**Dual Port RAM (RTL to GDSII) 3EC610IE24 – Physical Design for CMOS Technology**

Special Assignment Project Report

**MINOR IN**

**VLSI DESIGN**

By

**Heer Ramavat (22BCE296)**



**Department of Electronics and Communication Engineering, Institute of Technology,**

**Nirma University, Ahmedabad- 382481**

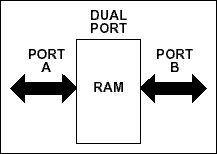
# Introduction

**What is Dual Port RAM?**

**Dual Port RAM** is a type of Random Access Memory that allows **simultaneous access to two different memory locations** through **two separate ports**. This makes it especially useful in applications requiring high-speed data processing and communication between two independent systems or processes.

## Key Features:

* **Two independent ports (Port A and Port B)** Each port can have its own address, data, and control lines.
* **Concurrent Read/Write** Both ports can perform read or write operations at the same time (with some restrictions if they access the same address).



## Applications:

* FIFO buffers
* Video/image processing
* Dual-processor systems
* Communication between clock domains (asynchronous dual port RAM)

# Simulation in NC launch (Cadence Tool)

## a. Design in Verilog Code (.v file)

`timescale 1ns / 1ps

module dual\_port\_ram1(

//it has two sets of address and data bus, input [7:0] data\_a, data\_b, //input data

input [5:0] addr\_a, addr\_b, //Port A and Port B address input we\_a, we\_b, //write enable for Port A and Port B input clock,reset, //clk

output reg [7:0] q\_a, q\_b //output data at Port A and Port B

);

reg [7:0] ram [63:0]; //8\*64 bit ram always @(posedge clock)

begin

if (reset) begin

q\_a <= 0;

q\_b <= 0;

end

end

always @ (posedge clock)

begin if(we\_a)

ram[addr\_a] <= data\_a; else

q\_a <= ram[addr\_a]; end

always @ (posedge clock) begin

if(we\_b)

ram[addr\_b] <= data\_b; else

q\_b <= ram[addr\_b]; end

endmodule

**Testbench in Verilog Code (.v file)**

`timescale 1ns / 1ps

module dual\_port\_ram1\_tb();

reg [7:0] data\_a, data\_b; //input data

reg [5:0] addr\_a, addr\_b; //Port A and Port B address reg we\_a, we\_b; //write enable for Port A and Port B reg clock, reset; //clk

wire [7:0] q\_a, q\_b; //output data at Port A and Port B

dual\_port\_ram1 dual\_port\_ram( .data\_a(data\_a), .data\_b(data\_b), .addr\_a(addr\_a), .addr\_b(addr\_b), .we\_a(we\_a), .we\_b(we\_b), .clock(clock), .reset(reset), .q\_a(q\_a), .q\_b(q\_b) );

initial

begin clock=1'b0;

forever #5 clock = ~clock; end

initial begin

reset = 1'b1; #10;

reset = 1'b0;

we\_a = 1'b1; we\_b = 1'b1;

data\_a = 8'h33; addr\_a = 6'h01;

data\_b = 8'h44; addr\_b = 6'h02;

#10;

we\_a= 1'b0;

addr\_a = 6'h01; //read a

#10

we\_b = 1'b0;

addr\_b = 6'h02;//read b #10;

we\_a = 1'b1; we\_b = 1'b1;

data\_a = 8'h55; addr\_a = 6'h04;

data\_b = 8'h66; addr\_b = 6'h05;

#10

we\_a = 1'b0;

