1. INTRODUCTION

Clock generation and distribution are foundational to synchronous digital system design, enabling coordinated data transfer, computation, and control across various processing and memory blocks. In advanced processors such as those based on the RISC-V architecture, subsystems often operate at different performance and power requirements, necessitating flexible and efficient clocking solutions. One widely adopted solution is the use of clock chain dividers, which generate slower clocks derived from a high-frequency input clock. These dividers are particularly important in clock domain crossing, peripheral timing units, and power-gated functional blocks, where reduced switching activity and localized clock control significantly improve energy efficiency.

Traditionally, clock chain dividers are implemented using D flip-flop (DFF) chains or counter-based designs due to their straightforward behavior and ease of timing analysis. However, such implementations suffer from continuous toggling even when downstream logic is idle, resulting in increased dynamic power consumption. With power becoming a primary design constraint in modern SoC and ASIC development, designers are turning toward low-power techniques such as clock gating and power gating. Clock gating conditionally disables the clock signal to idle portions of the circuit, while power gating disconnects power rails to prevent leakage current in unused logic blocks. These techniques are essential in reducing overall power consumption without compromising system functionality.

This project aims to investigate and compare three distinct RTL-level architectures for clock chain dividers, with a focus on low-power operation. The first is a baseline DFF-based clock divider, representing conventional design practice. The second design uses a gated Linear Feedback Shift Register (LFSR), which introduces feedback-based pseudo-random behavior while incorporating clock and power gating logic. The third and most novel approach uses symbolic 8-transistor (8T) SRAM bitcells to construct a sequential chain that mimics clock division behavior. All three architectures were designed in Verilog HDL, with integrated support for clock gating, power gating, and a test mode to override gating during verification.

2. OBJECTIVES

1. To design and implement three clock chain divider architectures:

    (i) D flip-flop (DFF)-based divider,

(ii) Gated LFSR-based divider, and

(iii) 8T SRAM-based divider.

2. To integrate clock gating, power gating, and test mode features into each design to enable low-power operation and observability.
3. To perform functional verification of all architectures through RTL simulation using ModelSim.
4. To synthesize the RTL designs using Synopsys Design Compiler and obtain reports for area, power, timing, and gate-level netlists.
5. To analyze and compare the three architectures based on synthesis results to determine the most efficient design.
6. To carry out physical design implementation of the selected SRAM-based divider using Cadence Innovus.

3. SOFTWARES USED

Model Sim 18.1[5]	Pre-synthesis Simulation and Functional Verification
Synopsys Design Compiler [2]	Synthesis, gate-level netlist generation, and area, power and timing analysis
Cadence Innovus	Physical design flow including floor planning, placement, routing, and timing analysis
CentOS Linux 7[10]	Operating system environment for Synopsys toolchain compatibility
Synopsys Design Vision	RTL schematic viewing and gate-level netlist visualization

4.Functional Description of Clock Chain Divider Architectures

This section provides a functional overview of the three clock chain divider architectures implemented in this project: D flip-flop (DFF)-based, gated Linear Feedback Shift Register (LFSR)-based, and 8T SRAM-based designs. Each design includes support for clock gating, power gating, and test mode, enabling low-power operation and enhanced observability.

4.1 D Flip-Flop (DFF)-Based Clock Chain Divider

Inputs: clk (clock), rst (reset)

Functionality: This design uses a chain of D flip-flops to propagate a toggling bit through the register stages. On every rising edge of the clock, the contents of the register are shifted left, and the most

significant bit toggles. The register resets to all zeros when the reset signal is asserted.

Output: An 8-bit signal representing the current state of the divider register, with the toggling bit shifting sequentially.

4.2 Gated LFSR-Based Clock Chain Divider

Inputs: clk (clock), rst (reset), en_clk (clock gating enable), en_pwr (power gating enable), test_mode (test override)

Functionality: This architecture employs an 8-bit LFSR that generates a pseudo-random sequence using XOR feedback logic. Clock gating disables the internal clock when not required, and power gating prevents register updates when power is gated off. The test mode overrides gating logic to ensure visibility and operability during verification.

Output: An 8-bit signal representing the current LFSR state, updated only when clock and power gating conditions permit.

