knowledge and experience in the field provided us with a solid foundation on which to build our research and implementation. Throughout the semester, He consistently challenged us to think critically, encouraging us to explore innovative approaches and consider different perspectives. His regular feedback, constructive criticism, and guidance have played a pivotal role in refining our ideas, improving our methodologies, and enhancing the quality of our work.

We want to thank our fellow group members for their hard work, dedication, and cooperation throughout the project. Each team member played a crucial role, bringing unique skills and perspectives, greatly enriching our work. We are truly grateful for the collaboration, guidance, support, and resources provided. Thank you all for making this project a fulfilling and rewarding experience.

Their encouragement and patience provided the motivation we needed to dedicate ourselves to this internship. We also thank SRM University, AP, for offering us this opportunity and providing excellent facilities and support throughout our project.

Ghantasala Nagendra Babu - AP22110020017

Chebrolu Nanda Gopal – AP22110020018

TABLE OF THE CONTENTS

1. ABSTRACT

This project focuses on the complete ASIC design and implementation of a Ternary Content Addressable Memory (TCAM), a high-speed search engine commonly used in applications requiring rapid data lookup, such as networking devices, firewalls, and routing tables. The primary objective of the work is to follow a comprehensive ASIC flow to model, simulate, verify, synthesize, test, and physically implement a fully functional TCAM using industry-standard EDA tools.

The design begins at the Register Transfer Level (RTL), where the TCAM architecture is modeled in Verilog HDL. The structure includes key components such as a match detection unit, line arrays, and a memory controller capable of handling ternary data (0, 1, X). The RTL is written in a modular and parameterizable fashion to support scalability in terms of key width, data width, and depth. A testbench is developed to simulate various operational scenarios, including valid match, mismatch, and undefined ('X') conditions across all memory entries. Functional simulation is carried out using the Cadence NC Launch environment, with comprehensive waveform inspection and validation of the core TCAM logic.

To ensure complete functional correctness, code coverage and functional coverage metrics are collected and analyzed using Cadence Incisive Metrics Center (IMC). The feedback from the coverage analysis is used to refine the testbench and introduce additional corner-case stimuli, allowing us to achieve close to 100% coverage. The verified RTL is then synthesized using Cadence Genus, where the design is mapped onto a standard cell library targeting a 90nm technology node. The synthesis process is optimized for area, timing, and power, and synthesis reports are analyzed to verify the quality of results.

Linting is performed post-synthesis to check for RTL coding style issues, unreachable code, and structural warnings using in-built Genus lint checks. To enhance testability, Design-for-Test (DFT) techniques are applied, including scan chain insertion and the generation of Automatic Test Patterns (ATPG) using Cadence Modus. This step ensures fault coverage for post-manufacturing testing and enhances the robustness of the TCAM design. Logical Equivalence Checking (LEC) using Cadence Conformal is then conducted to confirm that the functionality of the synthesized netlist matches the original RTL, ensuring no discrepancies were introduced during synthesis.

After passing all front-end verification stages, the design is taken into the physical design flow using Cadence Innovus. Physical design begins with floorplanning, followed by standard cell placement and Clock Tree Synthesis (CTS). Detailed routing is performed while adhering to design rules and minimizing wire length, congestion, and signal integrity issues. Power planning ensures proper distribution of VDD and GND through the use of power rings and stripes. Post-route Static Timing Analysis (STA) is performed to validate that the design meets all setup and hold time constraints under worst-case Process-Voltage-Temperature (PVT) corners.

Finally, the layout is verified through DRC (Design Rule Check), and the final design is exported in GDSII format, ready for fabrication. This marks the successful completion of the ASIC implementation of a high-speed TCAM core. The project not only showcases a practical application of a content-addressable memory design but also demonstrates a complete ASIC design flow — from RTL design and functional simulation to synthesis, DFT, physical implementation, timing analysis, and final tape-out preparation.

This work stands as a comprehensive example of how modern EDA tools and methodologies are employed in the ASIC industry, bridging the gap between theoretical knowledge and practical implementation, while reinforcing concepts of digital logic design, timing closure, verification strategies, and physical design practices.

2.INTRODUCTION

2.1 Overview

Ternary Content Addressable Memory (TCAM) is a specialized high-speed memory that enables parallel data searches based on content rather than address. It is widely used in networking applications such as IP routing, firewalls, and packet classification, where rapid and flexible data matching is essential. TCAM supports ternary logic—'0', '1', and 'X' (don’t care)—allowing wildcard-based searches for complex rule matching.

Due to its parallel comparison capability and ternary matching logic, TCAMs are more complex and power-intensive than traditional memories. Implementing a TCAM in an ASIC flow involves challenges in terms of design optimization, verification, and physical implementation.

This project aims to implement a TCAM using a complete digital ASIC flow, starting from Verilog RTL design to GDSII generation. The process includes simulation, functional verification, synthesis, testability insertion, formal verification, physical design, and post-route timing analysis using industry-standard tools from Cadence.

2.2 Objective

The main objective of this project is to design and implement a Ternary Content Addressable Memory (TCAM) using a complete front-end and back-end ASIC flow. The key goals are:

- Develop a synthesizable Verilog RTL model of a TCAM with support for ternary bit matching.
- Create testbenches and verify functional correctness using simulation and coverage tools.
- Synthesize the design with timing, area, and power optimizations using Cadence Genus.
- Integrate scan chains and generate ATPG patterns for manufacturing testability using Cadence Modus.
- Perform Logical Equivalence Checking (LEC) between RTL and gate-level netlist.
- Complete physical implementation with floorplanning, placement, CTS, and routing using Cadence Innovus.
- Validate timing through post-route STA and generate the GDSII layout for tape-out.

This project demonstrates the application of a structured ASIC design methodology to implement a high-speed memory system suitable for modern networking applications.

3.BACKGROUND WORK

Ternary Content Addressable Memory (TCAM) is a high-speed memory type that enables content-based search operations by comparing input data against stored values in parallel. Unlike traditional RAM, where data retrieval is address-based and sequential, TCAMs enable constant-time search performance, making them highly suitable for applications requiring rapid decision-making such as IP routing, packet filtering, access control lists (ACLs), and network security appliances.

What makes TCAMs unique is their ability to handle ternary values—binary '0' and '1', along with a third state 'X' representing "don't care." This allows for flexible and wildcard-based matching, which is essential for implementing rules in networking tables and classification engines. However, implementing TCAM in ASIC comes with significant challenges due to its inherently parallel architecture, higher power consumption, and complex comparison logic.

In this project, a TCAM design was modeled using Verilog HDL at the RTL (Register Transfer Level), focusing on efficient implementation of match-line logic, ternary bit handling, and memory control. Functional correctness of the design was validated through rigorous testbench simulations using Cadence NC Launch. The design was further verified using coverage analysis via Incisive Metrics Center (IMC) to ensure comprehensive testing of all logic branches and edge cases.

To meet industry design standards, the project followed a structured ASIC flow. Synthesis was performed using Cadence Genus to map the RTL to a gate-level netlist while optimizing for timing, area, and power. Linting was used to catch coding issues and improve design robustness. Design for Testability (DFT) was achieved by inserting scan chains and generating Automatic Test Patterns (ATPG) using Cadence Modus, enhancing post-silicon test coverage. Formal equivalence between RTL and synthesized netlist was validated using Cadence Conformal (LEC).

The back-end physical implementation was carried out in Cadence Innovus, including floorplanning, placement, clock tree synthesis (CTS), and routing. Post-route Static Timing Analysis (STA) ensured that the final layout met timing constraints under worst-case conditions. The final step involved generating a GDSII file, marking the successful completion of the TCAM ASIC design process.

This project demonstrates how a complex, high-speed content-addressable memory can be implemented within a structured ASIC design methodology, highlighting critical aspects of logic design, verification, optimization, and silicon-level readiness in modern VLSI systems.

4.METHODOLOGY

The design and implementation of the Ternary Content Addressable Memory (TCAM) followed a structured ASIC digital design flow, incorporating both front-end and back-end processes using Cadence tools. The methodology adopted in this project ensures functional correctness, testability, and manufacturability while meeting timing, power, and area constraints. The complete flow can be broken down into the following stages:

1. RTL Design and Testbench Development

The TCAM architecture was designed using Verilog HDL, supporting configurable key width, data width, and depth. It included core components such as the memory array, match-line logic, and ternary comparison mechanism. A modular design approach was used for clarity and scalability. A dedicated testbench was developed to simulate various scenarios — including valid matches, mismatches, and don't-care ('X') conditions — to validate the correctness of the RTL.

2. Functional Simulation and Coverage Analysis

Functional simulation was performed using Cadence NC Launch and SimVision tools. The simulation results were observed through waveform inspection, and correctness was confirmed for various input patterns. Code and functional coverage were measured using Incisive Metrics Center (IMC). Based on coverage reports, additional test vectors were introduced to improve corner-case coverage and reach near 100% code and functional coverage.

3. RTL Refinement

Based on simulation and coverage feedback, necessary improvements were made to the RTL. These included optimizing the comparison logic, improving reset conditions, and making the design more robust to handle edge cases.

4. Logic Synthesis

Cadence Genus was used to synthesize the verified RTL into a gate-level netlist using a 90nm technology library. Synthesis constraints such as clock period, input/output delays, and driving loads were defined using SDC (Synopsys Design Constraints) files. Reports related to timing, area, and power were generated and analyzed to ensure quality of results.

5. Linting

Lint checks were performed using in-built tools to detect syntax violations, unreachable code, unused logic, and naming inconsistencies. This step helped clean up the RTL and ensure better readability and maintainability.

6. Design for Test (DFT) and ATPG

Scan chains were inserted into the synthesized netlist using Cadence Modus, making the design testable under manufacturing conditions. Automatic Test Pattern Generation (ATPG) was carried out to generate fault detection vectors for stuck-at and transition faults, enhancing post-silicon validation.

7. Logical Equivalence Checking (LEC)

Cadence Conformal was used to perform Logical Equivalence Checking between the RTL and the synthesized netlist. This ensured that the synthesis process preserved the intended functionality without introducing logic mismatches.

8. Physical Design

The gate-level netlist was imported into Cadence Innovus for back-end implementation. Physical design involved the following key stages:

- **Floorplanning**: Allocation of core area and I/O pads
- **Placement**: Optimal arrangement of standard cells
- **Clock Tree Synthesis (CTS)**: Distribution of the clock signal with minimal skew
- **Routing**: Full connection of nets using metal layers
- **Power Planning**: Creation of power rings and stripes for reliable power distribution

9. Static Timing Analysis (STA)

Post-route STA was conducted to verify that setup and hold timing constraints were met across all critical paths. Corner analysis under worst-case Process-Voltage-Temperature (PVT) conditions ensured design reliability.

10. GDSII Generation

The final layout was exported in GDSII format, ready for tape-out or fabrication. All design rule checks (DRC) and layout-versus-schematic (LVS) checks were completed successfully.

5.PROCEDURE

1. Create a folder.
2. Open the terminal in the folder
3. Type the commands
    - **csh**
    - **source /home/installs/cshrc**
    - **gedit** (where we should write the RTL code)
4. Write the code and name it as tcam.v and again write the test bench code and save it has tcam_tb.v and save it in the folder.
5. And check the two .v files are present.
6. Now again open command open and type the command
    - **nclaunch -new**
7. Click on multiple steps.
8. And click on the create.cds.lib.file.
9. Click save and then click on don't include any libraries.
10. Click on ok.
11. Now select the as tcam.v and tcam_tb.v run and check the errors.(2nd button in tools)
12. Click on worklib, select the tcam _tb.v and elaborator(3rd button in tools)
13. Now click on snapshots and worklib and select testbench file and launch simulation(4th button in tools)
14. Now a simulation window will be opened
15. Select the testbench and click on the run symbol
16. Now wave form will be display.
17. Now type the command
    - **Ncverilog tcam.v tcam_tb.v - access +rwc -coverage all -gui.**
18. Now the console window will be open and the type the command
    - **IMC**
19. Now one window will be open in that window we should load the data in that data it contains cov_work→scope→test
20. Code coverage will be display it should be more than 92%.
21. Now close the window.
22. Now again type the command gedit and paste the sdcfile as comparator and save it as constarints_sdc.sdc and same for rcscript1.tcl.
23. We should have constraints sdc and rcscript1.tcl in the folder.
24. And now again open the terminal and type the command as
    - **pwd(to copy the path)**
25. Now copy the path and paste in rcscript1.tcl in 3rd line and 5th line and save it.
26. And close the window.
27. Now again open the command window and type the command
    - **genus**
28. In that genus window type the command
    - **source rcscript1.tcl** and enter.
29. A layout window will be opened and then click on "+" symbol and select schematic then schematic will display.
30. Now close the window and type the command
    - report_timing -lint (To get linting report)
31. And check the folder whether there is power, area, gate, timing, netlist reports.
32. After linting close the window.
33. For DFT save the genus_DFT.tcl file , in the folder and open the terminal and type the

command genus in that genus window type the command as

- **source genus_script_DFT.tcl** and click on enter

34. For ATPG we should download constraints_top.sdc, slow.v and modus file to the folder and type the command **modus** in terminal and in that modus window type the command
    - **source modus.tcl**
35. Verify the build_model_design top and verifty_test_structures
36. For physical design open the terminal and type the commands
    - **Csh**
    - **Source /home/installs/cshrc**
    - **innovus**

a. As innovus window pop up, perform the following steps:Click on File → import design → autodesign → click on (three dots) symbol →add (>>) → add netlist file (clkcgd_netlist.v) → add →close.

b. LEF files → click on (three dots) → click on (>>) →"/" →home→installs→FOUNDRY→digital→90nm→dig→lef→ add the last two files → close

c. >Power nets: VDD

Ground nets: VSS

d. In MMMC→ create analysis configuration >

Library sets (right click) → new → name:slow→"/"→home→installs→FOUNDRY→digital→90nm→dig→ lib→ slow.lib →add and close

also attach another library set with the name: fast →"/"→home→installs→FOUNDRY→digital→90nm→dig→lib→ fast.lib →add and close .

e. RC Corners (right click) > Name: RC > select QRC technology file → add qrc file (→"/"→home→installs→FOUNDRY→digital→90nm→dig→lib→ qrc →select the file open and ok

f. Delay corners → new → Name: MAXIMUM→RC Corner: RC →library set→ slow→ok

And same process for fast; Name: Delay corners → new → Name: MINIMUM→RC Corner: RC →library set→ fast→ok

g. Constraints → new →Name: Constraints→ add→clkcgd block.sdc →add→close→ok

h. Analysis view →new→Bestcase,→minimum delay→add→ok

and same process for worst case Analysis view →new→worstcase,→maximum delay→add→ok

i. Setup analysis view → Bestcase → apply → 0K

j. Hold analysis view → Worstcase → apply → OK → save.

k. .Click on Floorplan → Specify floorplan→ Ratio (H/W)= 1; core utilization: 0.6

core to left =6

core to right=6

core to top=6

core to bottom= 6

l. Click on Power → connect global nets → Pin name: VDD/VSS; global nets: VDD/VSS →add to list → apply → OK.

m. Power → Power planning → add ring → nets (VDD,VSS) →

Top metal =9(H),

Bottom metal=8(V)

Left metal-8(V)

Right metal=8(V)

n.click on offset →update to reduce the sizing → apply → OK.

o. Power → Power planning → add stripe → nets (VDD, VSS) → ok

p. layer→(Metal 9)→ Horizontal →update →no. of sets = 2 → apply → OK.

Power →Power planning → add stripe → nets (VDD, VSS) → layer (Metal8)→Vertical→ no, of sets = 2 → apply →update > OK.

Route → Special Route →add nets (VDD, VSS) → OK

q. Place → Physical cell → add end caps →Precap cell→select (FILL1)→ Postcap cell →select (FILL2) → apply →OK.

r. Place → Physical cell → add well cap → cell name (FILL1) → distance interval =40 → OK

s. Place → Place Standard cell → Mode → Place I/Opins → OK → OK.

t.Clock → debugging → Unit delay → apply → OK→yes

u. Timing → Debug timing → Clock type→setup→ hold → OK

v. Timing→extract rc→click on save SP&F →ok.

w. Route→Nano route→route→click on SI driven→Time driven→ok

x. Verify→drc→ok

y.file→save design→click on innovus→ok

z.File→save→GDS/OMSS→output file name→clkcgd.gds→ok

a1.open folder→search→gds

a2.open innovus→click on layer→remove the tick mark→remove the file.