addr\_a = 6'h04; //read a

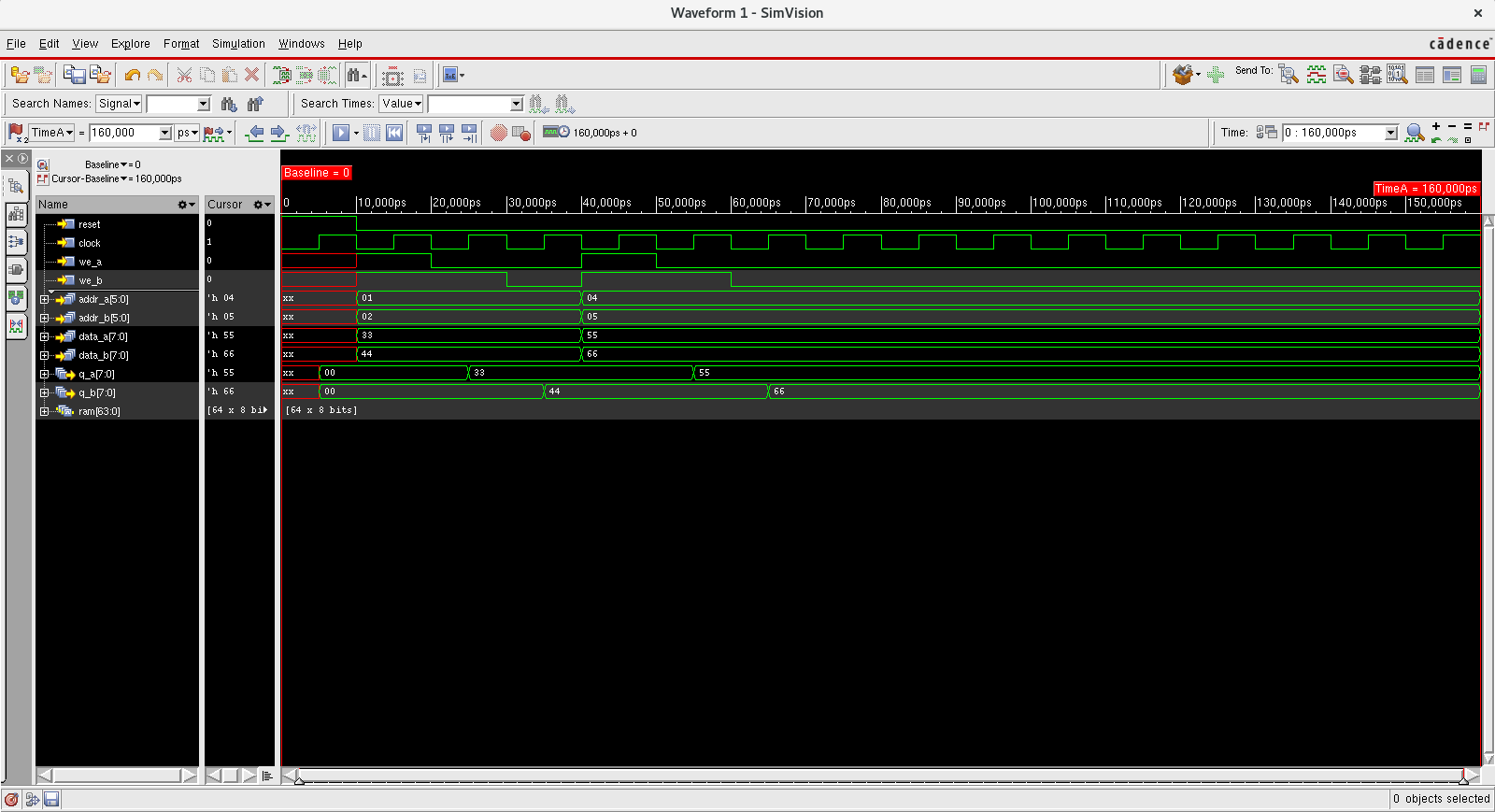
#10

we\_b = 1'b0; addr\_b = 6'h05;

#100;

$finish; end

**Waveform**





# Synthesis in Genus (Cadence Tool)

## TCL File Development (run.tcl file)

set\_db lib\_search\_path /home/install/FOUNDRY/digital/180nm/dig/lib

set\_db hdl\_search\_path ./

set\_db library slow.lib

read\_hdl dual\_port\_ram.v

elaborate

read\_sdc constraints\_top.sdc

synthesize -to\_mapped -effort medium

write\_sdf -timescale ns -nonegchecks -recrem split -edges check\_edge > delays.sdf

write\_hdl > dual\_port\_ram\_netlist.v

write\_sdc > dual\_port\_ram\_sdc.sdc

gui\_show

report timing > dual\_port\_ram\_timing.rep

report power > dual\_port\_ram\_power.rep

report area > dual\_port\_ram\_cell.rep

report messages > dual\_port\_ram\_message.rep

## SDC File Development (constraints\_top.sdc)

create\_clock -name clock -period 2 -waveform {0 1} [get\_ports "clock"] set\_clock\_transition -rise 0.1 [get\_clocks "clock"]

set\_clock\_transition -fall 0.1 [get\_clocks "clock"] set\_clock\_uncertainty 0.01 [get\_ports "clock"]

set\_input\_delay -max 1.0 [get\_ports "reset"] -clock [get\_clocks "clock"] set\_input\_delay -max 1.0 [get\_ports "clock"] -clock [get\_clocks "clock"] set\_input\_delay -max 1.0 [get\_ports "en"] -clock [get\_clocks "clock"] set\_input\_delay -max 1.0 [get\_ports "d"] -clock [get\_clocks "clock"]

set\_input\_delay -max 1.0 [get\_ports "dir"] -clock [get\_clocks "clock"] set\_output\_delay -max 1.0 [get\_ports "out"] "-clock" [get\_clocks "clock"]