4.3 8T SRAM-Based Clock Chain Divider

Inputs: clk (clock), rst (reset), en_clk (clock gating enable), en_pwr (power gating enable), test_mode (test override)

Functionality: This design mimics sequential behavior using a chain of symbolic 8T SRAM bitcells. A toggling bit is written into one SRAM cell at a time, while the rest hold their previous values. Write and read wordlines are selectively activated to control the flow of data through the chain. Clock and power gating logic prevent unnecessary switching activity, while the test mode allows full override for debugging and verification.

Output: An 8-bit signal reflecting the contents of all SRAM bitcells, forming the clock-divided output.

5.IMPLEMENTATION

5.1 Functional Verification (Pre-Synthesis Simulation using ModelSim)

Before moving on to synthesis and implementation, it is important to ensure that each design works correctly at the logical level. This process is known as functional verification or pre-synthesis simulation. It helps verify that the code we wrote in Verilog behaves exactly as intended — especially under different control signals like reset, clock gating, power gating, and test mode.

In this project, three different clock chain divider designs were verified using ModelSim:

- D Flip-Flop (DFF)-based divider
- Gated Linear Feedback Shift Register (LFSR)-based divider
- 8T SRAM-based divider

Each design was paired with a Verilog testbench that provided input signals and observed output responses. These testbenches helped simulate real-world scenarios and ensured that the designs reacted properly to various conditions.

5.1.1Steps Followed in ModelSim for Functional Verification:

Step 1: Firstly, Open Modelsim and Create new project and enter name as "ECE240_FinalProject"

File-> New-> Project-> ECE240_FinalProject-> OK

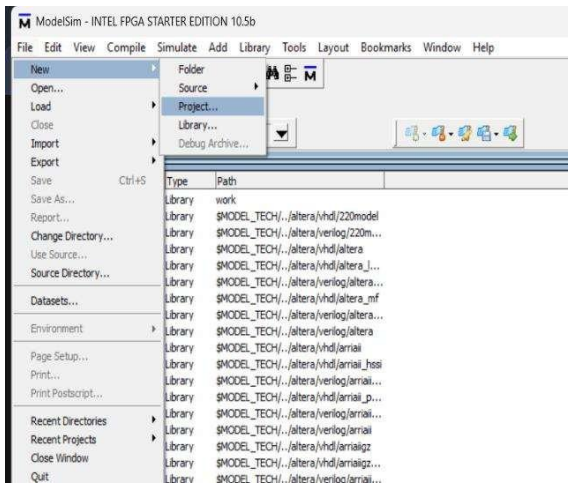


Figure 1:Creating New ModelSim Project

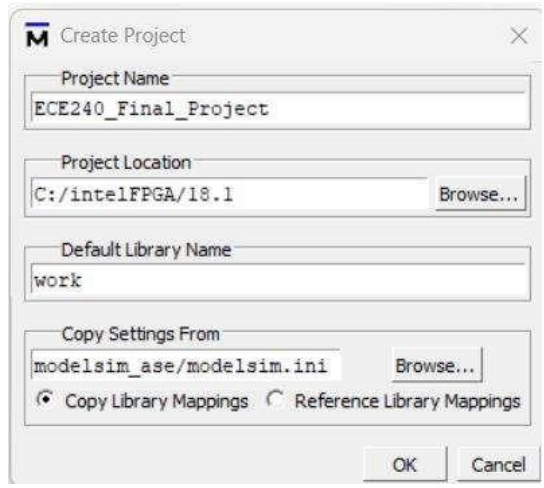


Figure 2: Adding Existing Verilog File

Step 2: Add all Verilog files like main module, sub modules (if any) and the appropriate testbench for the SRAM Based Clock chain divider by following this flow of steps