37. For STA click on the file→restore design→double click om innovus→select folder→type clkcgd.enc→open→ok.
38. Tools→set mode→specify analysis mode→on chip verification→CPRR→ok(setup)
39. Tools→ set mode→specify analysis mode→on chip verification→CPRR→ok(hold)
40. Timing→report timing→postroute(select)→setup→ok.
41. Open command window→innvous setup→violations→take ss →and same process for hold.
42. Timing→debug timing→hold→ok→take ss.
43. Timing→extract rc→save SPEF→ok
44. File save deskign→innovus→clkcgd.enc→ok
45. Open command window→right click→open teriminal type commands
    - csh
    - source /home/installs/cshrc
    - tempus

46. Click the "+" symbol→layout→file→redesign→tempustiming→select folder (clkcgd.enc)→ok
47. File→read SPEF→select the folder (clkcgd.spef)
    →rc corner→ok.
48. File→read SDC→select the folder (block_sdc)→ok.
49. File→read SDF→select the folder (sdf_file)→ok.
50. Open the command window (tempus) and type the command
    - **read design - physical_data clkcgd.enc.dat/ counter** and enter
51. "+"→Layout→timing→debug timing→select folder→setup→ok→command window→voilation.
52. Timing→debug timing→folder→hold→ok.
53. In analysis page→right click →negative values→showing timing path→data dealy→SDEF→right click→interactive ECO→Add repeat→add→the BUFX2→apply and close→select folder→ok
54. Same process for(BUFX 4,6,8,12) until the filling path is zero.
55. In analysis→folder→hold→ok.
56. Open command window and type the command
    - **write_eco -format innovus -output hold -optimized.tcl** and enter.
57. Check the folder→take tempus screenshot→close tempus and command window→open→innovus→window→file→restore→innovus→TrafficLightController.enc→timing→report timing→post route→hold→ok.
58. Open command window and type the command
    - **source hold -optimized.tcl** and enter
59. Timing→debug→timing→hold→ok→falling points 0.
60. Click→"+"→schematic→file→save→design→innovus→clkcgd.GDS→ ok
61. For LEC open command window type the commands
    - Csh
    - Source /home/installs/cshrc
    - Lec_auto
62. And setup window type the commands
    - **read design clkcgd.v -golden** and enter
63. Now download the LEC_command file and change the path.
64. Now copy the first line in LEC_command file and paste in the setup window
65. Again type the command in set up window
    - read design clkcgd_netlist.v -revised
66. Now press the LEC.
67. Now in compare, click compare, the select add all points in the click and click on compare.
68. Now in tools click on mapping manger→No unmapping points→mapping points→compare points→Right click→diagnosis manger→diagnosis pointer(should be active).
69. Now in compare points, right click →source code.
70. In compare points, right click→schematic.
71. In compare points, right click→report gate.

6. CODE

Design Code

```
module tcam #(parameter WIDTH = 8, DEPTH = 4)(
    input clk,
    input rst,
    input wr_en,
    input search_en,
    input [$clog2(DEPTH)-1:0] wr_addr,
    input [WIDTH-1:0] wr_data,
    input [WIDTH-1:0] wr_mask,
    input [WIDTH-1:0] search_key,
    output reg [WIDTH-1:0] match_data,
    output reg match_found
);
    reg [WIDTH-1:0] data_mem [0:DEPTH-1];
    reg [WIDTH-1:0] mask_mem [0:DEPTH-1];
    integer i;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            for (i = 0; i < DEPTH; i = i + 1) begin
                data_mem[i] <= 0;
                mask_mem[i] <= 0;
            end
        end else if (wr_en) begin
            data_mem[wr_addr] <= wr_data;
            mask_mem[wr_addr] <= wr_mask;
        end
    end

    always @(posedge clk) begin
        match_found = 0;
        match_data = 0;
        if (search_en) begin
            for (i = 0; i < DEPTH; i = i + 1) begin
                if ((data_mem[i] & mask_mem[i]) == (search_key & mask_mem[i])) begin
                    match_found = 1;
                    match_data = data_mem[i];
                end
            end
        end
    end
endmodule
```

TestBench Code

```
module tcam_tb;
   parameter WIDTH = 8;
   parameter DEPTH = 4;
   reg clk, rst, wr_en, search_en;
   reg [$clog2(DEPTH)-1:0] wr_addr;
   reg [WIDTH-1:0] wr_data, wr_mask, search_key;
   wire [WIDTH-1:0] match_data;
   wire match_found;
   simple_tcam #(WIDTH, DEPTH) dut (
      .clk(clk),
      .rst(rst),
      .wr_en(wr_en),
      .search_en(search_en),
      .wr_addr(wr_addr),
      .wr_data(wr_data),
      .wr_mask(wr_mask),
      .search_key(search_key),
      .match_data(match_data),
      .match_found(match_found)
   );
   always #5 clk = ~clk;
   initial begin
      $dumpfile("simple_tcam.vcd");
      $dumpvars(0, tb_simple_tcam);
      clk = 0; rst = 1;
      wr_en = 0; search_en = 0;
```

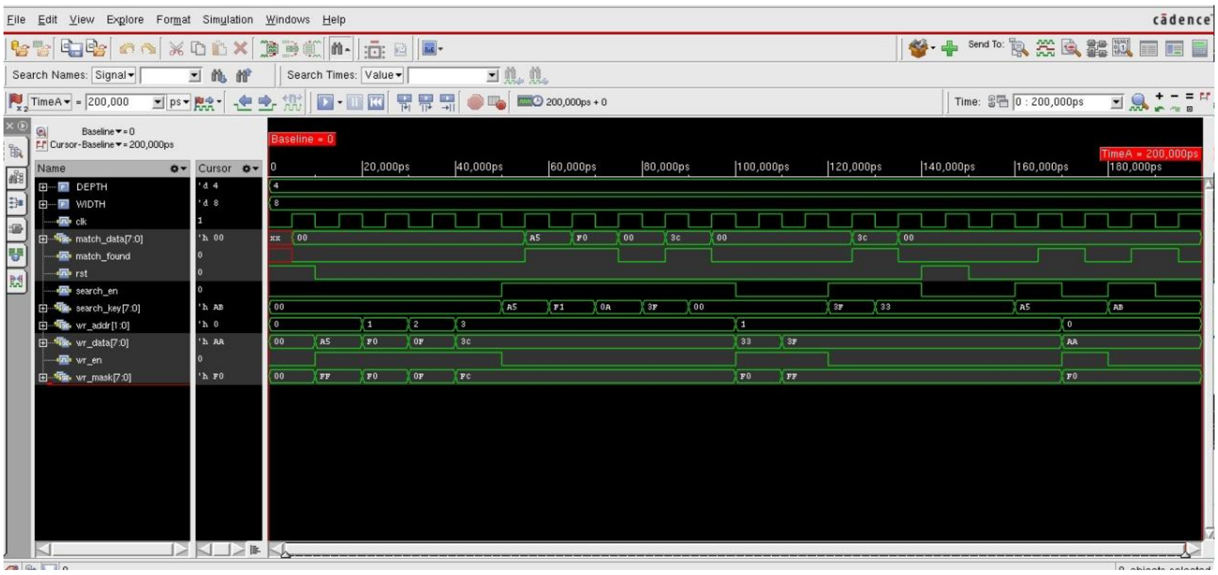
```
wr_data = 0; wr_mask = 0; search_key = 0; wr_addr = 0;
#10 rst = 0;

// Write 4 entries
wr_en = 1;
wr_addr = 0; wr_data = 8'hA5; wr_mask = 8'hFF; #10;
wr_addr = 1; wr_data = 8'hF0; wr_mask = 8'hF0; #10;
wr_addr = 2; wr_data = 8'h0F; wr_mask = 8'h0F; #10;
wr_addr = 3; wr_data = 8'h3C; wr_mask = 8'hFC; #10;
wr_en = 0;

// Search tests
search_key = 8'hA5; search_en = 1; #10; search_en = 0;  // Exact match
search_key = 8'hF1; search_en = 1; #10; search_en = 0;  // Masked match
search_key = 8'h0A; search_en = 1; #10; search_en = 0;  // Masked match
search_key = 8'h3F; search_en = 1; #10; search_en = 0;  // Close to match
search_key = 8'h00; search_en = 1; #10; search_en = 0;  // No match

// Write to same address again (overwrite)
wr_en = 1;
wr_addr = 1; wr_data = 8'h33; wr_mask = 8'hF0; #10;
wr_addr = 1; wr_data = 8'h3F; wr_mask = 8'hFF; #10;
wr_en = 0;

// Search for newly written value
search_key = 8'h3F; search_en = 1; #10; search_en = 0;
```

```
        // Search for old value (should not match)
        search_key = 8'h33; search_en = 1; #10; search_en = 0;
        // Reset to clear memory
        rst = 1; #10;
        rst = 0; #10;

        // Search after reset (should not match anything)
        search_key = 8'hA5; search_en = 1; #10; search_en = 0;

        // Trigger toggle on search_en when wr_en is high
        wr_en = 1; wr_addr = 0; wr_data = 8'hAA; wr_mask = 8'hF0; #10;
        wr_en = 0;
        search_en = 1; search_key = 8'hAB; #10;
        search_en = 0;

        #10 $finish;
    end
endmodule
```

7.SIMULATIONS

7.1Functional simulation:

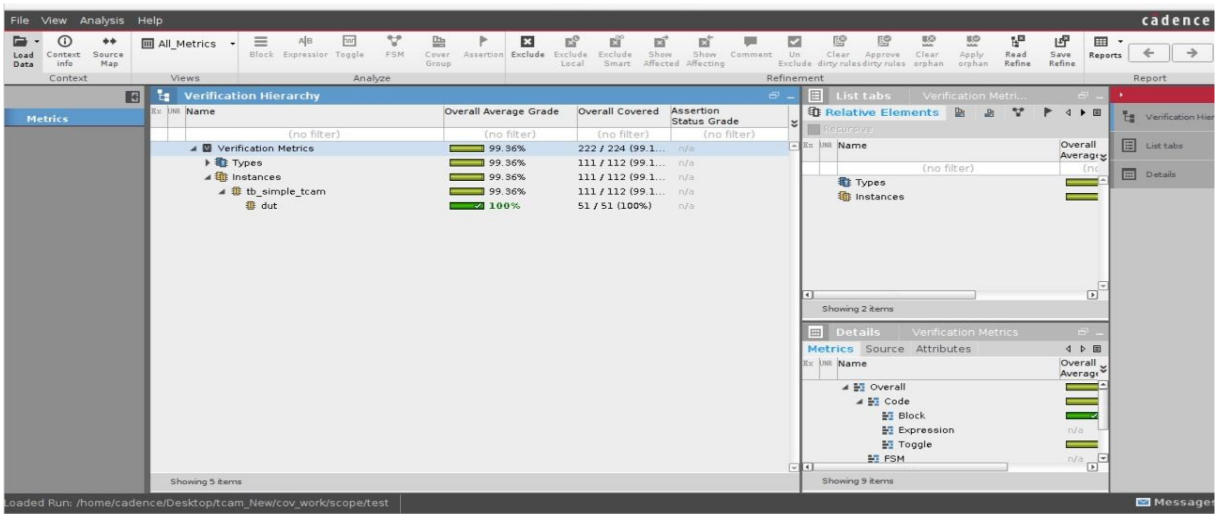
Functional simulation plays an essential role in VLSI design, ensuring that the RTL (Register Transfer Level) design performs correctly by running simulations with testbenches. This process helps to verify that the design operates as intended across different scenarios, detect any potential issues or bugs, and confirm the accuracy of state transitions. Conducting functional simulations enhances the overall quality of the design, minimizing risks and ensuring the design's reliability before physical implementation.



Name	Cursor
DEPTH	'd 4
WIDTH	'd 8
clk	1
match_data[7:0]	'h 00
match_found	0
rst	0
search_en	0
search_key[7:0]	'h AB
wr_addr[1:0]	'h 0
wr_data[7:0]	'h AA
wr_en	0
wr_mask[7:0]	'h F0

7.2. Code Coverage

Code coverage is an important metric that evaluates how much of the design's code is exercised during simulations or testing. It helps identify sections of the code that have not been tested, ensuring that all parts of the design are properly verified. Achieving high code coverage improves the quality of the tests, reduces the likelihood of undetected bugs, and ensures that the design is reliable and robust before it moves to the next stage of development.



Verification Hierarchy

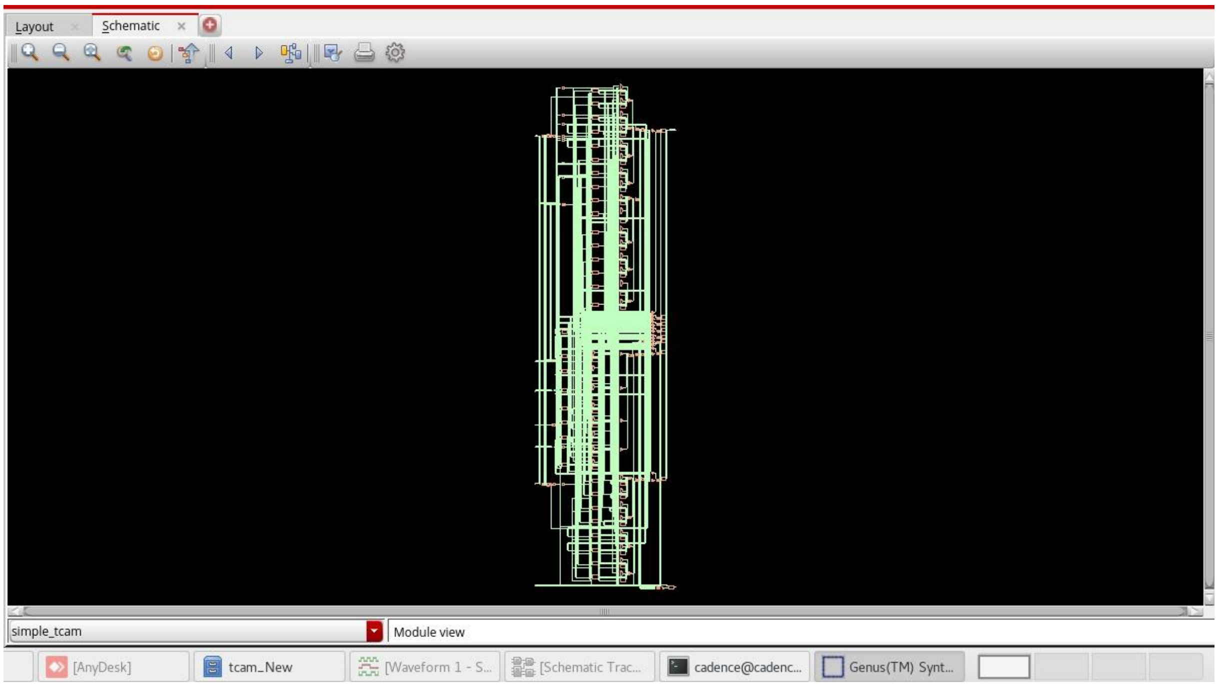
Name	Overall Average Grade	Overall Covered	Assertion Status Grade
Verification Metrics	99.36%	222 / 224 (99.1...	n/a
Types	99.36%	111 / 112 (99.1...	n/a
Instances	99.36%	111 / 112 (99.1...	n/a
tb_simple_tcam	99.36%	111 / 112 (99.1...	n/a
dut	100%	51 / 51 (100%)	n/a

Showing 5 items

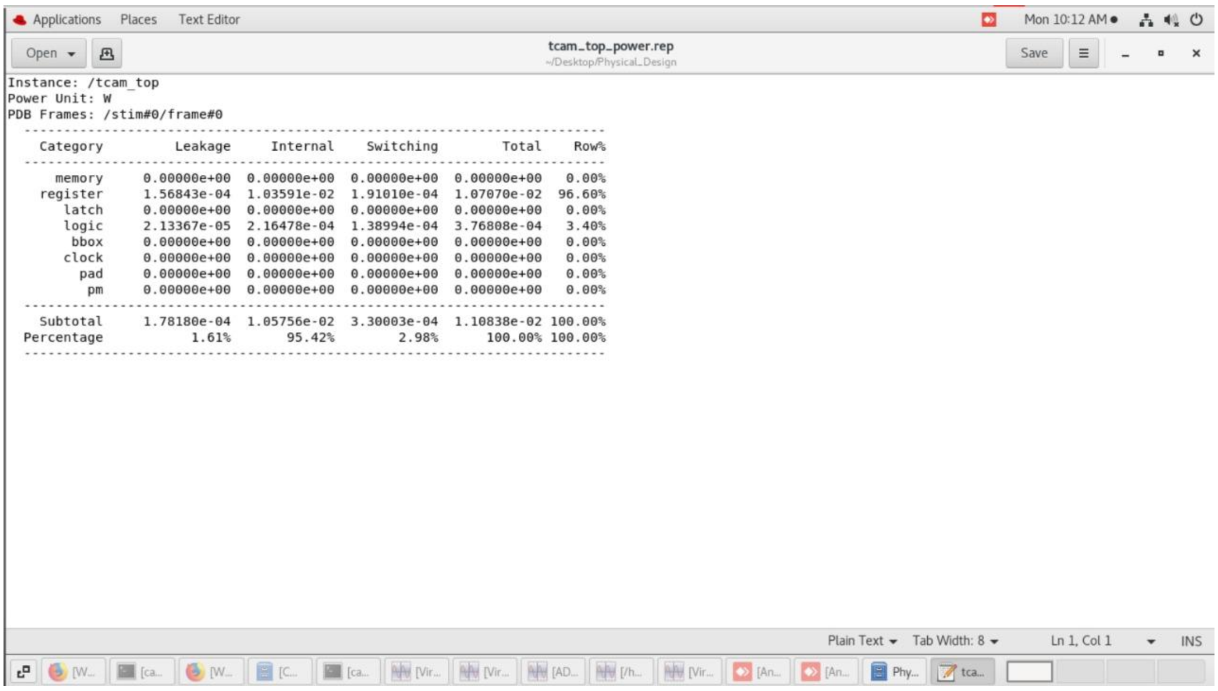
Loaded Run: /home/cadence/Desktop/tcam_New/cov_work/scope/test

7.3. Genus

Cadence Genus is an advanced synthesis tool that aids designers in optimizing digital designs for key parameters such as area, power, and timing. It offers sophisticated features for both logic and physical synthesis, along with design optimization techniques. This ensures the generation of a high-quality gate-level netlist that meets the desired specifications, performance standards, and efficiency requirements, contributing to the successful realization of the design.