## Dual port RAM Schematic

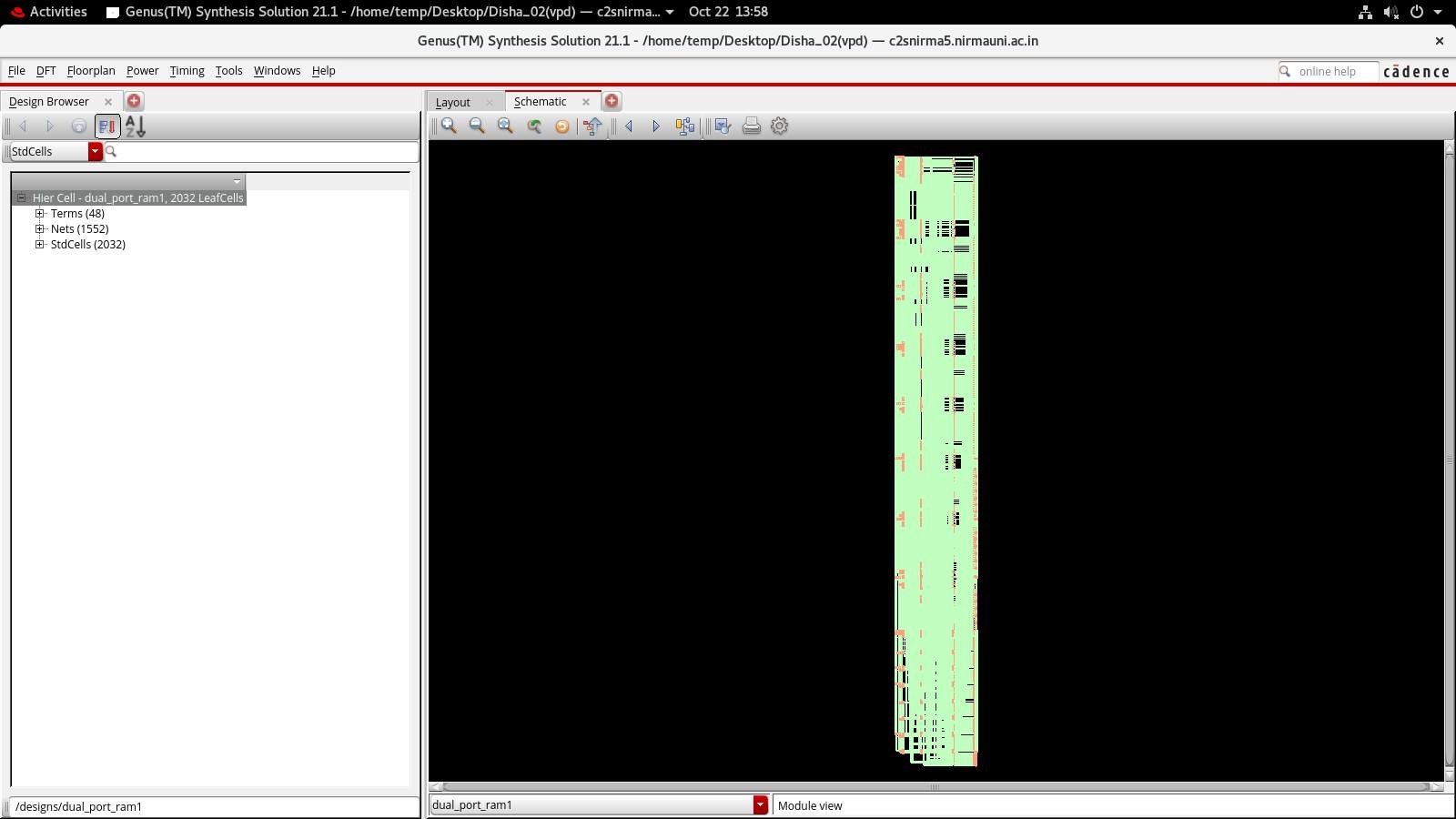


Fig: 2 Schematic in Genus tool

# ASIC flow in Innovus (Cadence Tool)

## Floor Planning

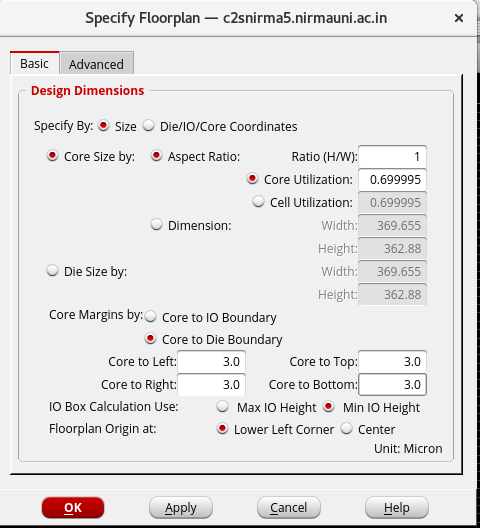


Fig. 3: Floorplanning Parameters



### Fig. 4: Floorplanning

## Power Planning

### Step 1: Add Rings

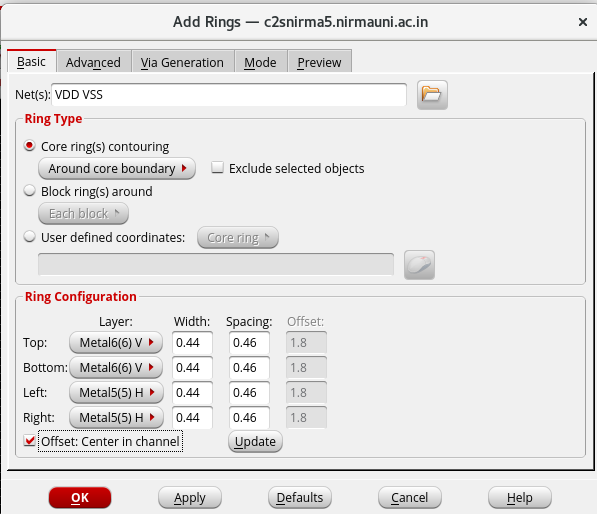


Fig.5: Power Planning (Step-1)

### Step 2: Add Strips

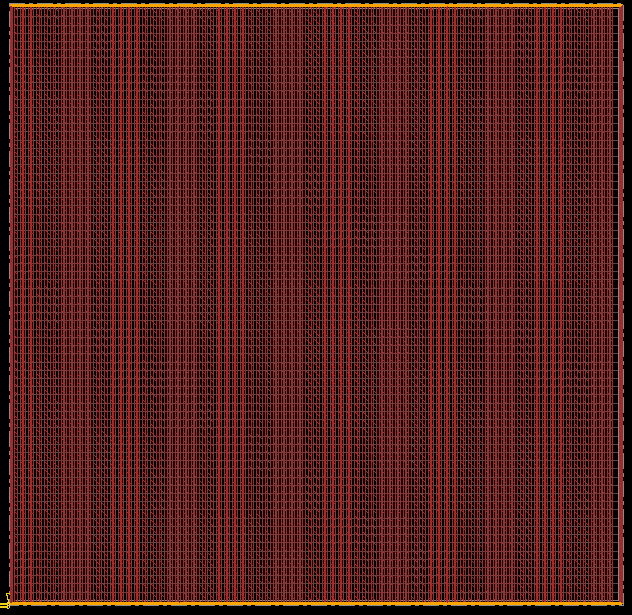


Fig. 6: Power Planning (Step-2)