Add Existing File(s)-> Browse-> Select 2 verilog modules and Testbench-> Open

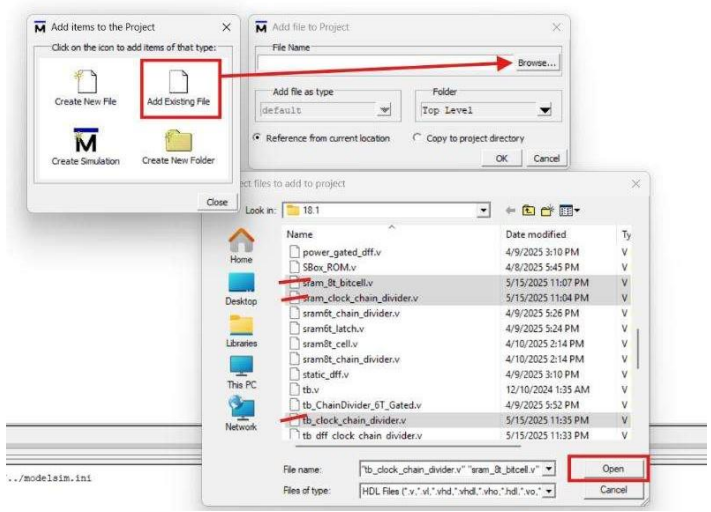


Figure 3: Compiling All Files

Step 3: write and modify the Verilog codes and testbench till no errors exist functionally. Later, select all the files from the left side bar and perform the following steps
Select all Verilog files-> Compile-> Compile all-> OK
Now, this process will display if there were any compilation errors for the codes implemented

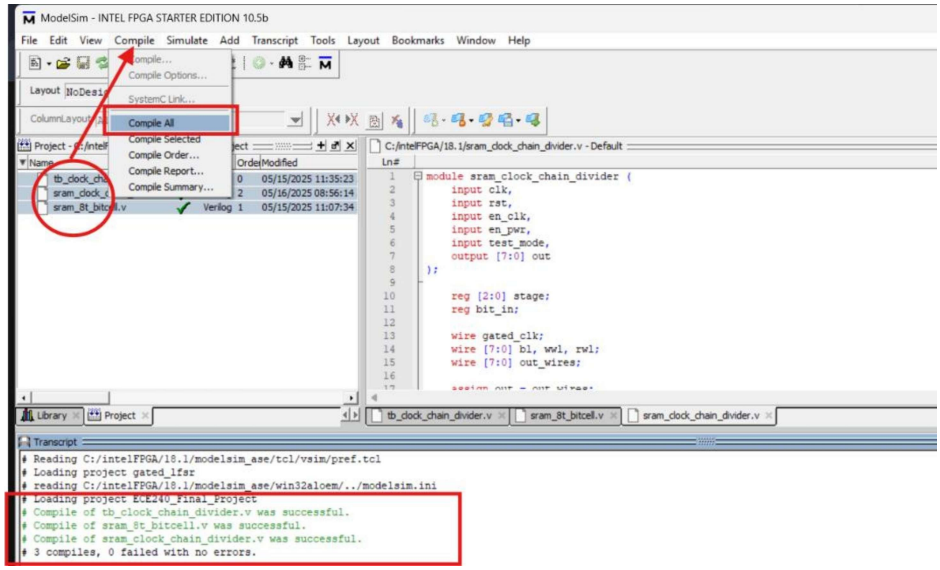


Figure 4: Start compiling in ModelSim

Step 4: Nextly, testbench simulation can be done by performing following flow of steps
Simulate-> Start Simulation-> tb_clock_chain_divider.v-> OK

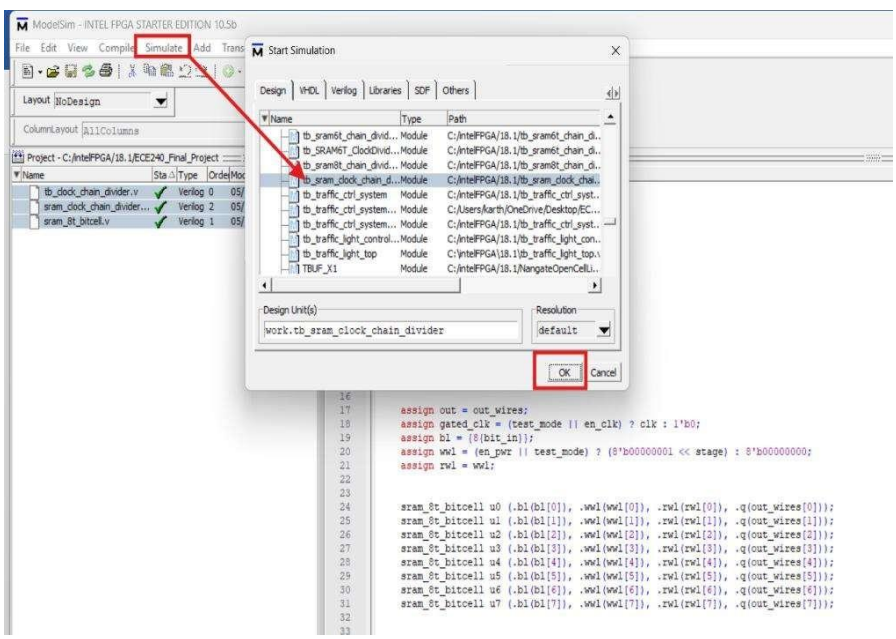


Figure 5: Start Simulating in ModelSim

Step 5: Now, we have to obtain waveforms which will provide the functional simulation of implemented

sram_clock_chain_divider by following these steps
Select all Instances with CTRL+ A-> CTRL+ W-> Run all-> No