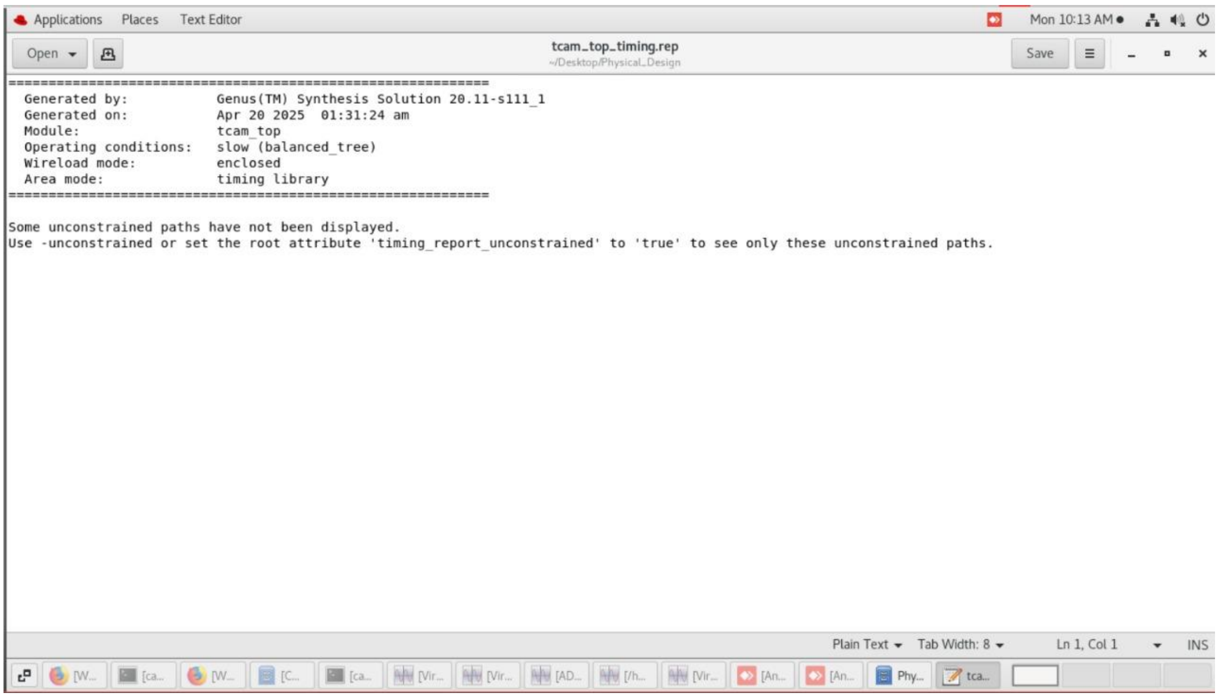


7.3.1 Power Report



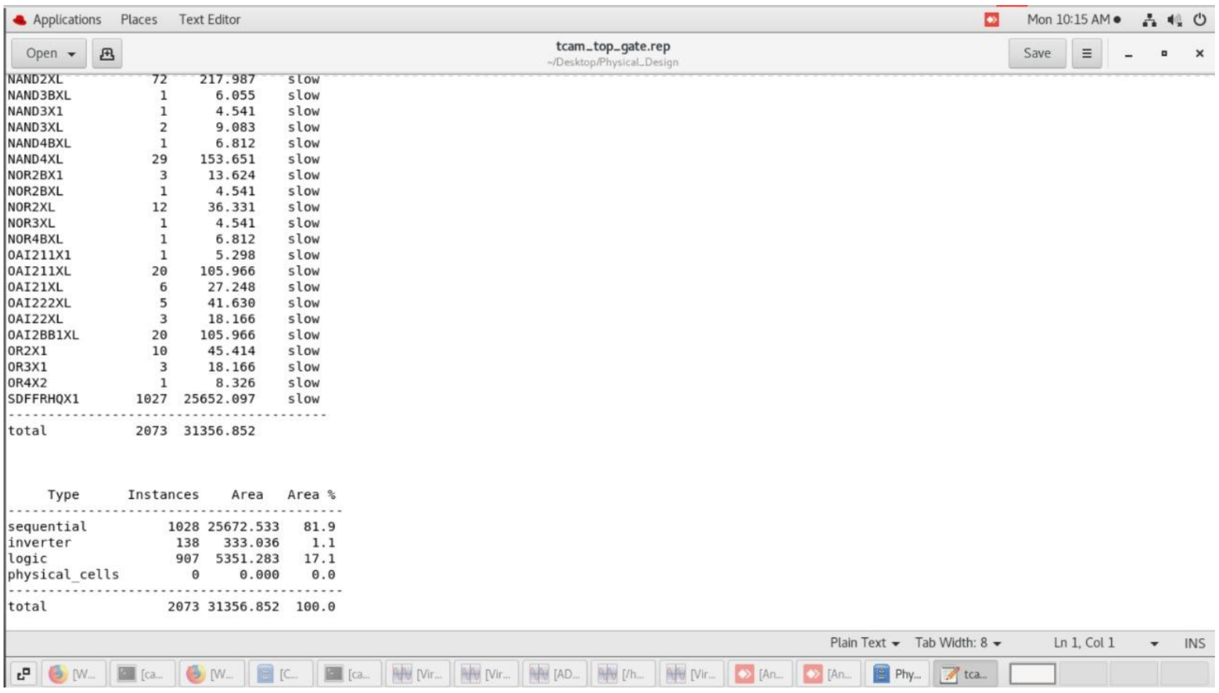
```
Instance: /tcam_top
Power Unit: W
PDB Frames: /stim#0/frame#0
  ---------------------------------------------------------------------------
    Category         Leakage     Internal    Switching        Total    Row%
  ---------------------------------------------------------------------------
      memory     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
    register     1.56843e-04  1.03591e-02  1.91010e-04  1.07070e-02  96.60%
       latch     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
       logic     2.13367e-05  2.16478e-04  1.38994e-04  3.76808e-04   3.40%
        bbox     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
       clock     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
         pad     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
          pm     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
  ---------------------------------------------------------------------------
    Subtotal     1.78180e-04  1.05756e-02  3.30003e-04  1.10838e-02 100.00%
  Percentage           1.61%       95.42%        2.98%      100.00% 100.00%
  ---------------------------------------------------------------------------
```

7.3.2 Timing Report



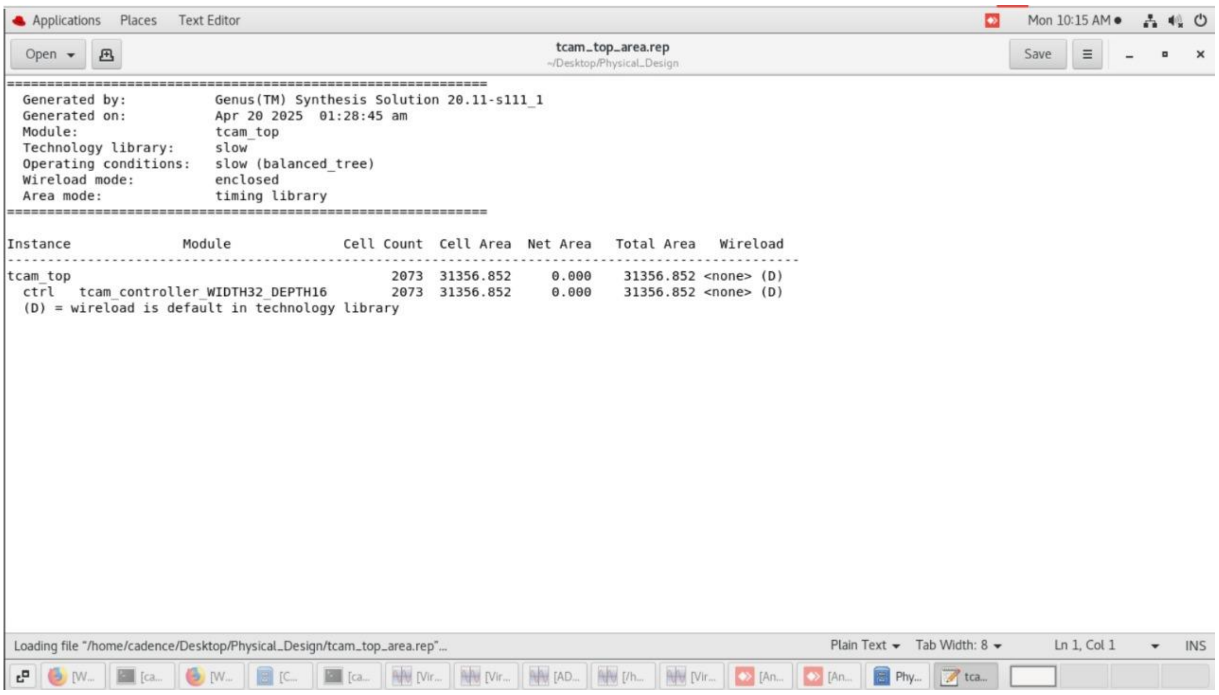
```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 20.11-s111_1
  Generated on:           Apr 20 2025  01:31:24 am
  Module:                 tcam_top
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

Some unconstrained paths have not been displayed.
Use -unconstrained or set the root attribute 'timing_report_unconstrained' to 'true' to see only these unconstrained paths.
```

7.3.3 Gate Report



```
NAND2XL          72    217.987    slow
NAND3BXL          1      6.055    slow
NAND3X1           1      4.541    slow
NAND3XL           2      9.083    slow
NAND4BXL          1      6.812    slow
NAND4XL          29    153.651    slow
NOR2BX1           3     13.624    slow
NOR2BXL           1      4.541    slow
NOR2XL           12     36.331    slow
NOR3XL            1      4.541    slow
NOR4BXL           1      6.812    slow
OAI211X1          1      5.298    slow
OAI211XL         20    105.966    slow
OAI21XL           6     27.248    slow
OAI222XL          5     41.630    slow
OAI22XL           3     18.166    slow
OAI2BB1XL        20    105.966    slow
OR2X1            10     45.414    slow
OR3X1             3     18.166    slow
OR4X2             1      8.326    slow
SDFFRHQX1      1027  25652.097    slow
----------------------------------------
total          2073  31356.852


     Type       Instances    Area    Area %
------------------------------------------
sequential           1028 25672.533   81.9
inverter              138   333.036    1.1
logic                 907  5351.283   17.1
physical_cells          0     0.000    0.0
------------------------------------------
total                2073 31356.852  100.0
```

7.3.4 Area Report

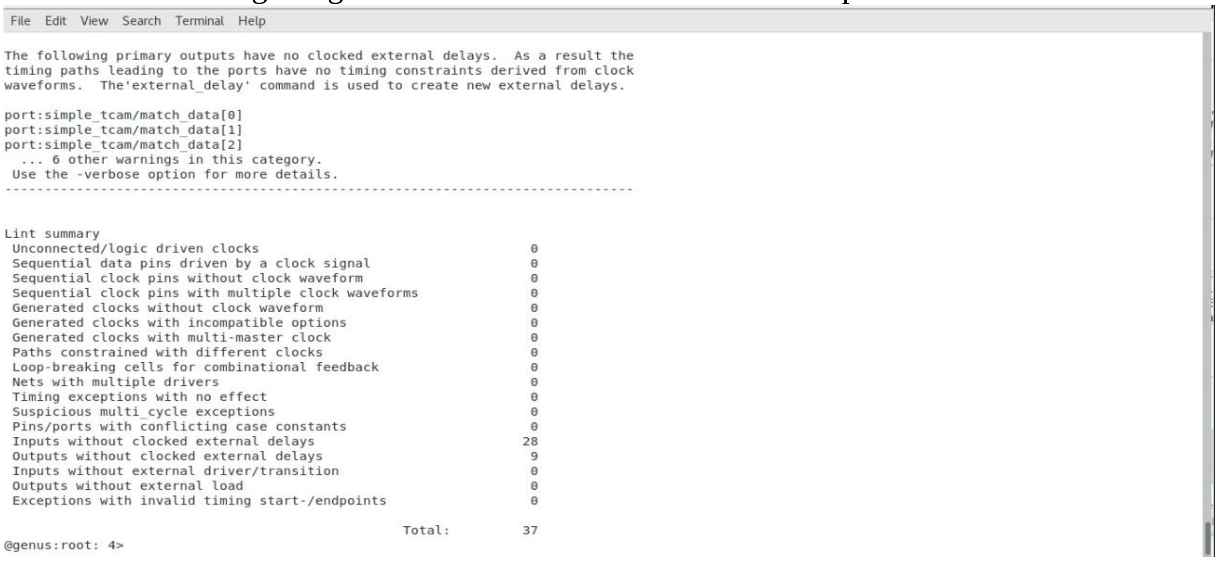


```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 20.11-s111_1
  Generated on:           Apr 20 2025  01:28:45 am
  Module:                 tcam_top
  Technology library:     slow
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

Instance             Module               Cell Count  Cell Area  Net Area   Total Area   Wireload
----------------------------------------------------------------------------------------------------
tcam_top                                        2073  31356.852     0.000    31356.852 <none> (D)
  ctrl    tcam_controller_WIDTH32_DEPTH16       2073  31356.852     0.000    31356.852 <none> (D)
  (D) = wireload is default in technology library
```

Loading file "/home/cadence/Desktop/Physical_Design/tcam_top_area.rep"...

7.4. Linting

Linting is an essential process in digital design that focuses on maintaining code quality by checking for syntax errors, style inconsistencies, and logical flaws. It helps identify potential issues early in the design process, ensuring cleaner and more reliable code. By reducing debugging efforts and promoting adherence to coding standards, linting enhances collaboration among designers and streamlines the overall development workflow.