### Power planning report

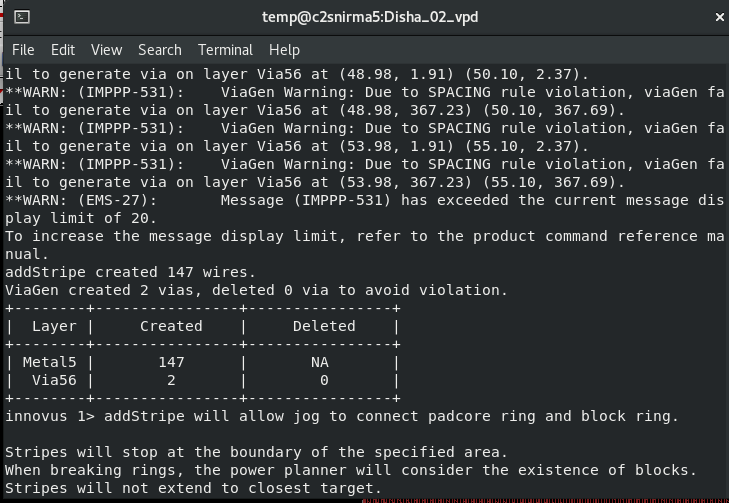


Fig. 7: Power Planning report

## Routing

### Step 3: Routing

### Fig. 8: Step- 3 Routing

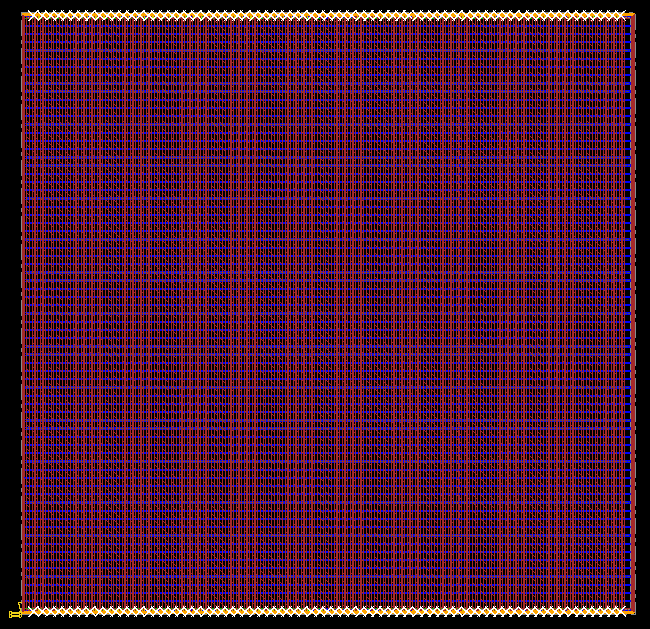


Fig. 9: Step- 3 Layout after Routing

### Routing report

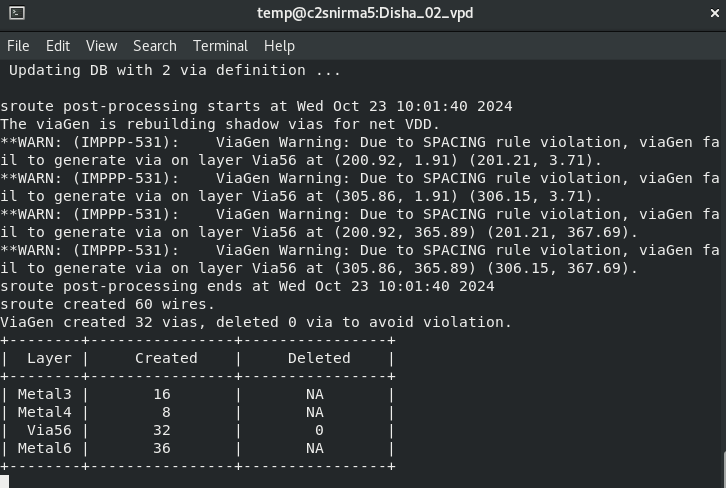


Fig 10: Routing Report

## Placement of Standard Cells

### Step 4: End Cap Placed

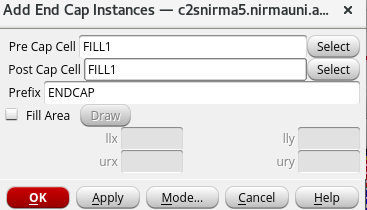
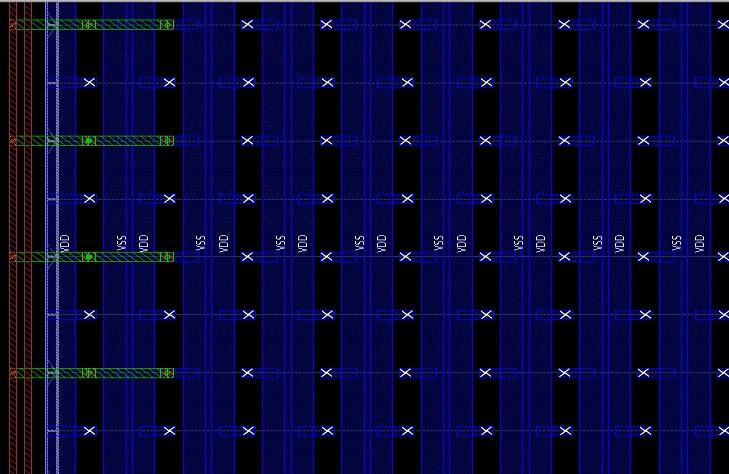
 