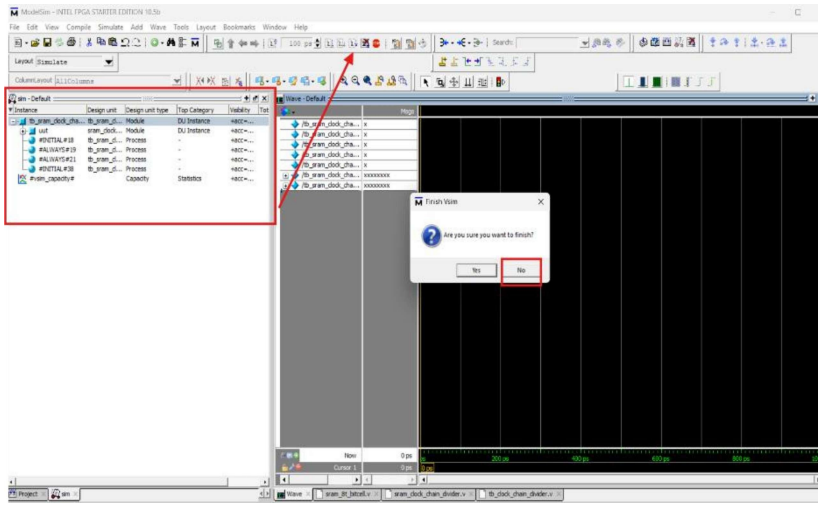


Figure 6: Adding waveforms to analyze

Step 6: Later, save the waveforms and transcripts for sram_clock_chain_divider. Now, repeat these steps for dff and gated lfsr based clock chain divider and save the appropriate waveforms and transcript as shown below