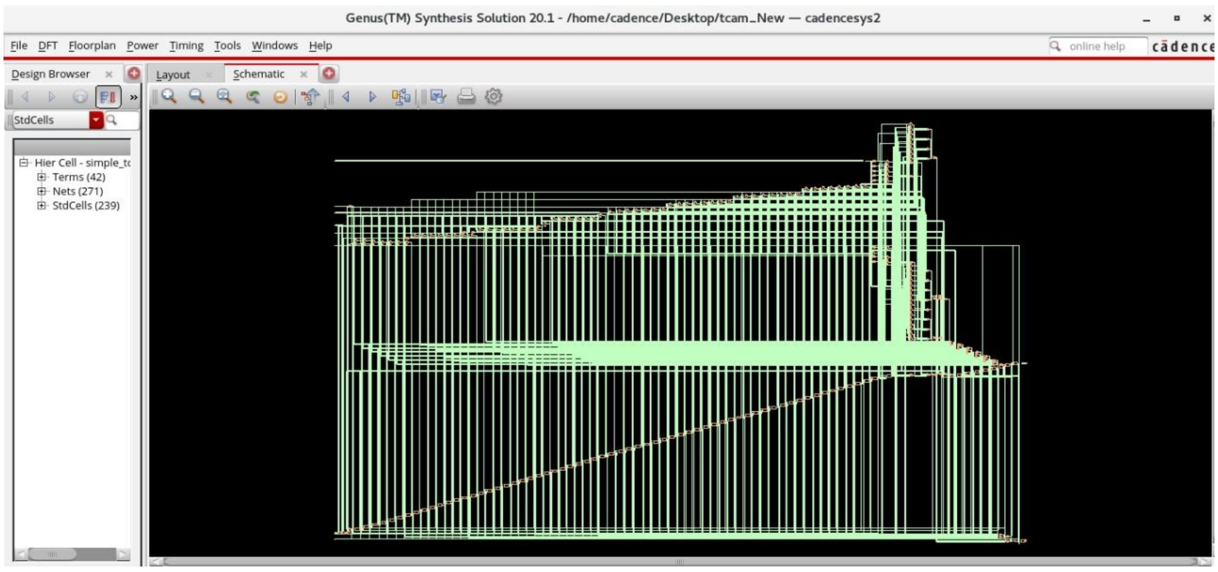


```
The following primary outputs have no clocked external delays.  As a result the
timing paths leading to the ports have no timing constraints derived from clock
waveforms.  The'external_delay' command is used to create new external delays.

port:simple_tcam/match_data[0]
port:simple_tcam/match_data[1]
port:simple_tcam/match_data[2]
  ... 6 other warnings in this category.
 Use the -verbose option for more details.
--------------------------------------------------------------------------------


Lint summary
 Unconnected/logic driven clocks                                    0
 Sequential data pins driven by a clock signal                      0
 Sequential clock pins without clock waveform                       0
 Sequential clock pins with multiple clock waveforms                0
 Generated clocks without clock waveform                            0
 Generated clocks with incompatible options                         0
 Generated clocks with multi-master clock                           0
 Paths constrained with different clocks                            0
 Loop-breaking cells for combinational feedback                     0
 Nets with multiple drivers                                         0
 Timing exceptions with no effect                                   0
 Suspicious multi_cycle exceptions                                  0
 Pins/ports with conflicting case constants                         0
 Inputs without clocked external delays                            28
 Outputs without clocked external delays                            9
 Inputs without external driver/transition                          0
 Outputs without external load                                      0
 Exceptions with invalid timing start-/endpoints                    0

                                                  Total:           37
@genus:root: 4>
```

7.5.DFT SYNTHESIS

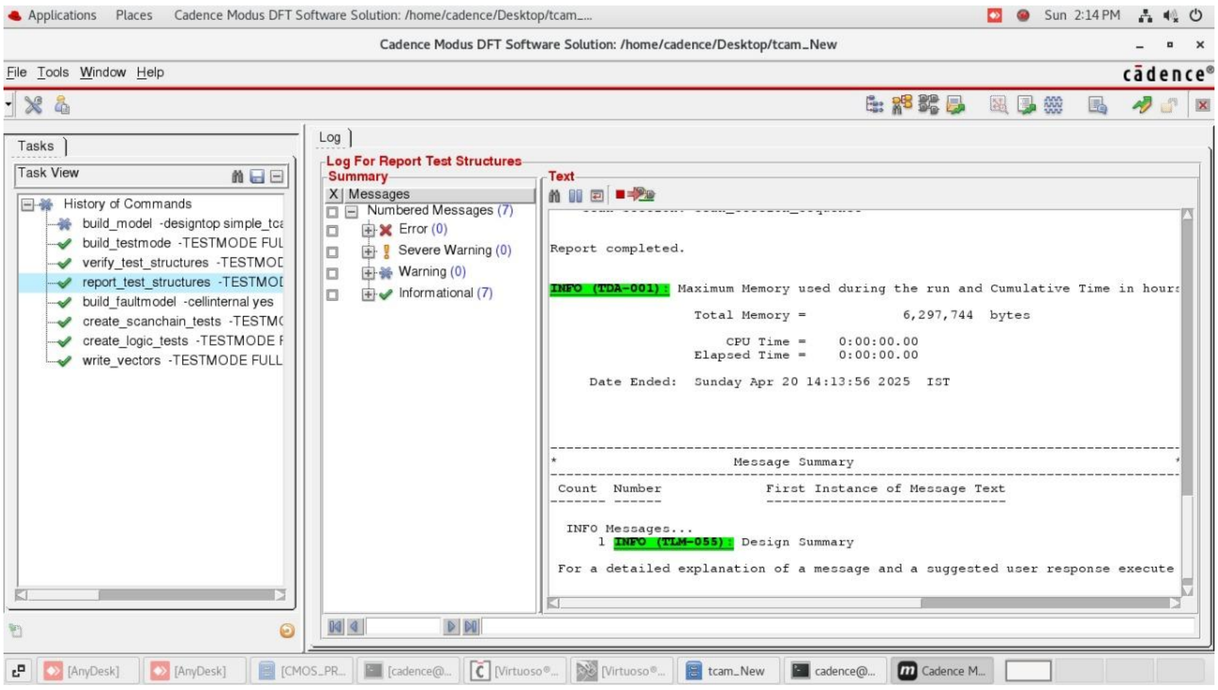
Design for Testability (DFT) is a methodology aimed at improving the testability of digital circuits by integrating features such as scan chains, boundary scan, and test points. These techniques help increase fault coverage, minimize testing time and costs, and ensure comprehensive verification of complex ASIC and FPGA designs. DFT is especially crucial in high-reliability applications, where thorough testing is necessary to ensure the functionality and robustness of the design before deployment.

7.5.1 Schematic after DFT


Genus(TM) Synthesis Solution 20.1 - /home/cadence/Desktop/tcam_New — cadencesys2
File DFT Floorplan Power Timing Tools Windows Help
online help
cadence
Design Browser
Layout
Schematic
StdCells
Hier Cell - simple_tc
Terms (42)
Nets (271)
StdCells (239)

7.6 ATPG

Automatic Test Pattern Generation (ATPG) is an essential method in digital design testing that automatically generates optimized test patterns to detect faults in the design. By improving fault coverage and minimizing test time and costs, ATPG ensures that complex digital systems, such as ASIC and FPGA designs, are thoroughly tested. This technique is critical for enhancing the reliability and quality of the design, ensuring that it meets performance standards and is free from defects before production.

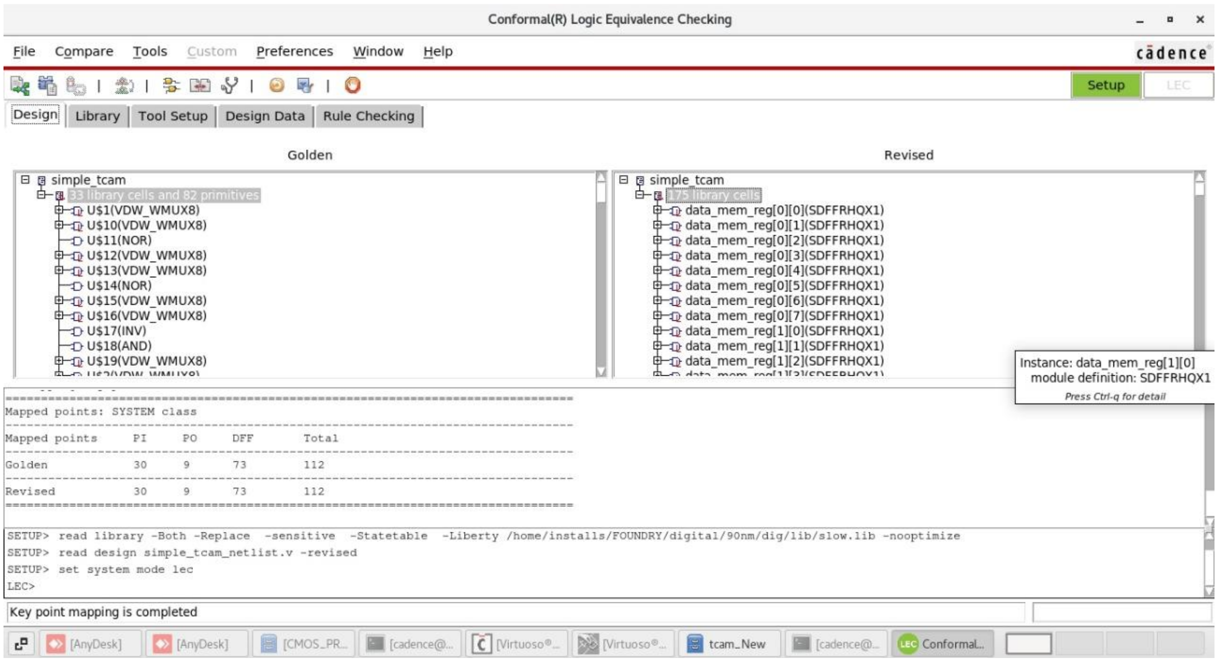


History of Commands
- build_model -designtop simple_tc
- build_testmode -TESTMODE FUL
- verify_test_structures -TESTMOD
- report_test_structures -TESTMOD
- build_faultmodel -cellinternal yes
- create_scanchain_tests -TESTMC
- create_logic_tests -TESTMODE F
- write_vectors -TESTMODE FULL

```
Report completed.

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

                  Total Memory =            6,297,744  bytes

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

     Date Ended:  Sunday Apr 20 14:13:56 2025  IST



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

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

 For a detailed explanation of a message and a suggested user response execute
```

7.7. LEC (Logical Equivalence checking)

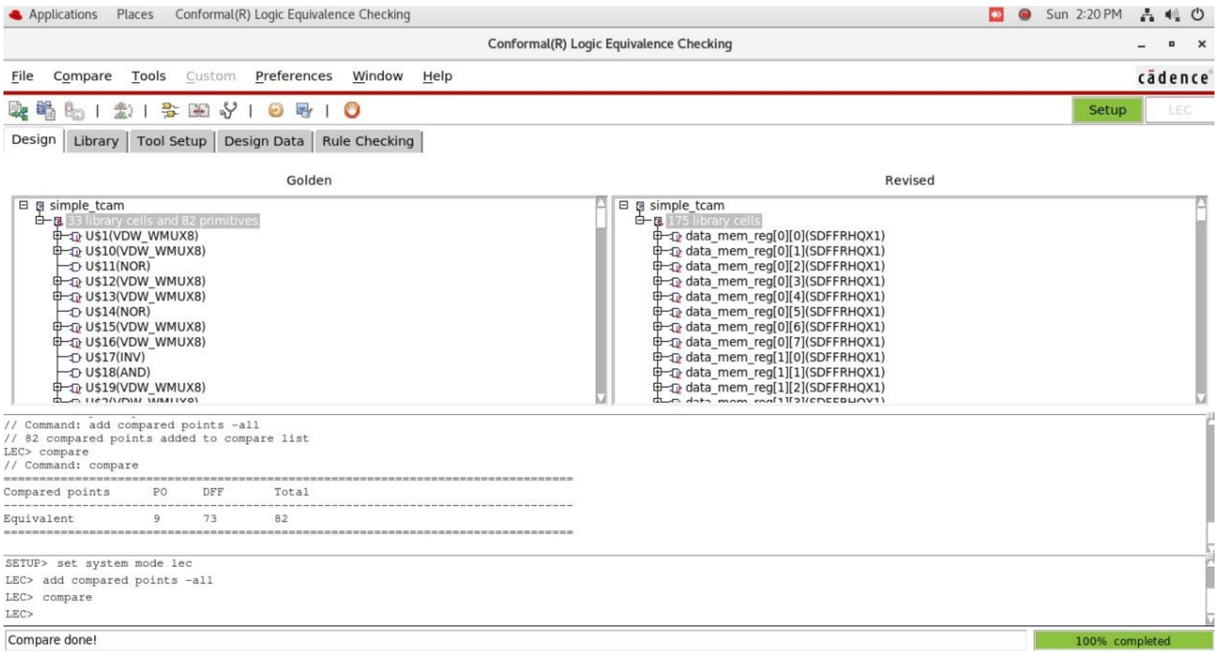
Logical Equivalence Checking (LEC) is a formal verification method used to compare two versions of a digital design—usually the RTL and its corresponding synthesized netlist—to ensure they are functionally identical. It verifies that no functional discrepancies have been introduced during synthesis or optimization, thereby ensuring the correctness, consistency, and reliability of the final design before proceeding to implementation.


Conformal(R) Logic Equivalence Checking
File Compare Tools Custom Preferences Window Help
cadence
Setup
LEC
Design Library Tool Setup Design Data Rule Checking
Golden
Revised
simple_tcam
33 library cells and 82 primitives
U$1(VDW_WMUX8)
U$10(VDW_WMUX8)
U$11(NOR)
U$12(VDW_WMUX8)
U$13(VDW_WMUX8)
U$14(NOR)
U$15(VDW_WMUX8)
U$16(VDW_WMUX8)
U$17(INV)
U$18(AND)
U$19(VDW_WMUX8)
simple_tcam
175 library cells
data_mem_reg[0][0](SDFFRHQX1)
data_mem_reg[0][1](SDFFRHQX1)
data_mem_reg[0][2](SDFFRHQX1)
data_mem_reg[0][3](SDFFRHQX1)
data_mem_reg[0][4](SDFFRHQX1)
data_mem_reg[0][5](SDFFRHQX1)
data_mem_reg[0][6](SDFFRHQX1)
data_mem_reg[0][7](SDFFRHQX1)
data_mem_reg[1][0](SDFFRHQX1)
data_mem_reg[1][1](SDFFRHQX1)
data_mem_reg[1][2](SDFFRHQX1)
Instance: data_mem_reg[1][0]
module definition: SDFFRHQX1
Press Ctrl-q for detail
[AnyDesk]
[AnyDesk]
[CMOS_PR...
[cadence@...
[Virtuoso®...
[Virtuoso®...
tcam_New
[cadence@...
Conformal...

```
================================================================================
Mapped points: SYSTEM class
--------------------------------------------------------------------------------
Mapped points     PI     PO     DFF      Total
--------------------------------------------------------------------------------
Golden            30     9      73       112
--------------------------------------------------------------------------------
Revised           30     9      73       112
================================================================================
```

```
SETUP> read library -Both -Replace  -sensitive  -Statetable  -Liberty /home/installs/FOUNDRY/digital/90nm/dig/lib/slow.lib -nooptimize
SETUP> read design simple_tcam_netlist.v -revised
SETUP> set system mode lec
LEC>
```

Key point mapping is completed


Applications Places Conformal(R) Logic Equivalence Checking
Sun 2:20 PM
Conformal(R) Logic Equivalence Checking
File Compare Tools Custom Preferences Window Help
cadence
Setup
LEC
Design Library Tool Setup Design Data Rule Checking
Golden
Revised
simple_tcam
33 library cells and 82 primitives
U$1(VDW_WMUX8)
U$10(VDW_WMUX8)
U$11(NOR)
U$12(VDW_WMUX8)
U$13(VDW_WMUX8)
U$14(NOR)
U$15(VDW_WMUX8)
U$16(VDW_WMUX8)
U$17(INV)
U$18(AND)
U$19(VDW_WMUX8)
simple_tcam
175 library cells
data_mem_reg[0][0](SDFFRHQX1)
data_mem_reg[0][1](SDFFRHQX1)
data_mem_reg[0][2](SDFFRHQX1)
data_mem_reg[0][3](SDFFRHQX1)
data_mem_reg[0][4](SDFFRHQX1)
data_mem_reg[0][5](SDFFRHQX1)
data_mem_reg[0][6](SDFFRHQX1)
data_mem_reg[0][7](SDFFRHQX1)
data_mem_reg[1][0](SDFFRHQX1)
data_mem_reg[1][1](SDFFRHQX1)
data_mem_reg[1][2](SDFFRHQX1)