Fig. 11: Step- 4 End Cap Placed

### Step 5: Well Cells Placed

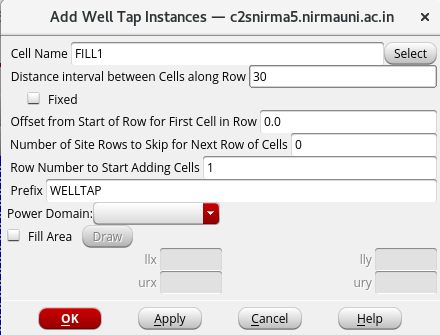


Fig. 12: Step- 5 Well Cells Placed

### Step 6: Placement of Standard Cell and Pin done

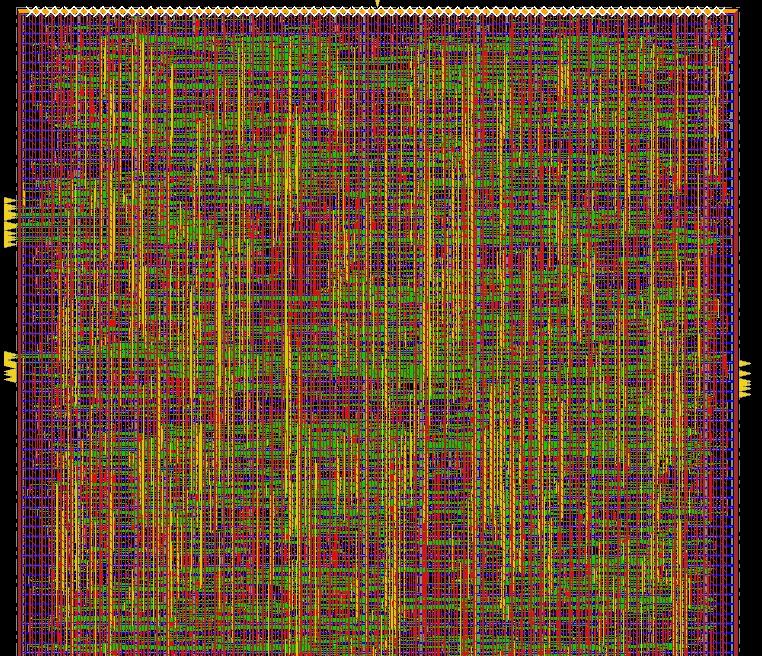


Fig. 13: Step- 6 Placement of Standard Cell and Pin done

## Pre-CTS analysis (Setup time)

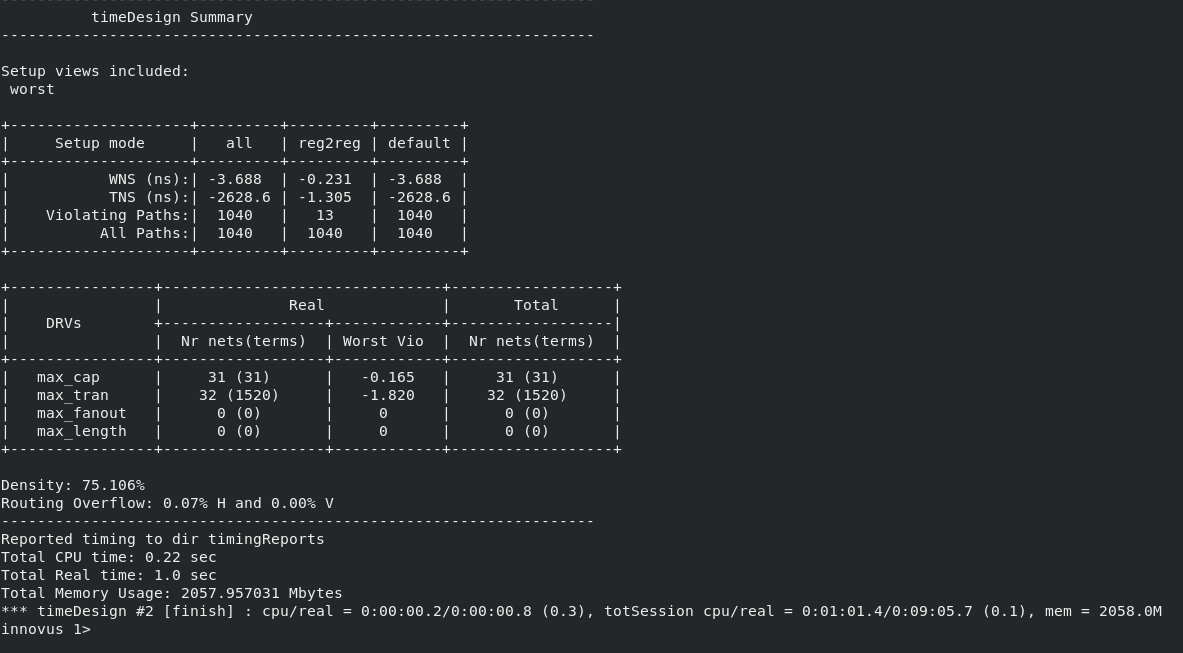


Fig. 14: Violation in Pre-CTS simulation

## Optimization of Pre-CTS Setup time Violation

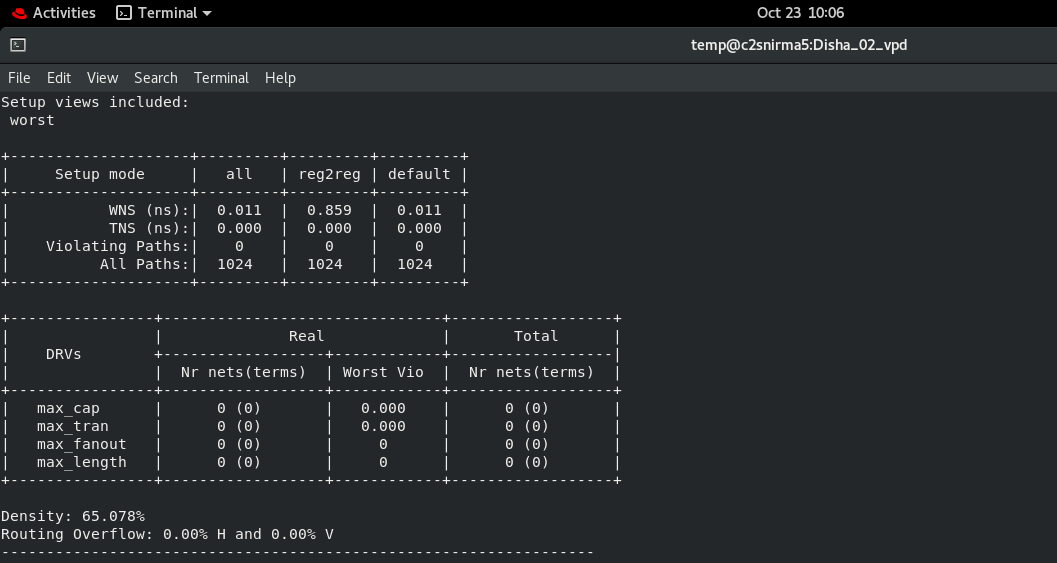


Fig. 15: Violation Removed in Pre- CTS simulation

## Clock Tree Synthesis (CTS)

### Clock tree insertion commands:

add\_ndr -width {Metal1 0.12 Metal2 0.14 Metal3 0.14 Metal4 0.14

Metal5 0.14 Metal6 0.14} -spacing {Metal1 0.12 Metal2 0.14 Metal30.14Metal4 0.14

Metal5 0.14 Metal6 0.14} -name 2w2s

create\_route\_type -name clkroute -non\_default\_rule 2w2s - bottom\_preferred\_layer Metal5 -top\_preferred\_layer Metal6

set\_ccopt\_property route\_type clkroute -net\_type trunk set\_ccopt\_property route\_type clkroute -net\_type leaf set\_ccopt\_property buffer\_cells {CLKBUFX8 CLKBUFX12} set\_ccopt\_property inverter\_cells {CLKINVX8 CLKINVX12}

set\_ccopt\_property clock\_gating\_cells TLATNTSCA\* create\_ccopt\_clock\_tree\_spec -file ccopt.spec