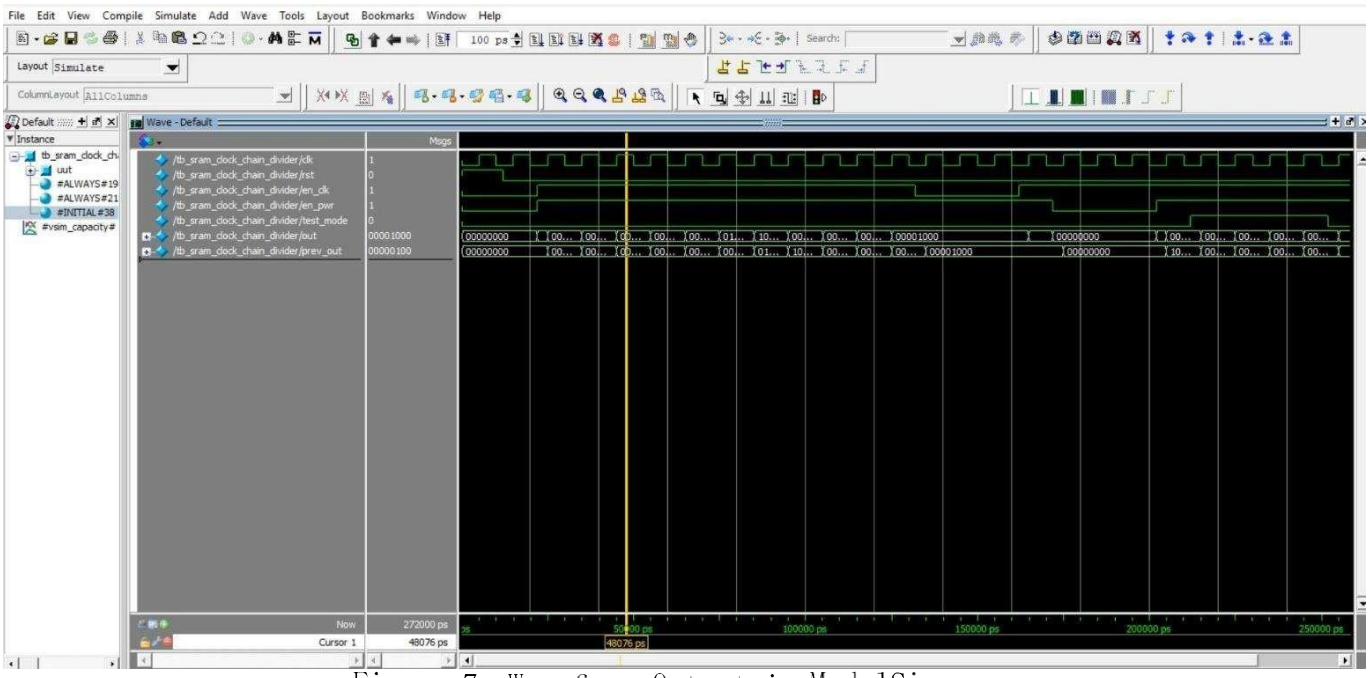
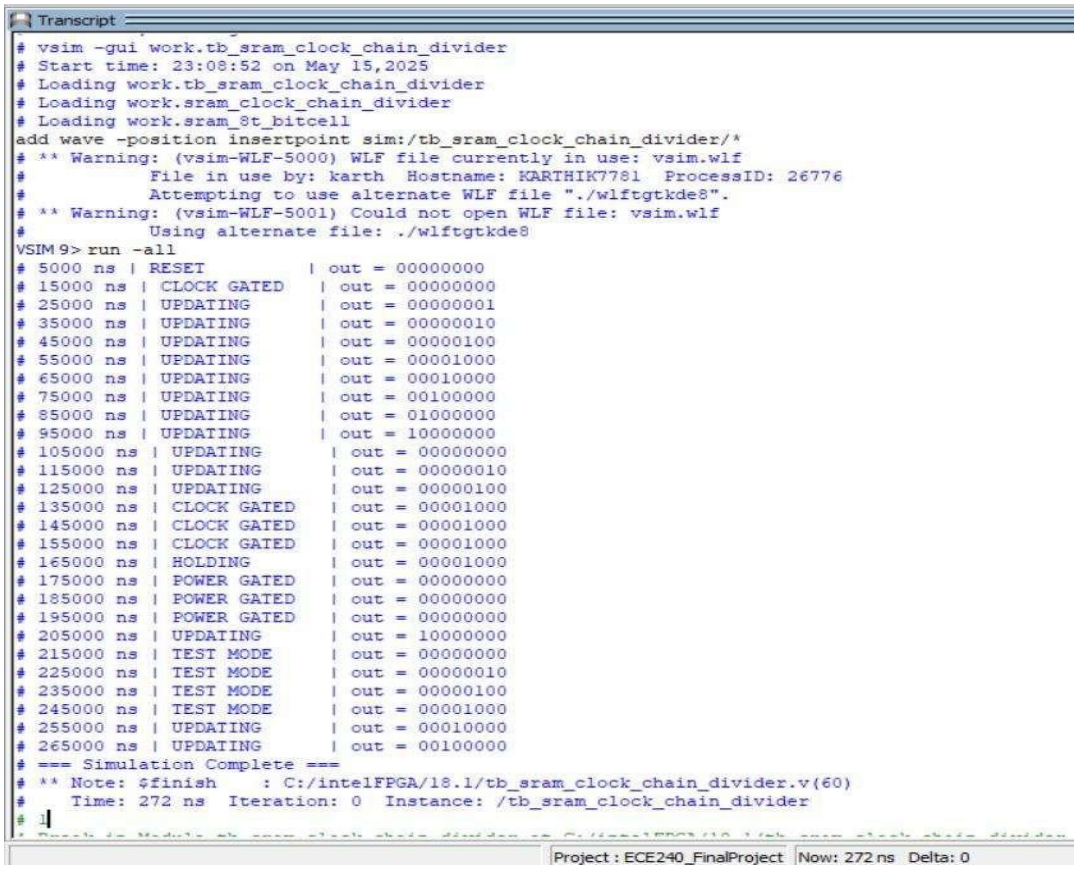