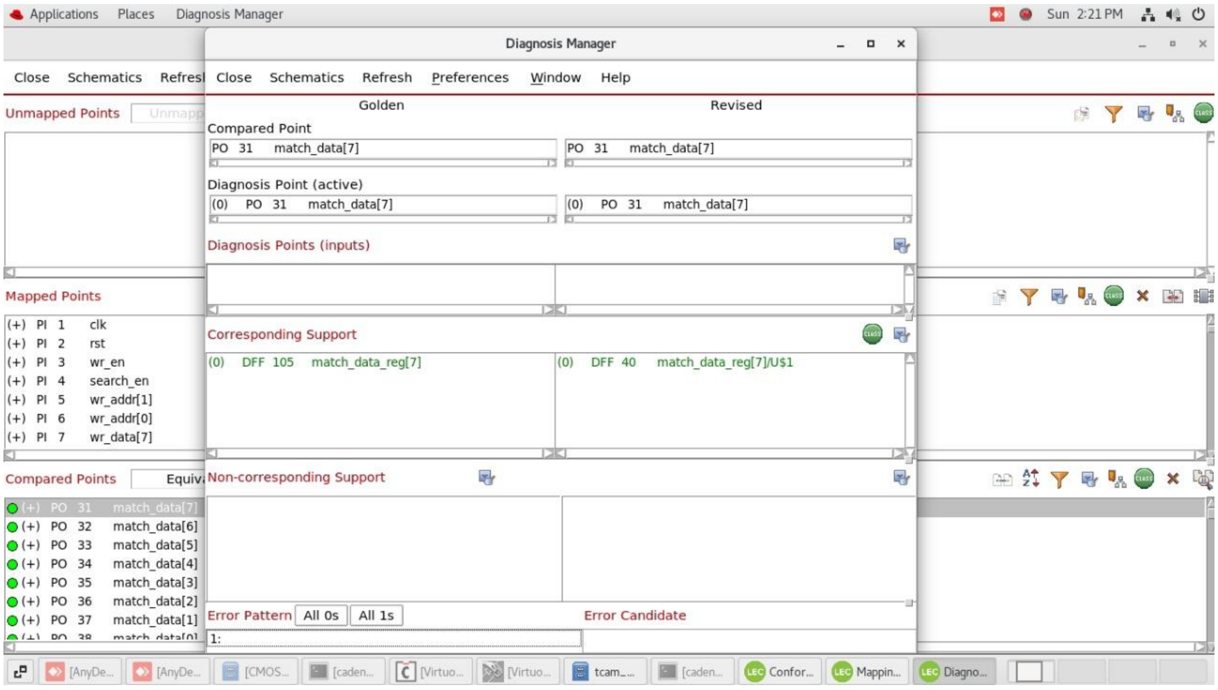
```
// Command: add compared points -all
// 82 compared points added to compare list
LEC> compare
// Command: compare
================================================================================
Compared points      PO     DFF      Total
--------------------------------------------------------------------------------
Equivalent           9      73       82
================================================================================
```

```
SETUP> set system mode lec
LEC> add compared points -all
LEC> compare
LEC>
```

Compare done!

100% completed

7.71 Diagnosis


Diagnosis Manager

Golden | Revised

Compared Point
PO 31 match_data[7] | PO 31 match_data[7]

Diagnosis Point (active)
(0) PO 31 match_data[7] | (0) PO 31 match_data[7]

Diagnosis Points (inputs)

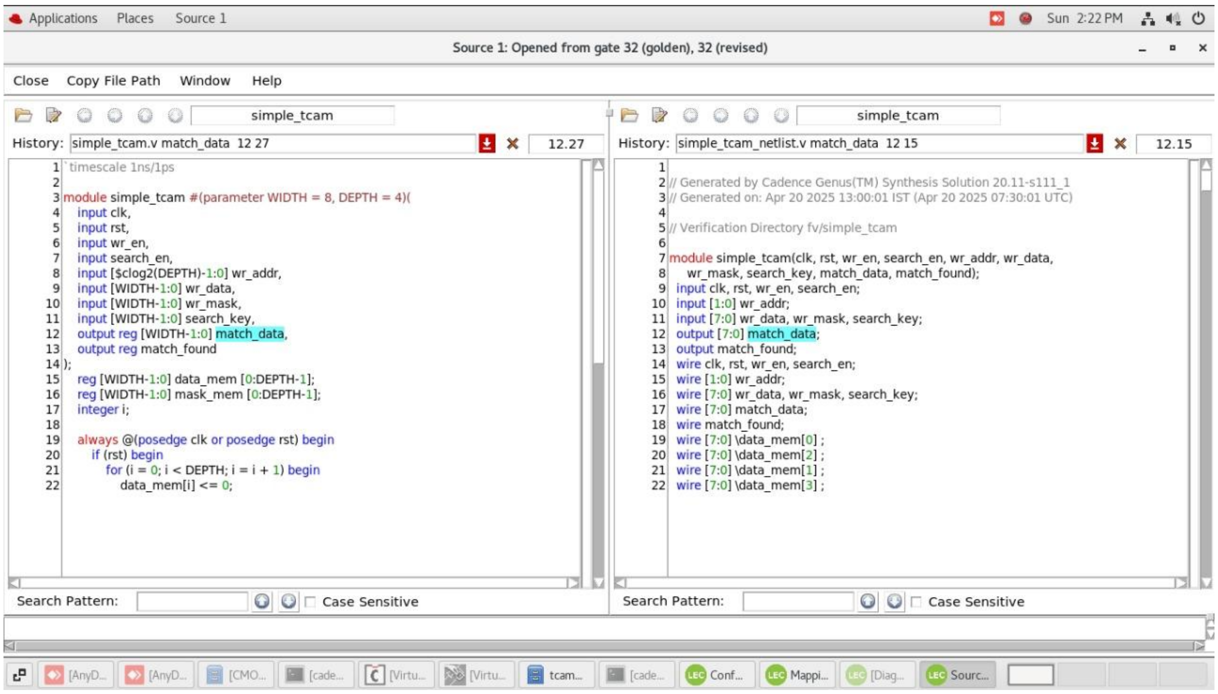
Corresponding Support
(0) DFF 105 match_data_reg[7] | (0) DFF 40 match_data_reg[7]/U$1

Non-corresponding Support

Error Pattern: All 0s / All 1s — 1:
Error Candidate

```
Mapped Points
(+) PI 1 clk
(+) PI 2 rst
(+) PI 3 wr_en
(+) PI 4 search_en
(+) PI 5 wr_addr[1]
(+) PI 6 wr_addr[0]
(+) PI 7 wr_data[7]

Compared Points
● (+) PO 31 match_data[7]
● (+) PO 32 match_data[6]
● (+) PO 33 match_data[5]
● (+) PO 34 match_data[4]
● (+) PO 35 match_data[3]
● (+) PO 36 match_data[2]
● (+) PO 37 match_data[1]
● (+) PO 38 match_data[0]
```

7.72 Source code


Source 1: Opened from gate 32 (golden), 32 (revised)

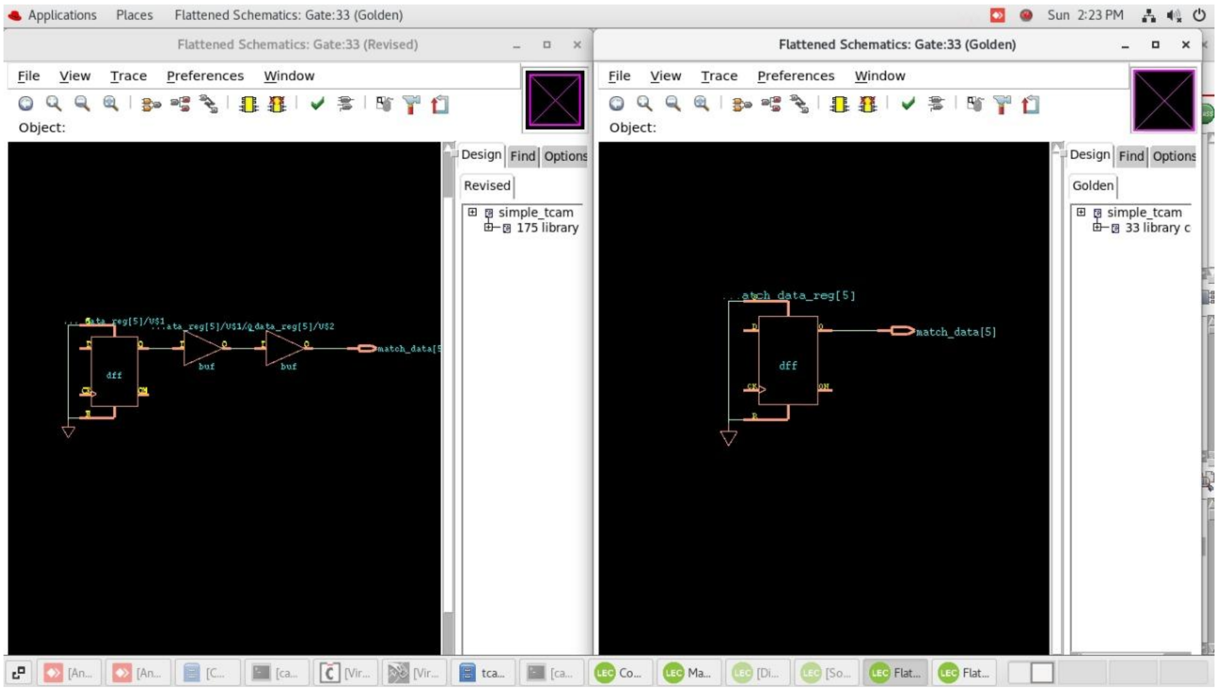
simple_tcam — History: simple_tcam.v match_data 12 27 — 12.27

```
`timescale 1ns/1ps

module simple_tcam #(parameter WIDTH = 8, DEPTH = 4)(
   input clk,
   input rst,
   input wr_en,
   input search_en,
   input [$clog2(DEPTH)-1:0] wr_addr,
   input [WIDTH-1:0] wr_data,
   input [WIDTH-1:0] wr_mask,
   input [WIDTH-1:0] search_key,
   output reg [WIDTH-1:0] match_data,
   output reg match_found
);
   reg [WIDTH-1:0] data_mem [0:DEPTH-1];
   reg [WIDTH-1:0] mask_mem [0:DEPTH-1];
   integer i;

   always @(posedge clk or posedge rst) begin
      if (rst) begin
         for (i = 0; i < DEPTH; i = i + 1) begin
            data_mem[i] <= 0;
```

simple_tcam — History: simple_tcam_netlist.v match_data 12 15 — 12.15

```

// Generated by Cadence Genus(TM) Synthesis Solution 20.11-s111_1
// Generated on: Apr 20 2025 13:00:01 IST (Apr 20 2025 07:30:01 UTC)

// Verification Directory fv/simple_tcam

module simple_tcam(clk, rst, wr_en, search_en, wr_addr, wr_data,
    wr_mask, search_key, match_data, match_found);
  input clk, rst, wr_en, search_en;
  input [1:0] wr_addr;
  input [7:0] wr_data, wr_mask, search_key;
  output [7:0] match_data;
  output match_found;
  wire clk, rst, wr_en, search_en;
  wire [1:0] wr_addr;
  wire [7:0] wr_data, wr_mask, search_key;
  wire [7:0] match_data;
  wire match_found;
  wire [7:0] \data_mem[0] ;
  wire [7:0] \data_mem[2] ;
  wire [7:0] \data_mem[1] ;
  wire [7:0] \data_mem[3] ;
```

7.73 Schematics

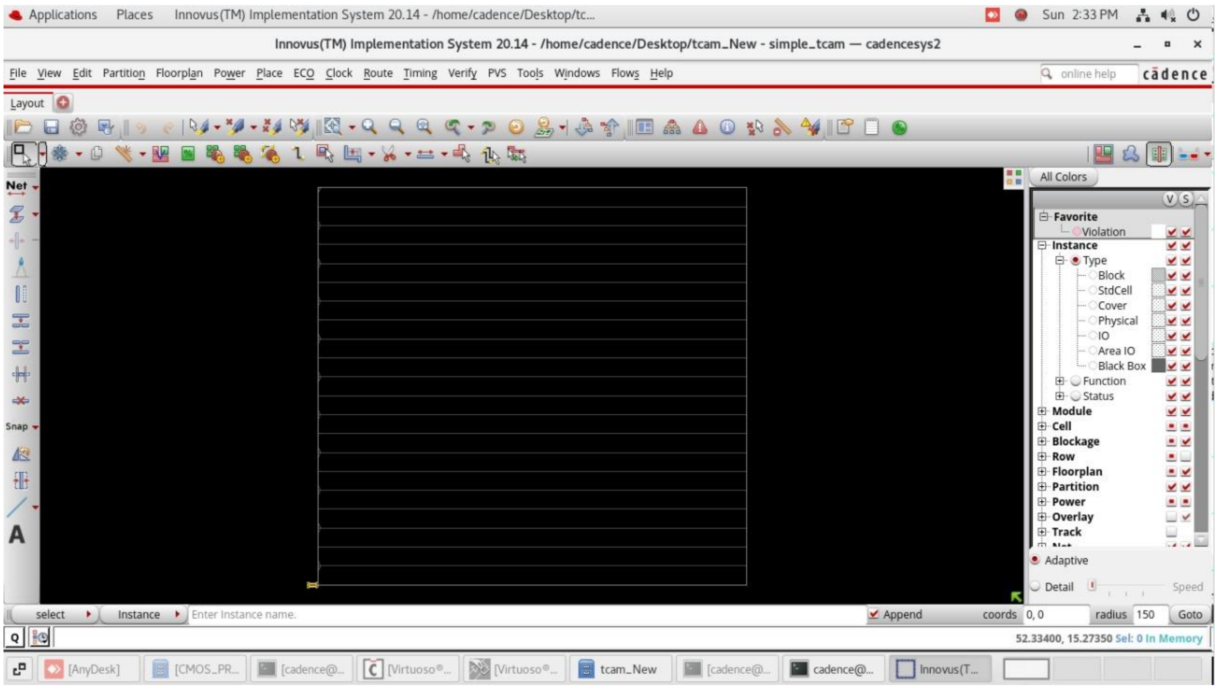


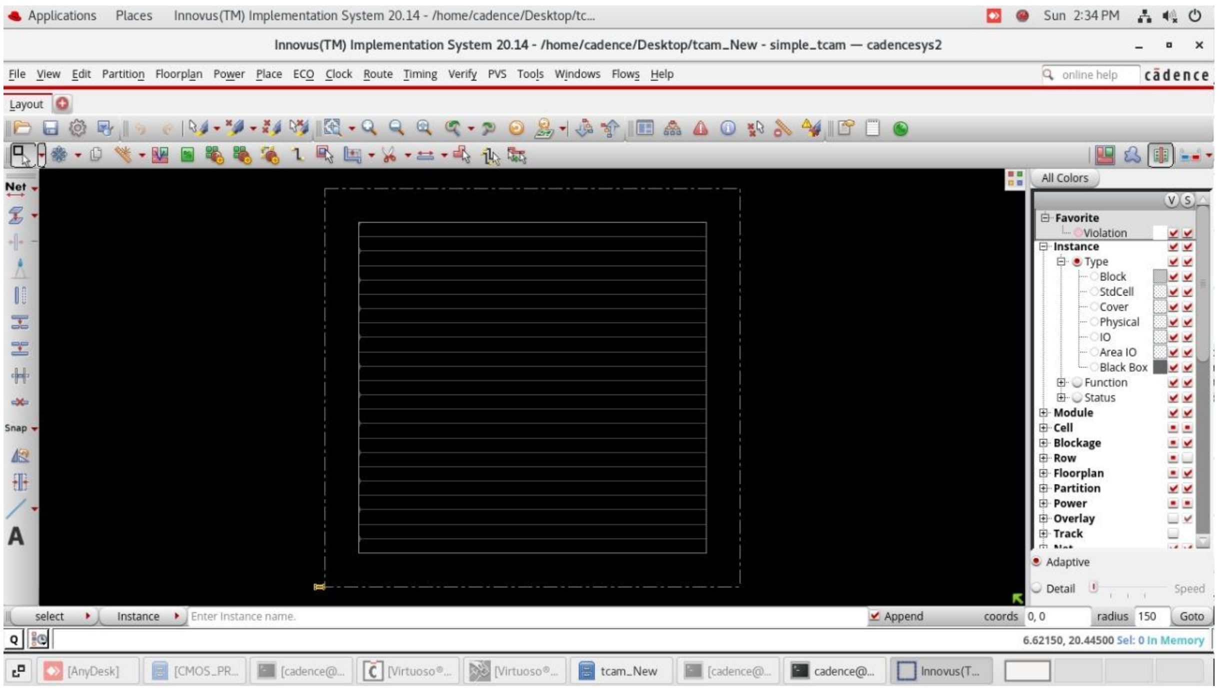
7.8. Physical Design

Physical Design is a key backend process in the VLSI design flow where the synthesized netlist is translated into a physical layout ready for chip fabrication. For the clock glitch detector, Cadence Innovus was used to carry out this process, starting with floorplanning to define the placement of major components. This was followed by standard cell placement and Clock Tree Synthesis (CTS) to distribute the clock signal efficiently, minimizing skew and satisfying timing requirements. Routing connected all components using appropriate metal layers, focusing on minimizing delay and maintaining signal integrity. Design Rule Checks (DRC) and Layout Versus Schematic (LVS) checks were performed to ensure compliance with manufacturing constraints and logical correctness. Finally, Static Timing Analysis (STA) was repeated after routing to verify that timing constraints, including setup and hold times, were met. This phase ensures the design is not only functionally accurate but also physically realizable, timing-closed, and ready for fabrication