source ccopt.spec

## Clock Tree (with ccopt.spec report)

### Step 7: Clock Tree

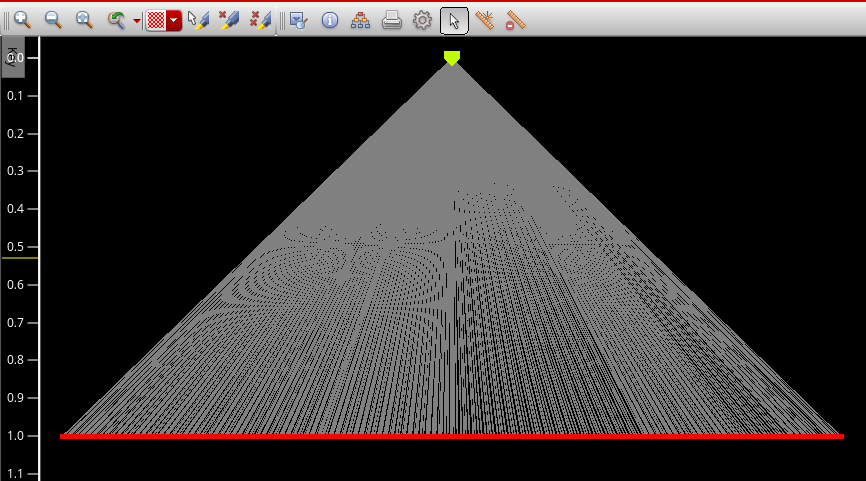


Fig. 16: Step- 7 Clock Tree

### ccopt.spec report:



Fig. 17: ccopt.spec report

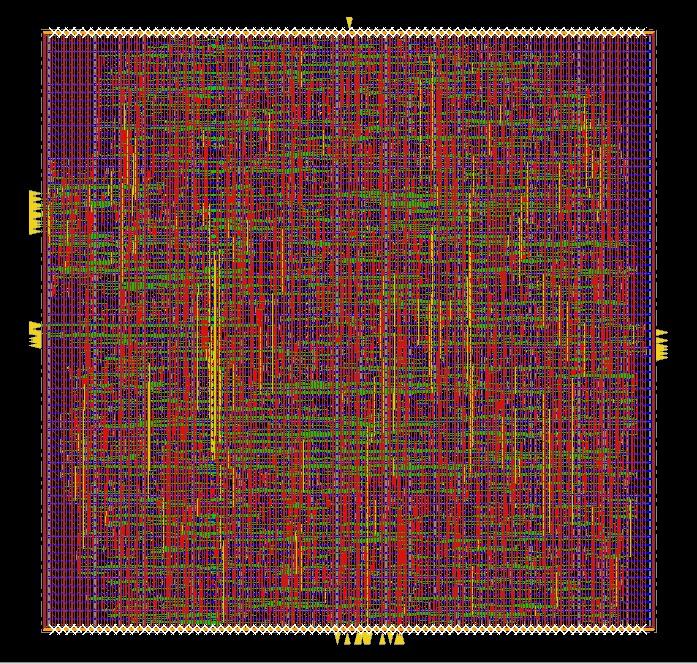


Fig. 18: Clock tree added in design

## Post-CTS analysis Setup and Hold time

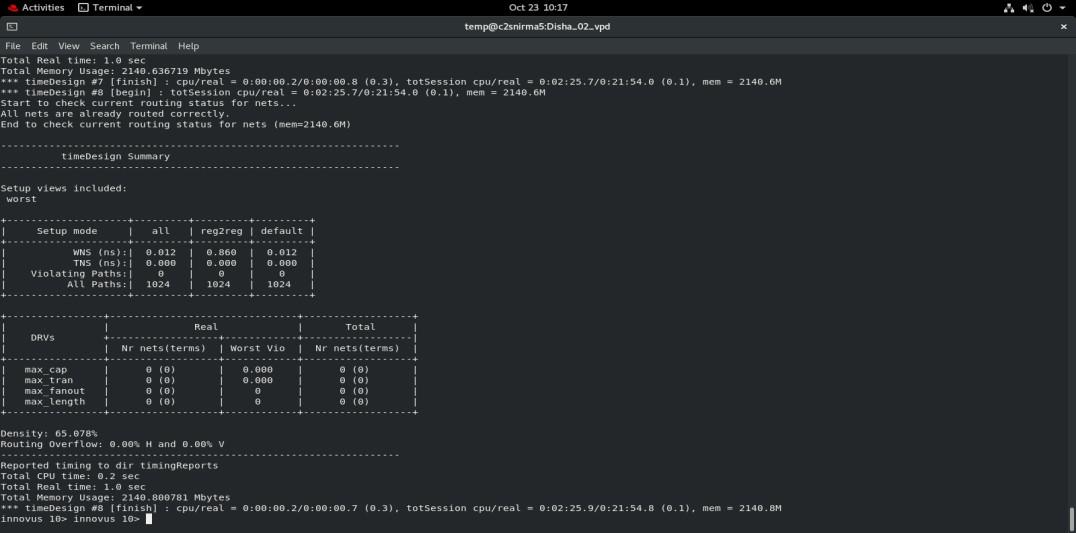
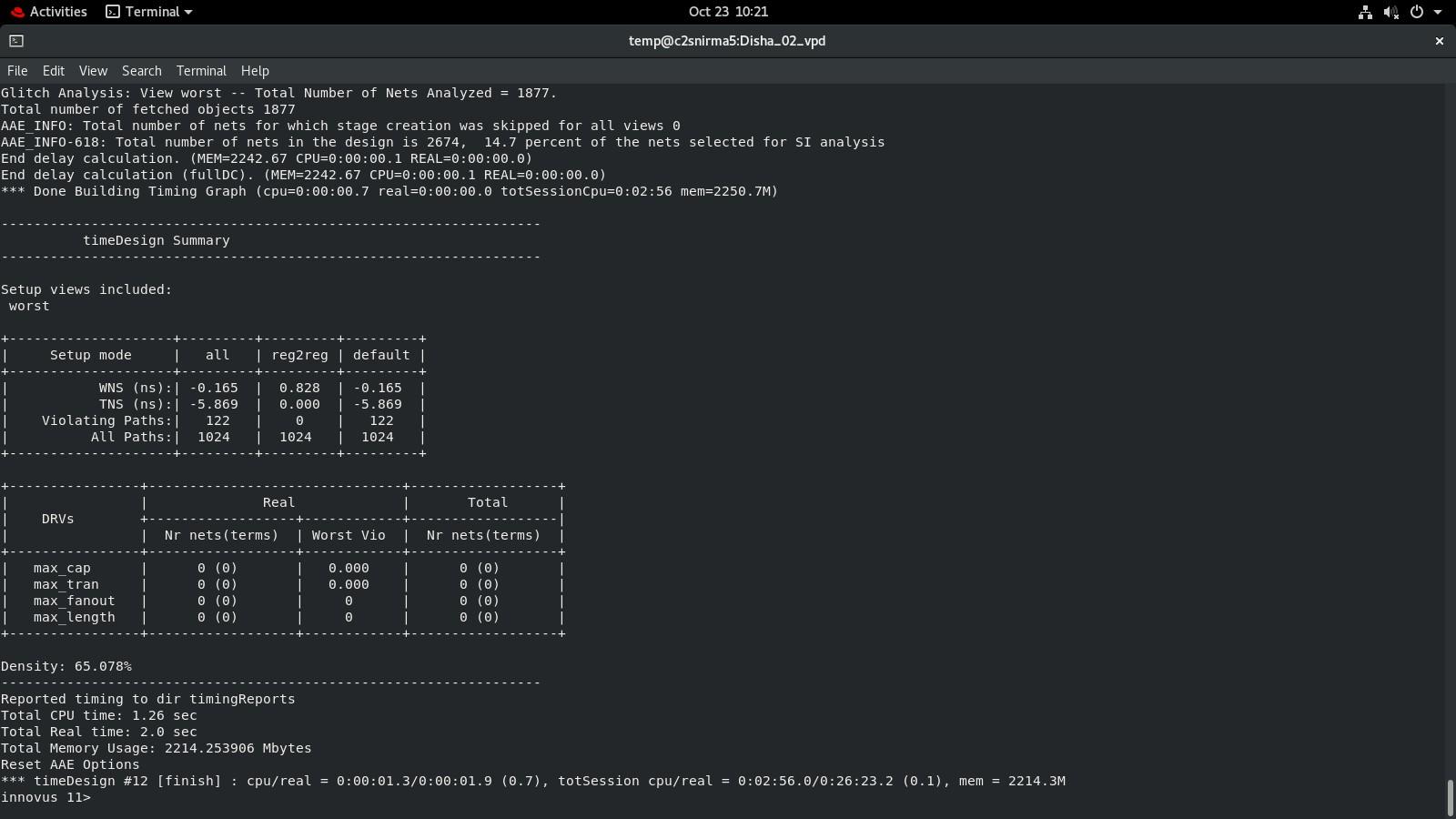
Fig. 19: Post-CTS Setup report