Figure 7: Waveform Output in ModelSim



```
Transcript
# vsim -gui work.tb_sram_clock_chain_divider
# Start time: 23:08:52 on May 15,2025
# Loading work.tb_sram_clock_chain_divider
# Loading work.sram_clock_chain_divider
# Loading work.sram_8t_bitcell
add wave -position insertpoint sim:/tb_sram_clock_chain_divider/*
# ** Warning: (vsim-WLF-5000) WLF file currently in use: vsim.wlf
#           File in use by: karth  Hostname: KARTHIK7781  ProcessID: 26776
#           Attempting to use alternate WLF file "./wlftgtkde8".
# ** Warning: (vsim-WLF-5001) Could not open WLF file: vsim.wlf
#           Using alternate file: ./wlftgtkde8
VSIM 9> run -all
# 5000 ns | RESET         | out = 00000000
# 15000 ns | CLOCK GATED   | out = 00000000
# 25000 ns | UPDATING      | out = 00000001
# 35000 ns | UPDATING      | out = 00000010
# 45000 ns | UPDATING      | out = 00000100
# 55000 ns | UPDATING      | out = 00001000
# 65000 ns | UPDATING      | out = 00010000
# 75000 ns | UPDATING      | out = 00100000
# 85000 ns | UPDATING      | out = 01000000
# 95000 ns | UPDATING      | out = 10000000
# 105000 ns | UPDATING      | out = 00000000
# 115000 ns | UPDATING      | out = 00000010
# 125000 ns | UPDATING      | out = 00000100
# 135000 ns | CLOCK GATED   | out = 00001000
# 145000 ns | CLOCK GATED   | out = 00001000
# 155000 ns | CLOCK GATED   | out = 00001000
# 165000 ns | HOLDING       | out = 00001000
# 175000 ns | POWER GATED   | out = 00000000
# 185000 ns | POWER GATED   | out = 00000000
# 195000 ns | POWER GATED   | out = 00000000
# 205000 ns | UPDATING      | out = 10000000
# 215000 ns | TEST MODE     | out = 00000000
# 225000 ns | TEST MODE     | out = 00000010
# 235000 ns | TEST MODE     | out = 00000100
# 245000 ns | TEST MODE     | out = 00001000
# 255000 ns | UPDATING      | out = 00010000
# 265000 ns | UPDATING      | out = 00100000
# === Simulation Complete ===
# ** Note: $finish    : C:/intelFPGA/18.1/tb_sram_clock_chain_divider.v(60)
#    Time: 272 ns  Iteration: 0  Instance: /tb_sram_clock_chain_divider
# 1
Project : ECE240_FinalProject   Now: 272 ns   Delta: 0
```

Figure 8: Simulation Transcript Showing Functional States of SRAM-Based Clock Divider