7.8.1 Floor planning

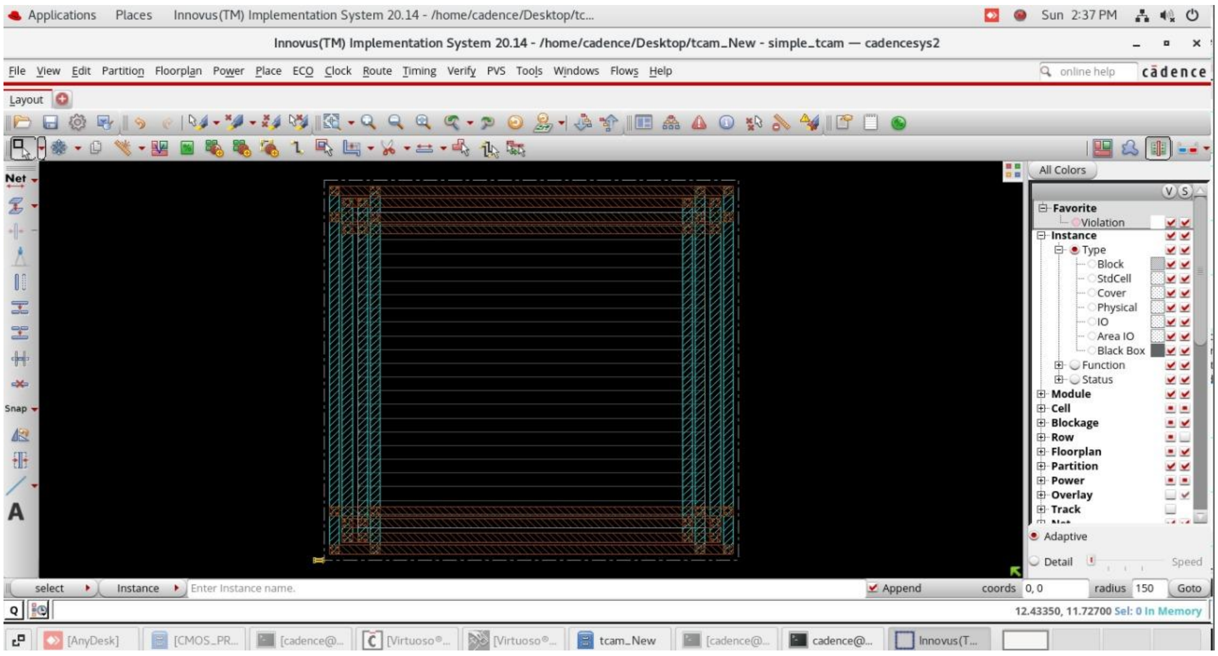
Floorplanning is a foundational step in physical design where major functional blocks of the chip are strategically arranged to achieve optimal layout efficiency and meet performance goals. In the case of the clock glitch detector, this involved organizing components such as the FSM controller, timer modules, and pedestrian/emergency logic within the chip’s core area in a well-structured manner. The objective was to minimize interconnect lengths, reduce propagation delays, and make efficient use of silicon area. Important considerations during this stage included the positioning of I/O ports, clock and power routing paths, and maintaining adequate spacing between critical modules. An effective floorplan lays the groundwork for easier routing, reduces congestion, and supports meeting both timing and power targets in subsequent stages.


Innovus(TM) Implementation System 20.14 - /home/cadence/Desktop/tcam_New - simple_tcam — cadencesys2
File View Edit Partition Floorplan Power Place ECO Clock Route Timing Verify PVS Tools Windows Flows Help
Layout
All Colors
Favorite
Violation
Instance
Type
Block
StdCell
Cover
Physical
IO
Area IO
Black Box
Function
Status
Module
Cell
Blockage
Row
Floorplan
Partition
Power
Overlay
Track
Adaptive
Detail
Speed
select
Instance
Enter Instance name.
Append
coords 0, 0
radius 150
Goto
52.33400, 15.27350 Sel: 0 In Memory


Innovus(TM) Implementation System 20.14 - /home/cadence/Desktop/tcam_New - simple_tcam — cadencesys2
File View Edit Partition Floorplan Power Place ECO Clock Route Timing Verify PVS Tools Windows Flows Help
Layout
All Colors
Favorite
Violation
Instance
Type
Block
StdCell
Cover
Physical
IO
Area IO
Black Box
Function
Status
Module
Cell
Blockage
Row
Floorplan
Partition
Power
Overlay
Track
Adaptive
Detail
Speed
select
Instance
Enter Instance name.
Append
coords 0, 0
radius 150
Goto
6.62150, 20.44500 Sel: 0 In Memory

7.8.2 Power Planning

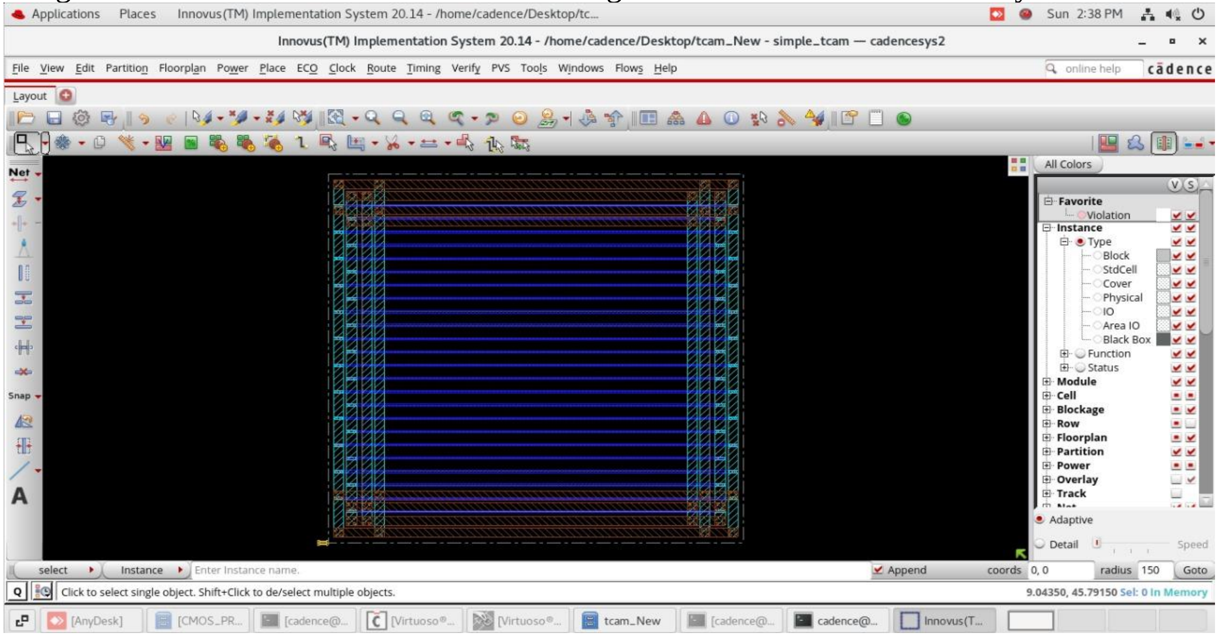
Power Planning ensures reliable power delivery across the chip. For the clock glitch detector, a power grid was created to distribute VDD and GND efficiently, with attention to avoiding voltage drops and overheating during switching events. Decoupling capacitors were used to manage power fluctuations and reduce noise. Effective power planning improves the chip's reliability, performance, and energy efficiency.



Applications
Places
Innovus(TM) Implementation System 20.14 - /home/cadence/Desktop/tc...
Sun 2:37 PM
Innovus(TM) Implementation System 20.14 - /home/cadence/Desktop/tcam_New - simple_tcam — cadencesys2
File View Edit Partition Floorplan Power Place ECO Clock Route Timing Verify PVS Tools Windows Flows Help
online help
cādence
Layout
All Colors
Favorite
Violation
Instance
Type
Block
StdCell
Cover
Physical
IO
Area IO
Black Box
Function
Status
Module
Cell
Blockage
Row
Floorplan
Partition
Power
Overlay
Track
Adaptive
Detail
Speed
select
Instance
Enter Instance name.
Append
coords 0, 0
radius 150
Goto
12.43350, 11.72700 Sel: 0 In Memory

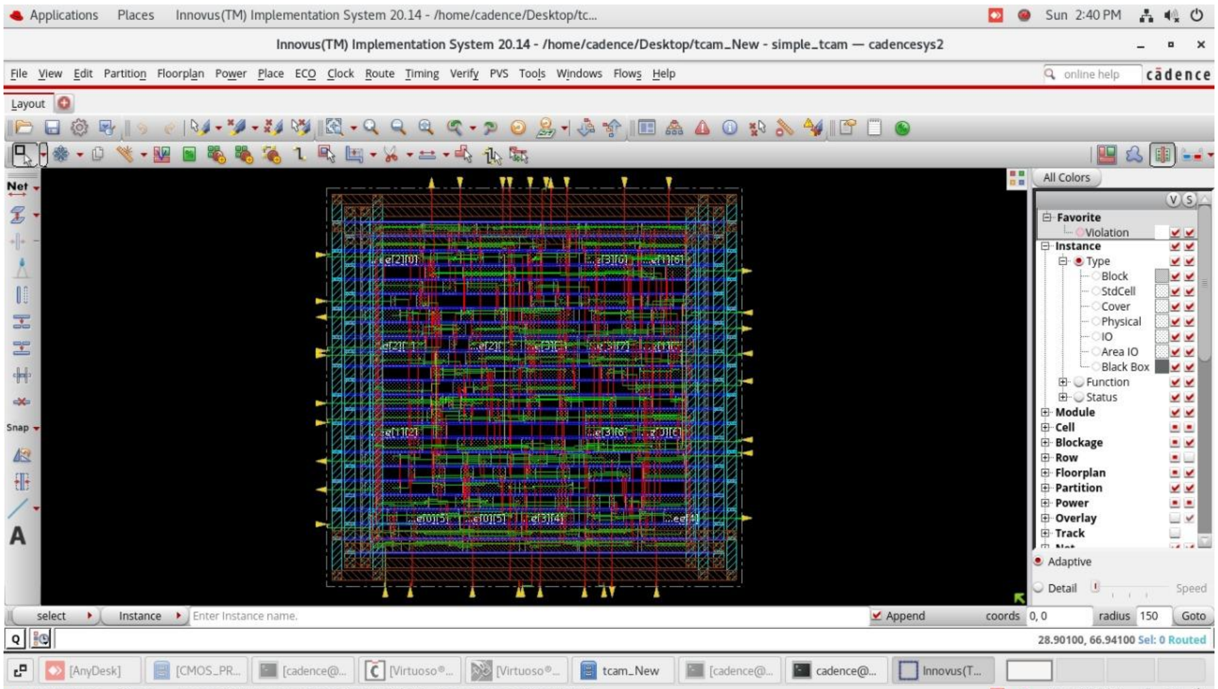
7.8.3 Placement

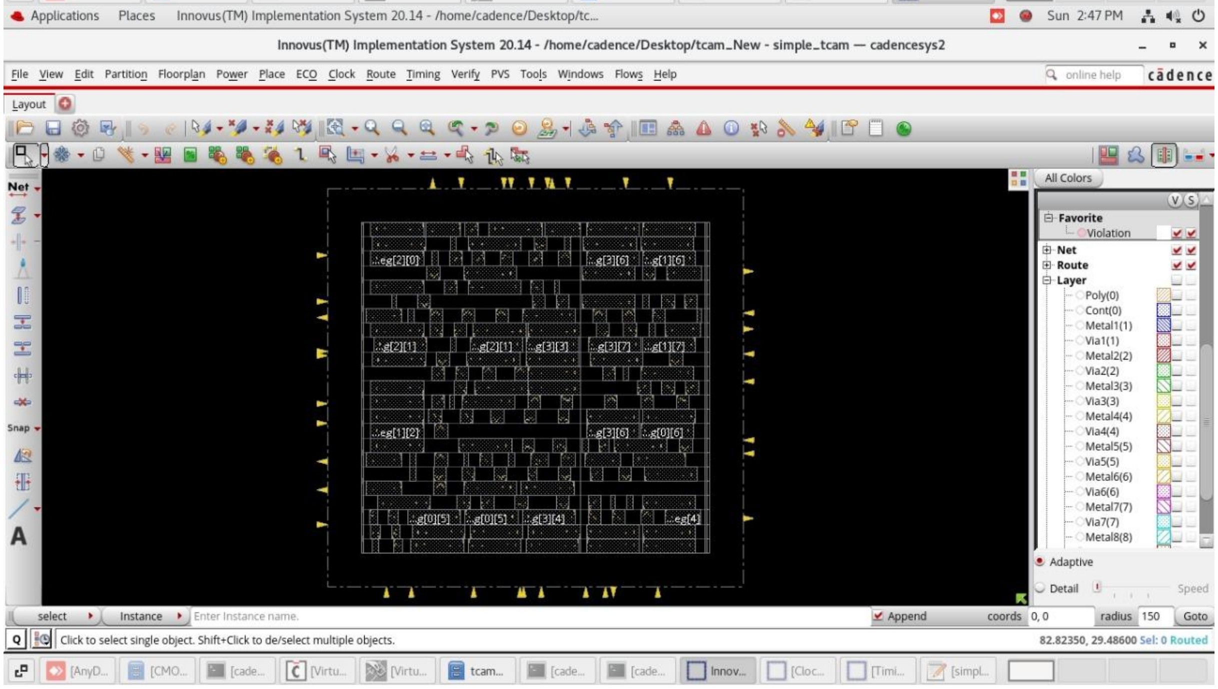
Placement arranges all standard cells of the clock glitch detector on the chip layout based on the floorplan. Cells are positioned in rows without overlap, keeping related logic close to reduce wire length and delay. Though routing isn't done yet, placement ensures enough space and optimizes timing, power, and area. Automated tools help meet design constraints like timing and clock synchronization.



Click to select single object. Shift+Click to de/select multiple objects.

9.04350, 45.79150 Sel: 0 In Memory

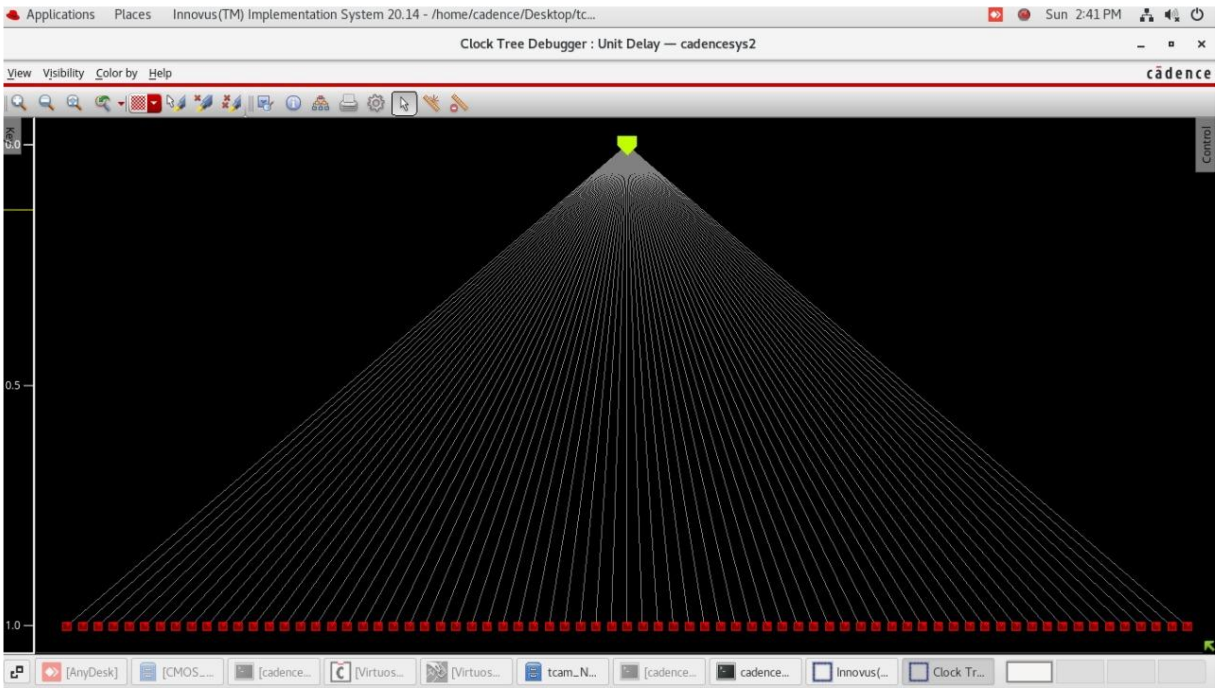




Click to select single object. Shift+Click to de/select multiple objects.