Fig. 20: Post-CTS Hold report

## Post Route Timing Analysis

Fig. 21: Post-Rout setup report

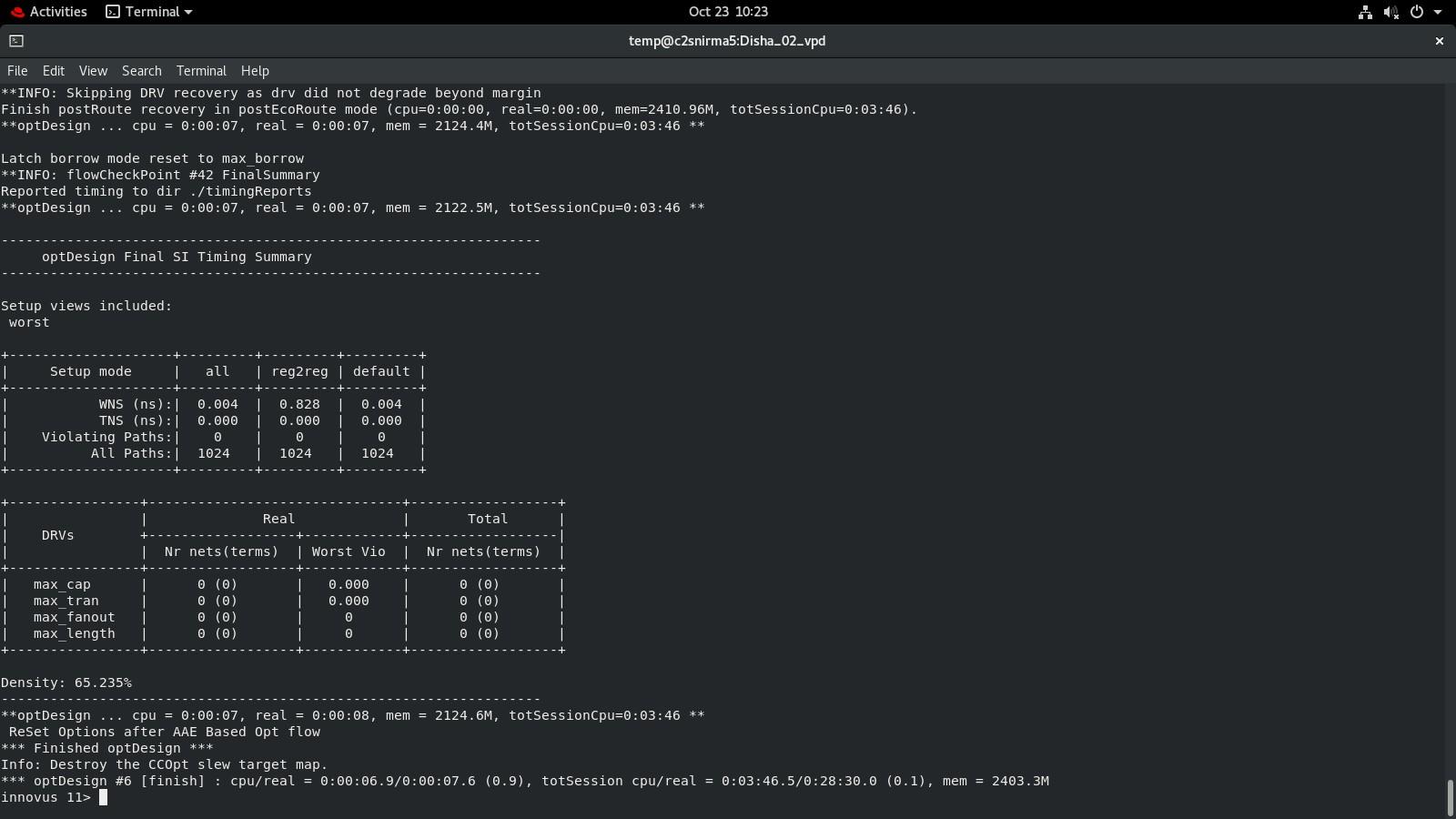


Fig. 22: Post-Rout optimized setup report

## Nano Routing

### Step 8: Nano Route (Final Layout)



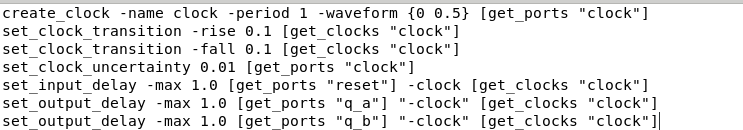
Fig. 23: Step- 8 Nano Route done (Final Layout)