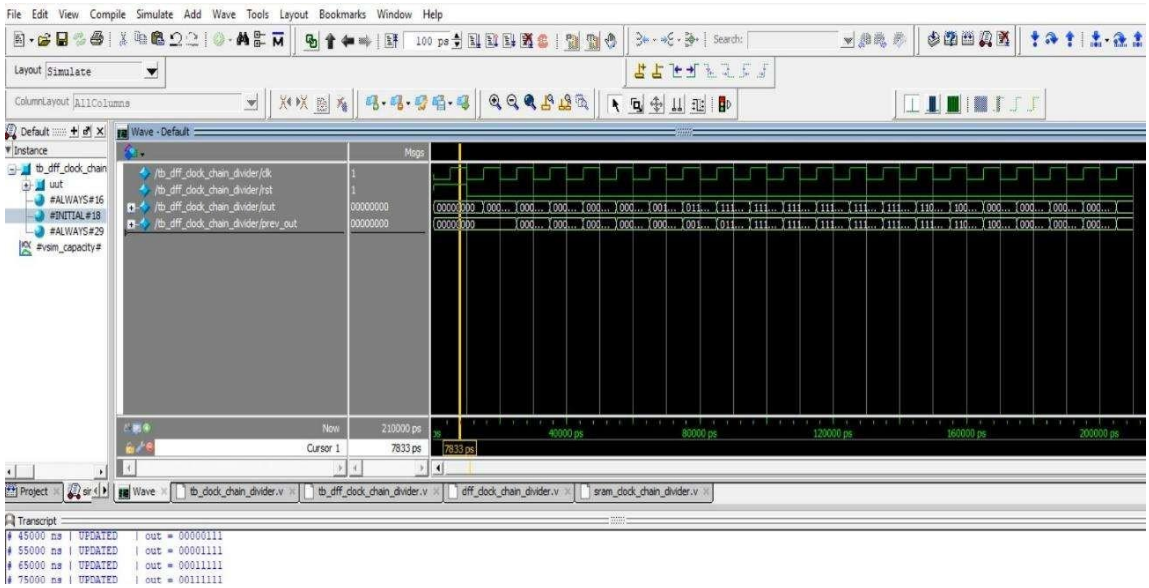


Figure 9: DFF-Based Clock Divider - Simulation Waveform

```
Transcript
add wave -position insertpoint sim:/tb_dff_clock_chain_divider/*
# ** Warning: (vsim-WLF-5000) WLF file currently in use: vsim.wlf
#           File in use by: karth  Hostname: KARTHIK7781  ProcessID: 26776
#           Attempting to use alternate WLF file "./wlft033yzy".
# ** Warning: (vsim-WLF-5001) Could not open WLF file: vsim.wlf
#           Using alternate file: ./wlft033yzy.
VSIM 3> run -all
# 5000 ns | RESET     | out = 00000000
# 15000 ns | HOLDING   | out = 00000000
# 25000 ns | UPDATED   | out = 00000001
# 35000 ns | UPDATED   | out = 00000011
# 45000 ns | UPDATED   | out = 00000111
# 55000 ns | UPDATED   | out = 00001111
# 65000 ns | UPDATED   | out = 00011111
# 75000 ns | UPDATED   | out = 00111111
# 85000 ns | UPDATED   | out = 01111111
# 95000 ns | UPDATED   | out = 11111111
# 105000 ns | UPDATED   | out = 11111110
# 115000 ns | UPDATED   | out = 11111100
# 125000 ns | UPDATED   | out = 11111000
# 135000 ns | UPDATED   | out = 11110000
# 145000 ns | UPDATED   | out = 11100000
# 155000 ns | UPDATED   | out = 11000000
# 165000 ns | UPDATED   | out = 10000000
# 175000 ns | UPDATED   | out = 00000000
# 185000 ns | UPDATED   | out = 00000001
# 195000 ns | UPDATED   | out = 00000011
# 205000 ns | UPDATED   | out = 00000111
# ** Note: $finish    : C:/intelFPGA/18.1/tb_dff_clock_chain_divider.v(26)
#    Time: 210 ns  Iteration: 0  Instance: /tb_dff_clock_chain_divider
# 1
# Break in Module tb_dff_clock_chain_divider at C:/intelFPGA/18.1/tb_dff_clock_chain_divider.v line 26

VSIM 4>
```