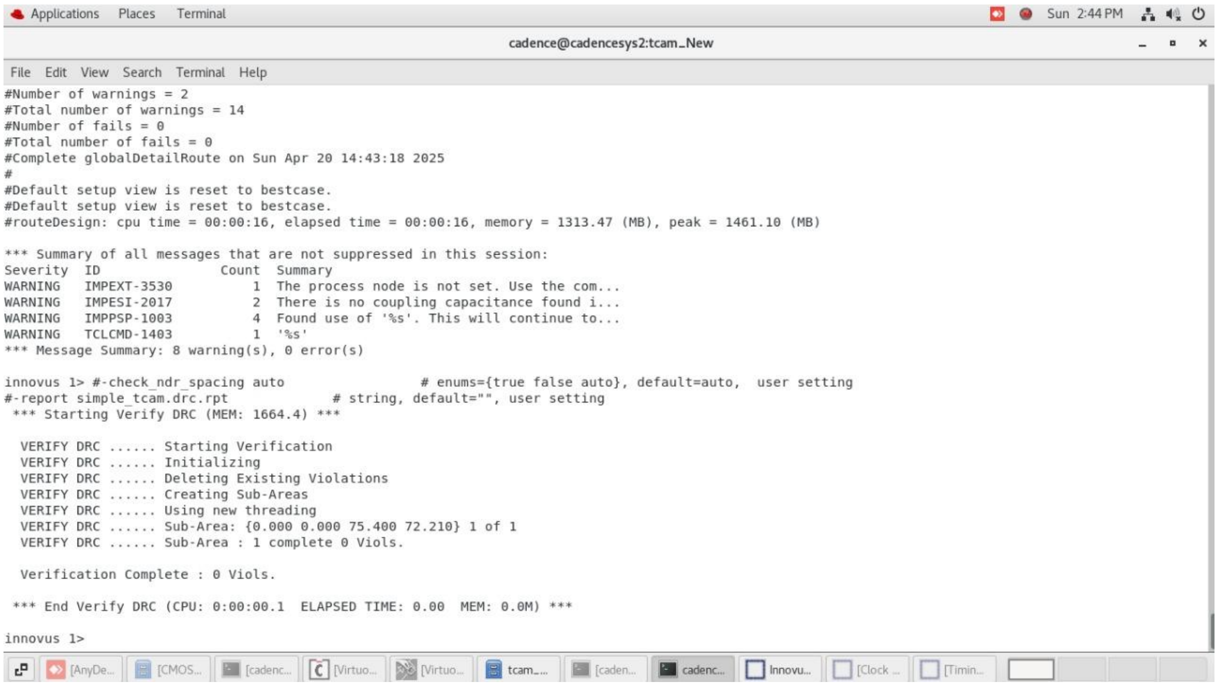
7.8.4 CTS (Clock tree synthesis)

Clock Tree Synthesis (CTS) distributes the clock signal uniformly to all sequential elements in the clock glitch detector. It minimizes clock skew and latency by inserting buffers/inverters in a tree structure. A balanced clock tree ensures synchronized data flow, enhancing the circuit's reliability and performance.



7.8.5 Verify DRC

Design Rule Checking (DRC) verification ensures a design meets manufacturing rules, checking for errors like minimum spacing, width, enclosure, overlap, and density, to prevent manufacturing issues, guarantee design reliability, and optimize yield.

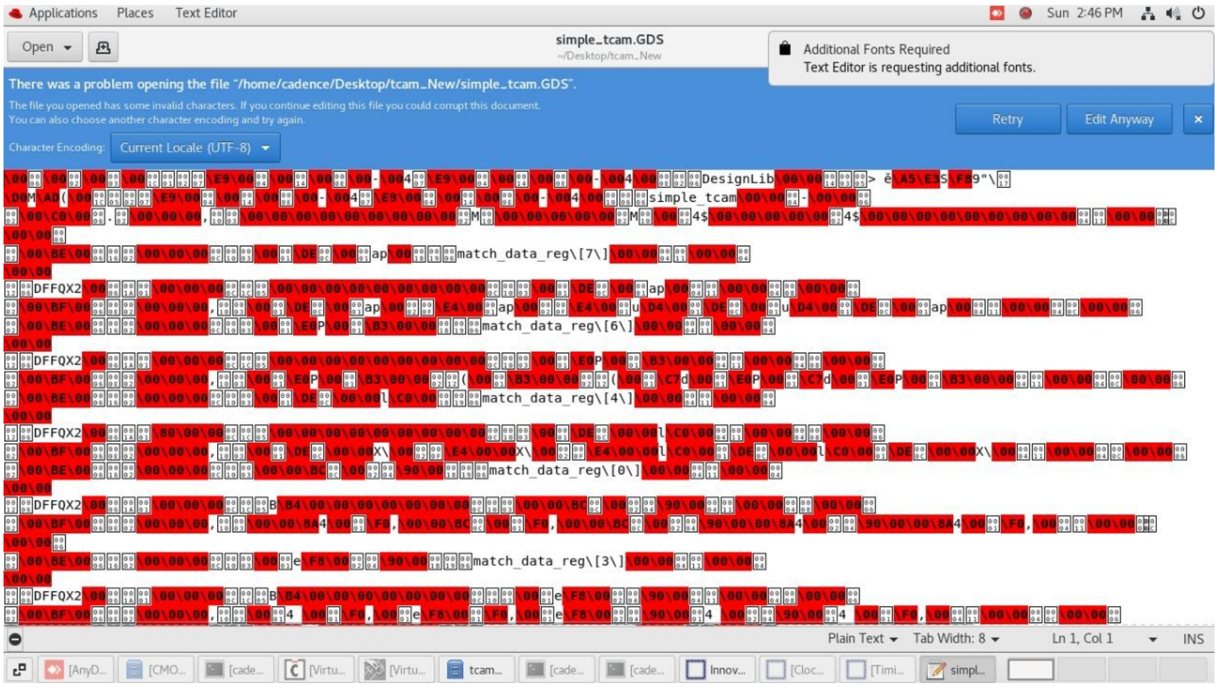


Applications Places Terminal
Sun 2:44 PM
cadence@cadencesys2:tcam_New
File Edit View Search Terminal Help

```
#Number of warnings = 2
#Total number of warnings = 14
#Number of fails = 0
#Total number of fails = 0
#Complete globalDetailRoute on Sun Apr 20 14:43:18 2025
#
#Default setup view is reset to bestcase.
#Default setup view is reset to bestcase.
#routeDesign: cpu time = 00:00:16, elapsed time = 00:00:16, memory = 1313.47 (MB), peak = 1461.10 (MB)

*** Summary of all messages that are not suppressed in this session:
Severity  ID               Count  Summary
WARNING   IMPEXT-3530          1  The process node is not set. Use the com...
WARNING   IMPESI-2017          2  There is no coupling capacitance found i...
WARNING   IMPPSP-1003          4  Found use of '%s'. This will continue to...
WARNING   TCLCMD-1403          1  '%s'
*** Message Summary: 8 warning(s), 0 error(s)

innovus 1> #-check_ndr_spacing auto                 # enums={true false auto}, default=auto,  user setting
#-report simple_tcam.drc.rpt             # string, default="", user setting
 *** Starting Verify DRC (MEM: 1664.4) ***

  VERIFY DRC ...... Starting Verification
  VERIFY DRC ...... Initializing
  VERIFY DRC ...... Deleting Existing Violations
  VERIFY DRC ...... Creating Sub-Areas
  VERIFY DRC ...... Using new threading
  VERIFY DRC ...... Sub-Area: {0.000 0.000 75.400 72.210} 1 of 1
  VERIFY DRC ...... Sub-Area : 1 complete 0 Viols.

  Verification Complete : 0 Viols.

 *** End Verify DRC (CPU: 0:00:00.1  ELAPSED TIME: 0.00  MEM: 0.0M) ***

innovus 1>
```

7.8.6 GDC (Global Design checks)

Global Design Checks (GDC) or Geometric Design Checking ensures design integrity by verifying compliance with geometric and design rules, identifying potential issues to guarantee design reliability and manufacturability.



Additional Fonts Required
Text Editor is requesting additional fonts.

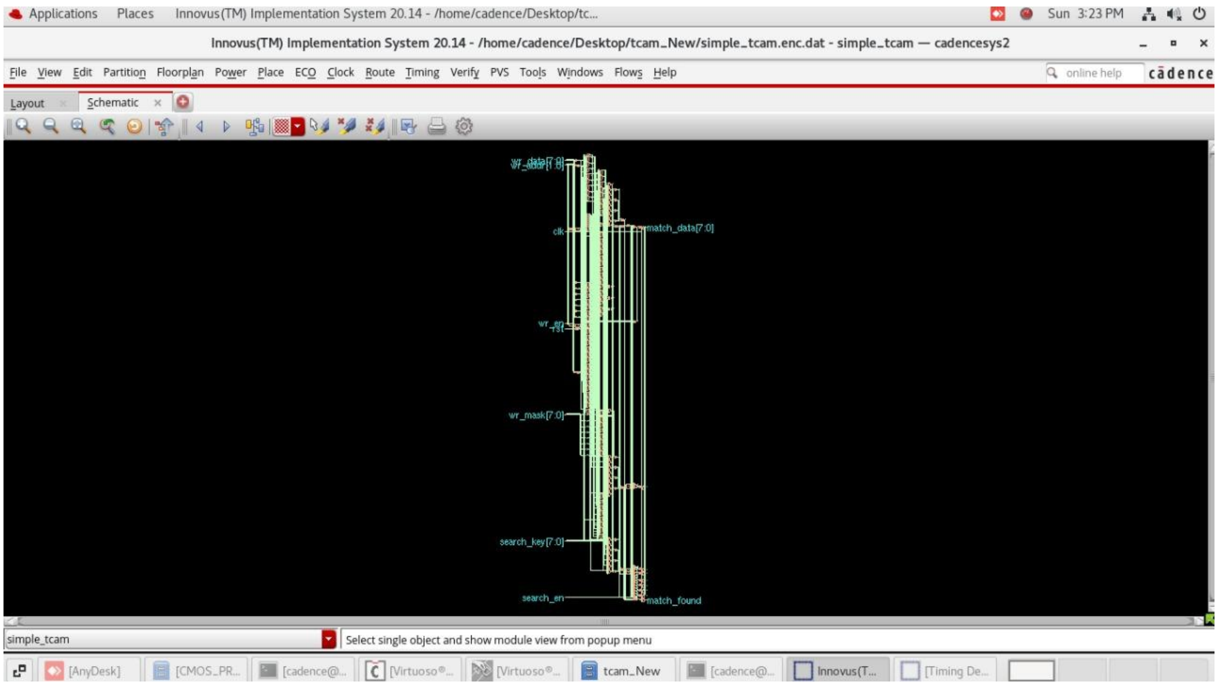
There was a problem opening the file "/home/cadence/Desktop/tcam_New/simple_tcam.GDS".

The file you opened has some invalid characters. If you continue editing this file you could corrupt this document.
You can also choose another character encoding and try again.

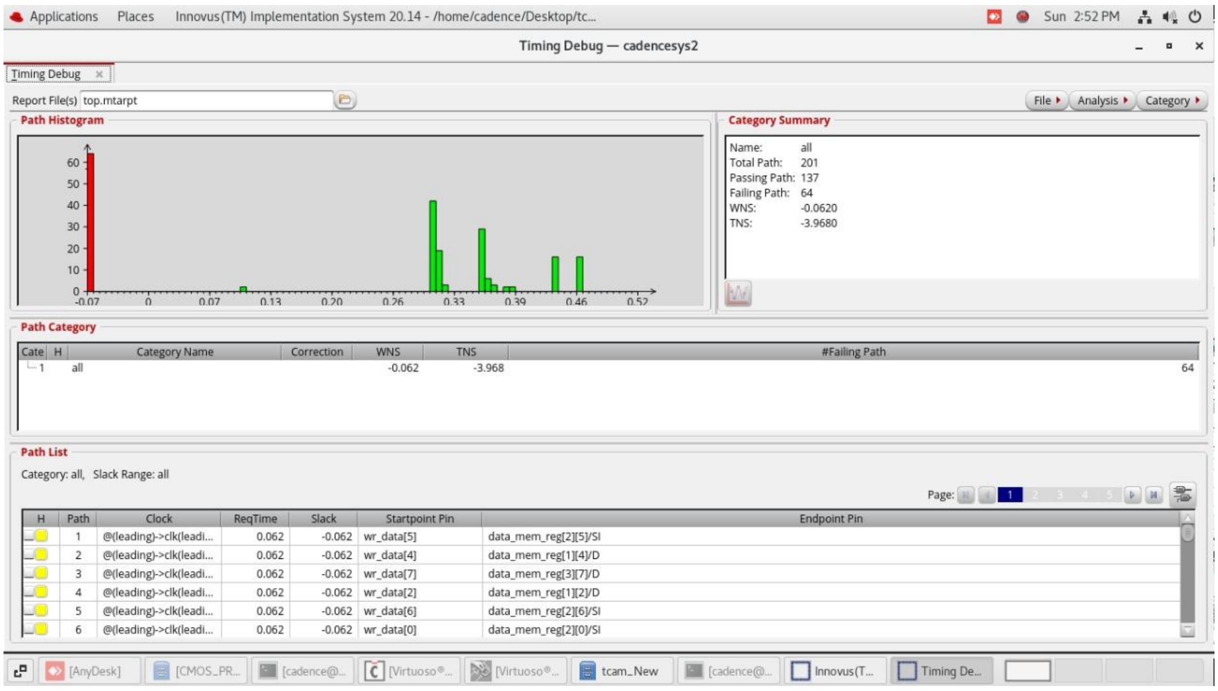
```
DesignLib
simple_tcam
match_data_reg\[7\]
DFFQX2
match_data_reg\[6\]
DFFQX2
match_data_reg\[4\]
DFFQX2
match_data_reg\[0\]
DFFQX2
match_data_reg\[3\]
DFFQX2
```

7.9. STA (Static Timing Analysis)

Static Timing Analysis (STA) verifies that a design meets timing constraints without using input vectors. For the clock glitch detector, STA was done after synthesis and placement to check setup/hold violations and analyze all signal paths. Tools like PrimeTime ensure timing reliability under worst-case PVT conditions, confirming the controller's accurate and stable operation.



Applications
Places
Innovus(TM) Implementation System 20.14 - /home/cadence/Desktop/tc...
Sun 3:23 PM
Innovus(TM) Implementation System 20.14 - /home/cadence/Desktop/tcam_New/simple_tcam.enc.dat - simple_tcam — cadencesys2
File View Edit Partition Floorplan Power Place ECO Clock Route Timing Verify PVS Tools Windows Flows Help
online help
cadence
Layout
Schematic
wr_data[7:0]
clk
match_data[7:0]
wr_en
wr_mask[7:0]
search_key[7:0]
search_en
match_found
simple_tcam
Select single object and show module view from popup menu
[AnyDesk]
[CMOS_PR...
[cadence@...
[Virtuoso®...
[Virtuoso®...
tcam_New
[cadence@...
Innovus(T...
[Timing De...