## Timing Analysis

### ◎ Time period – 1 ns

### ◎ Operating frequency – 1 GHZ

### ◎ Total standard cells -4056



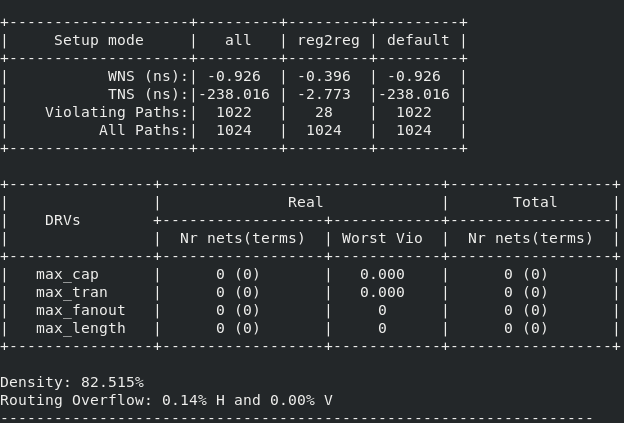


Fig. 24: Timing report when frequency is 1GHZ

### ◎ Time period – 1.3 ns

### ◎ Operating frequency – 0.75 GHZ

### ◎ Total standard cells -4049

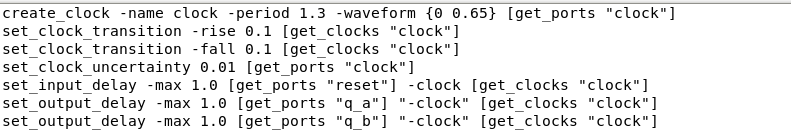


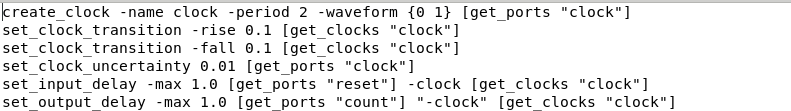


Fig. 25: Timing report when frequency is 0.75GHZ

### ◎ Time period – 2 ns

### ◎ Operating frequency – 0.5 GHZ

### ◎ Total standard cells -2032



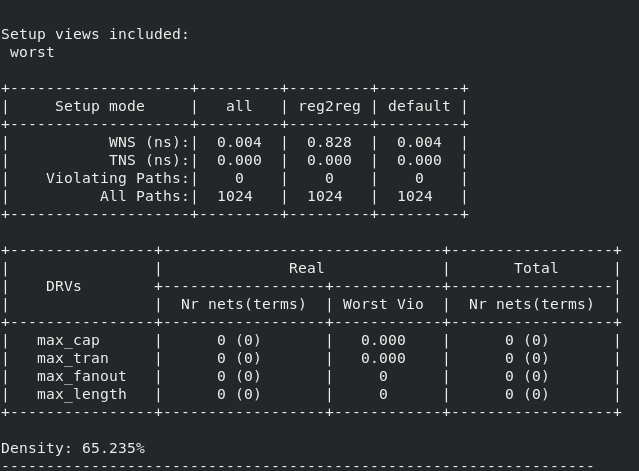


Fig. 26: Timing report when frequency is 0.5 GHZ

**Why there is a huge difference in number of standard cells?**

When you increase the targeted clock frequency for an ASIC design, several factors can lead to an increase in the number of standard cells used in the design:

Increased Timing Constraints: Higher clock frequencies require tighter timing constraints, which can lead to more complex logic arrangements to meet setup and hold times. This often necessitates additional standard cells for buffering and optimization.

Reduced Signal Propagation Delay: To achieve a higher frequency, the propagation delay through the logic gates must be minimized. This can involve using more cells with shorter delays, increasing the total number of cells.

Pipelining: To meet higher frequency requirements, designs often incorporate pipelining. This technique splits the logic into multiple stages, which increases the number of flip-flops and combinatorial logic cells.

Increased Use of Buffers and Repeaters: As clock frequency rises, more buffering may be needed to drive larger loads and maintain signal integrity, particularly for long interconnects.

Power and Area Trade-offs: Higher frequencies can also lead to trade-offs in power consumption and area. Designers might choose to utilize more cells to balance these aspects while maintaining performance.

Complexity of Logic: Some designs may require more complex logic functions to optimize for speed, leading to an increase in the number of standard cells utilized.

Optimization Techniques: Tools may optimize the design for area and speed, often resulting in more cells being instantiated to achieve the required performance.

In summary, achieving higher clock frequencies often requires more resources in terms of standard cells to ensure that the design meets the necessary performance criteria while maintaining reliability and signal integrity.

1. **Conclusion**

### In summary, The RTL to GDSII flow is a systematic and crucial process in VLSI design, converting a high-level digital circuit description into a manufacturable layout. Starting with RTL design and simulation, followed by synthesis to produce a gate-level netlist, each stage builds toward an optimized physical layout. The flow then moves through floor planning, placement, clock tree synthesis, and routing, ensuring that timing, power, and area requirements are met. Rigorous checks like DRC, LVS, and power analysis verify that the design meets both functional and fabrication standards. Cadence tools provide an integrated platform for managing this complex sequence, enabling thorough optimization and verification. The final GDSII file, a fully verified layout, is prepared for tape-out, making this flow essential for producing reliable, high-performance chips ready for fabrication.