Report File(s) top.mtarpt

Category Summary

Name:	all
Total Path:	201
Passing Path:	137
Failing Path:	64
WNS:	-0.0620
TNS:	-3.9680

Path Category

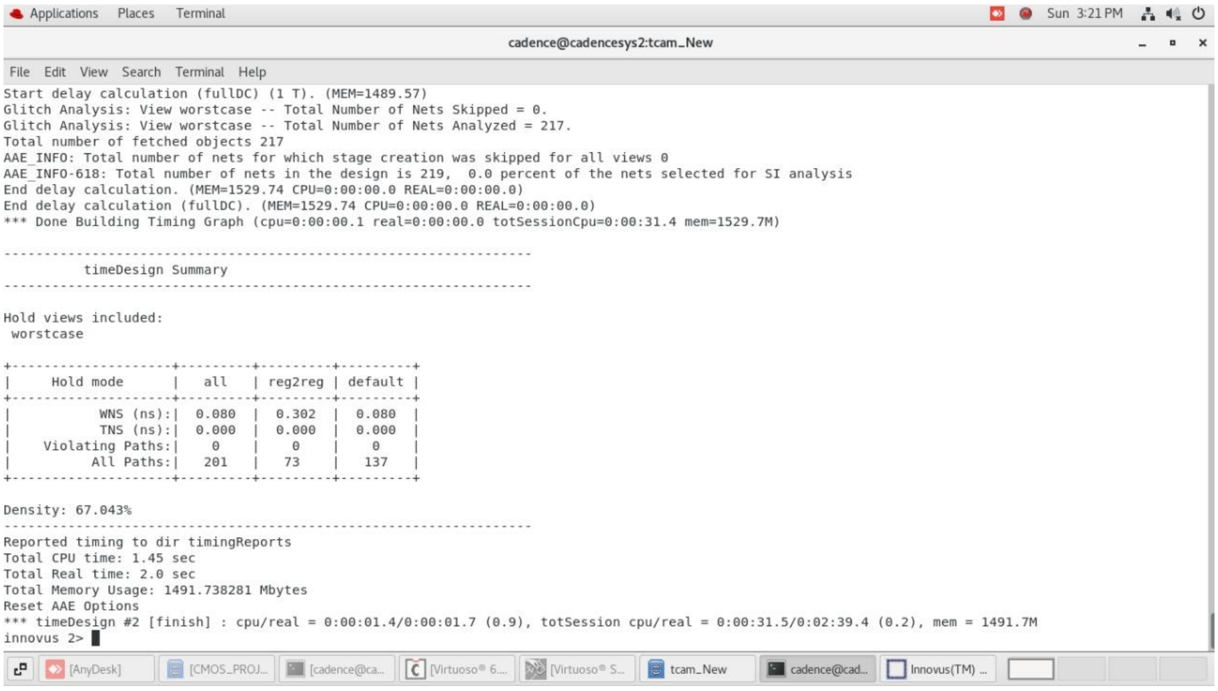
Cate	H	Category Name	Correction	WNS	TNS	#Failing Path
1		all		-0.062	-3.968	64

Path List

Category: all, Slack Range: all

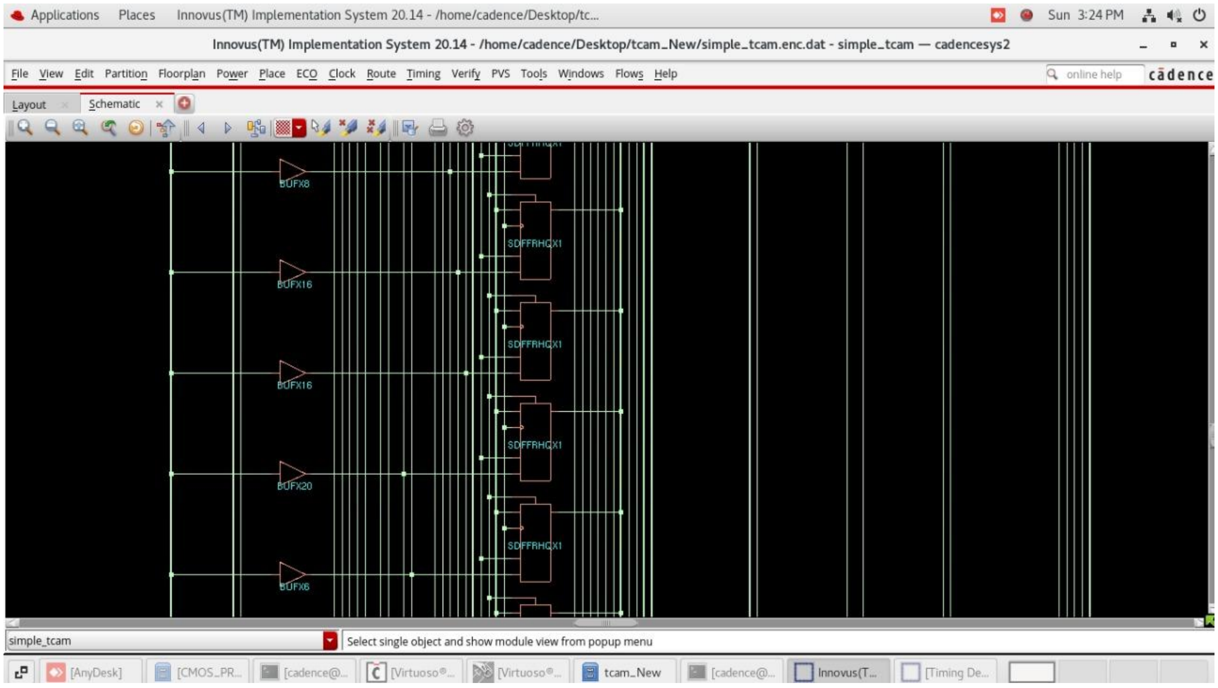
Page: 1 2 3 4 5

H	Path	Clock	ReqTime	Slack	Startpoint Pin	Endpoint Pin
	1	@(leading)->clk(leadi...	0.062	-0.062	wr_data[5]	data_mem_reg[2][5]/SI
	2	@(leading)->clk(leadi...	0.062	-0.062	wr_data[4]	data_mem_reg[1][4]/D
	3	@(leading)->clk(leadi...	0.062	-0.062	wr_data[7]	data_mem_reg[3][7]/D
	4	@(leading)->clk(leadi...	0.062	-0.062	wr_data[2]	data_mem_reg[1][2]/D
	5	@(leading)->clk(leadi...	0.062	-0.062	wr_data[6]	data_mem_reg[2][6]/SI
	6	@(leading)->clk(leadi...	0.062	-0.062	wr_data[0]	data_mem_reg[2][0]/SI


Applications Places Terminal
Sun 3:21 PM
cadence@cadencesys2:tcam_New
File Edit View Search Terminal Help
[AnyDesk] [CMOS_PROJ... [cadence@ca... [Virtuoso® 6.... [Virtuoso® S... tcam_New cadence@cad... Innovus(TM) ...

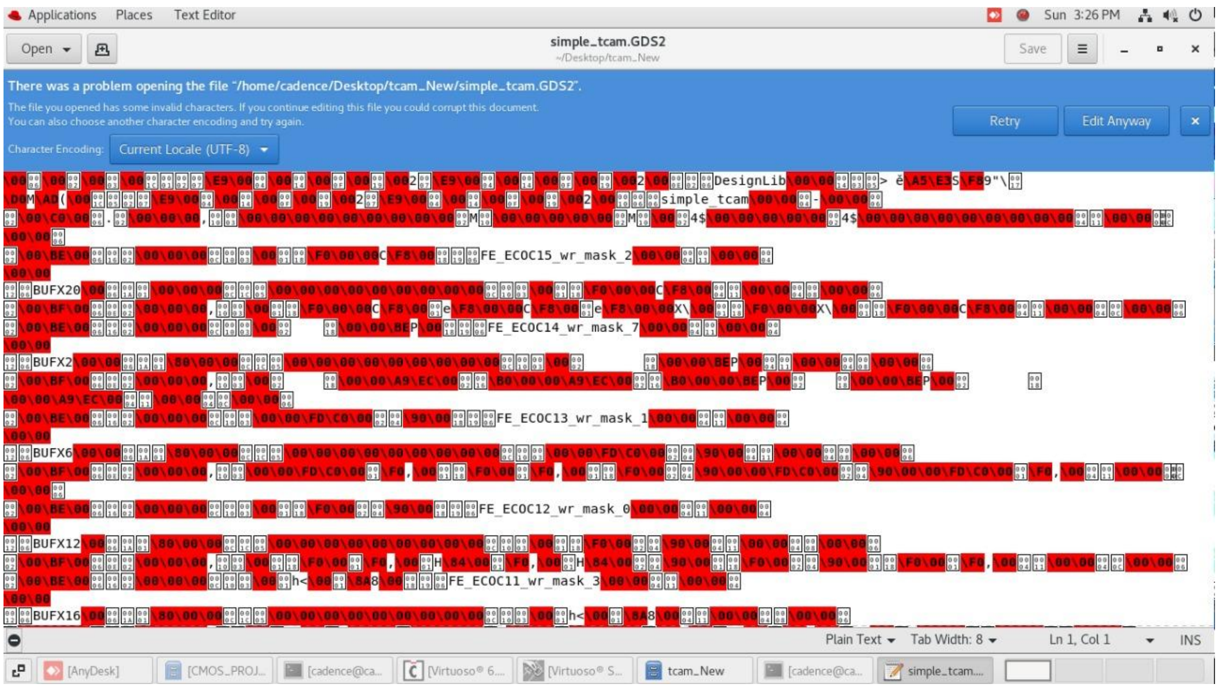
```
Start delay calculation (fullDC) (1 T). (MEM=1489.57)
Glitch Analysis: View worstcase -- Total Number of Nets Skipped = 0.
Glitch Analysis: View worstcase -- Total Number of Nets Analyzed = 217.
Total number of fetched objects 217
AAE_INFO: Total number of nets for which stage creation was skipped for all views 0
AAE_INFO-618: Total number of nets in the design is 219,  0.0 percent of the nets selected for SI analysis
End delay calculation. (MEM=1529.74 CPU=0:00:00.0 REAL=0:00:00.0)
End delay calculation (fullDC). (MEM=1529.74 CPU=0:00:00.0 REAL=0:00:00.0)
*** Done Building Timing Graph (cpu=0:00:00.1 real=0:00:00.0 totSessionCpu=0:00:31.4 mem=1529.7M)

------------------------------------------------------------------
          timeDesign Summary
------------------------------------------------------------------

Hold views included:
 worstcase

+--------------------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg | default |
+--------------------+---------+---------+---------+
|           WNS (ns):|  0.080  |  0.302  |  0.080  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |
|          All Paths:|   201   |   73    |   137   |
+--------------------+---------+---------+---------+

Density: 67.043%
------------------------------------------------------------------
Reported timing to dir timingReports
Total CPU time: 1.45 sec
Total Real time: 2.0 sec
Total Memory Usage: 1491.738281 Mbytes
Reset AAE Options
*** timeDesign #2 [finish] : cpu/real = 0:00:01.4/0:00:01.7 (0.9), totSession cpu/real = 0:00:31.5/0:02:39.4 (0.2), mem = 1491.7M
innovus 2> 
```

Schematic with buffers


Applications Places Innovus(TM) Implementation System 20.14 - /home/cadence/Desktop/tc...
Sun 3:24 PM
Innovus(TM) Implementation System 20.14 - /home/cadence/Desktop/tcam_New/simple_tcam.enc.dat - simple_tcam — cadencesys2
File View Edit Partition Floorplan Power Place ECO Clock Route Timing Verify PVS Tools Windows Flows Help
online help
cādence
Layout Schematic
BUFX8
BUFX16
BUFX16
BUFX20
BUFX6
SDFFRHQX1
SDFFRHQX1
SDFFRHQX1
SDFFRHQX1
simple_tcam
Select single object and show module view from popup menu
[AnyDesk] [CMOS_PR... [cadence@... [Virtuoso®... [Virtuoso®... tcam_New [cadence@... Innovus(T... [Timing De...

8. GDSII

GDSII (Graphic Data System II) is a file format used to represent the layout of integrated circuits (ICs) for fabrication. It contains geometric shapes and layers that define the chip's physical structure, such as metal traces, vias, and transistors. GDSII is the standard format for transferring the final design from the physical design stage to the foundry for manufacturing.



There was a problem opening the file "/home/cadence/Desktop/tcam_New/simple_tcam.GDS2".

The file you opened has some invalid characters. If you continue editing this file you could corrupt this document.
You can also choose another character encoding and try again.

```
\00\00\00\00\E9\00\00\00\00\002\E9\00\00\00\00\002\00DesignLib\00\00> ě\A5\E3S\F89"\
\D0M\AD(\00\E9\00\00\00\00\002\E9\00\00\00\00\002\00simple_tcam\00\00-\00\00
\00\C0\00.\00\00\00,\00\00\00\00\00\00\00\00\00M\00\00\00\00\00M\00 4$\00\00\00\00\00 4$\00\00\00\00\00\00\00\00\00\00\00
\00\00
\00\BE\00\00\00\00\00\00\F0\00\00C\F8\00FE_ECOC15_wr_mask_2\00\00\00\00
\00\00
BUFX20\00\00\00\00\00\00\00\00\00\00\00\00\00\00\00\F0\00\00C\F8\00\00\00\00\00
\00\BF\00\00\00\00,\00\F0\00\00C\F8\00e\F8\00\00C\F8\00e\F8\00\00X\\00\F0\00\00X\\00\F0\00\00C\F8\00\00\00\00\00
\00\BE\00\00\00\00\00\00\00\BEP\00FE_ECOC14_wr_mask_7\00\00\00\00
\00\00
BUFX2\00\00\80\00\00\00\00\00\00\00\00\00\00\00\00\00\00\BEP\00\00\00\00\00
\00\BF\00\00\00\00,\00\00\00\A9\EC\00\B0\00\00\A9\EC\00\B0\00\00\BEP\00\00\00\BEP\00
\00\00\A9\EC\00\00\00\00\00
\00\BE\00\00\00\00\00\00\FD\C0\00\90\00FE_ECOC13_wr_mask_1\00\00\00\00
\00\00
BUFX6\00\00\80\00\00\00\00\00\00\00\00\00\00\00\00\00\FD\C0\00\90\00\00\00\00\00
\00\BF\00\00\00\00,\00\00\FD\C0\00\F0,\00\F0\00\F0,\00\F0\00\90\00\00\FD\C0\00\90\00\00\FD\C0\00\F0,\00\00\00
\00\00
\00\BE\00\00\00\00\00\F0\00\90\00FE_ECOC12_wr_mask_0\00\00\00\00
\00\00
BUFX12\00\80\00\00\00\00\00\00\00\00\00\00\00\00\F0\00\90\00\00\00\00\00
\00\BF\00\00\00\00,\00\F0\00\F0,\00H\84\00\F0,\00H\84\00\90\00\F0\00\90\00\F0\00\F0,\00\00\00\00\00
\00\BE\00\00\00\00\00h<\00\B8\00FE_ECOC11_wr_mask_3\00\00\00\00
\00\00
BUFX16\00\80\00\00\00\00\00\00\00\00\00\00\00\00\00h<\00\B8\00\00\00\00\00
```

9. Conclusion:

This project involved a comprehensive exploration of the TCAM (Ternary Content Addressable Memory) design flow within an ASIC development context. Starting with the retrieval of Verilog code, we performed functional simulation to ensure the logic operated as expected under various conditions. The code coverage analysis highlighted untested areas, allowing for targeted improvements to ensure robust design verification.

The synthesis process was executed using Cadence Genus, optimizing the design for area, power, and timing, followed by linting to ensure clean, error-free code. The Design for Testability (DFT) and Automatic Test Pattern Generation (ATPG) steps enhanced the design's testability, ensuring that faults could be detected efficiently and minimizing test time and costs. Logical Equivalence Checking (LEC) was then performed to confirm that the synthesized netlist remained functionally equivalent to the original RTL design, ensuring no discrepancies had been introduced during the synthesis process.

Following these verification steps, the physical design phase was executed using Cadence Innovus. The design was floorplanned, placed, and routed, ensuring optimal performance and compliance with timing, power, and area constraints. Static Timing Analysis (STA) was conducted post-routing to verify that the design met all timing requirements across different paths. Finally, the GDSII file, representing the layout, was saved for manufacturing, completing the ASIC flow.

This systematic approach ensured that the TCAM design met all functional, timing, and testability requirements, while also optimizing the design for efficient power consumption and area usage. The project demonstrated the importance of each step in the ASIC design flow, from RTL design through to final physical implementation, and highlighted the tools and techniques necessary to ensure the design's correctness, reliability, and manufacturability.