Figure 10: DFF-Based Divider - Simulation Transcript

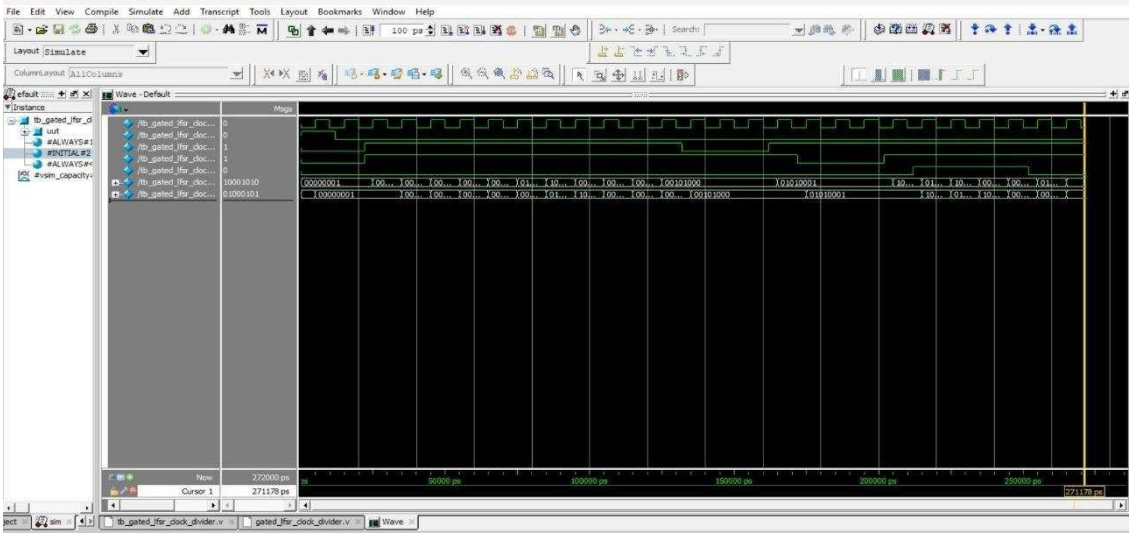


Figure: 11 Gated LFSR-Based Clock Divider - Simulation Waveform

```
Transcript
#           File in use by: karth  Hostname: KARTHIK7781  ProcessID: 26776
#           Attempting to use alternate WLF file "./wlftvjwkv3".
# ** Warning: (vsim-WLF-5001) Could not open WLF file: vsim.wlf
#           Using alternate file: ./wlftvjwkv3
VSIM 3> run -all
# 5000 ns | RESET         | out = 00000001
# 15000 ns | CLOCK GATED   | out = 00000001
# 25000 ns | HOLDING       | out = 00000001
# 35000 ns | UPDATING      | out = 00000010
# 45000 ns | UPDATING      | out = 00000100
# 55000 ns | UPDATING      | out = 00001000
# 65000 ns | UPDATING      | out = 00010000
# 75000 ns | UPDATING      | out = 00100000
# 85000 ns | UPDATING      | out = 01000001
# 95000 ns | UPDATING      | out = 10000010
# 105000 ns | UPDATING      | out = 00000101
# 115000 ns | UPDATING      | out = 00001010
# 125000 ns | UPDATING      | out = 00010100
# 135000 ns | CLOCK GATED   | out = 00101000
# 145000 ns | CLOCK GATED   | out = 00101000
# 155000 ns | CLOCK GATED   | out = 00101000
# 165000 ns | HOLDING       | out = 00101000
# 175000 ns | POWER GATED   | out = 01010001
# 185000 ns | POWER GATED   | out = 01010001
# 195000 ns | POWER GATED   | out = 01010001
# 205000 ns | HOLDING       | out = 01010001
# 215000 ns | TEST MODE     | out = 10100010
# 225000 ns | TEST MODE     | out = 01000100
# 235000 ns | TEST MODE     | out = 10001000
# 245000 ns | TEST MODE     | out = 00010001
# 255000 ns | UPDATING      | out = 00100010
# 265000 ns | UPDATING      | out = 01000101
# === Simulation Complete ===
# ** Note: $finish    : C:/intelFPGA/18.1/tb_gated_lfsr_clock_divider.v(46)
#    Time: 272 ns  Iteration: 0  Instance: /tb_gated_lfsr_clock_divider
# 1
# Break in Module tb_gated_lfsr_clock_divider at C:/intelFPGA/18.1/tb_gated_lfsr_clock_divider.v line 46

VSIM 4>
```

Figure: 12 Gated LFSR-Based Divider - Simulation Transcript

5.2 Synthesis of Implemented design(s):

Step 1: Go to Apporto Login Page, enter credentials and Open Synopsys-DC Compiler

Figure: 13 CentOS Environment for Design Tools

Step 2: Open Documents Folder by Clicking Home on the Desktop and create 3 New Folder(s) for these implemented designs and name them as dff_chain_divider, gated_lfsr_clock_divider, sram_clock_chain_divider as shown in figure below

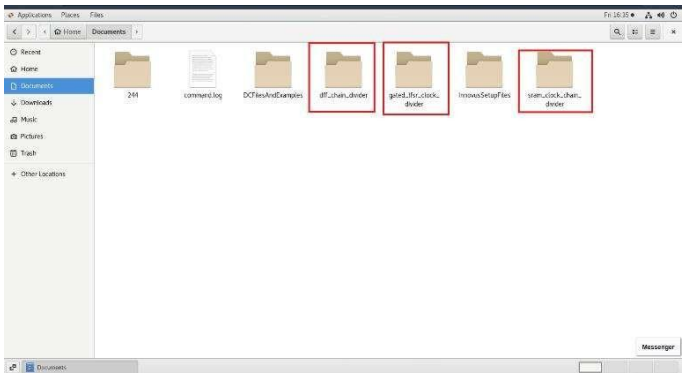


Figure: 14 Project Folder Organization in CentOS

Step 3: Copy-Paste all files from DCFilesandExamples Folder to created sram_clock_chain_divider folder and repeat this for other 2 Folders.

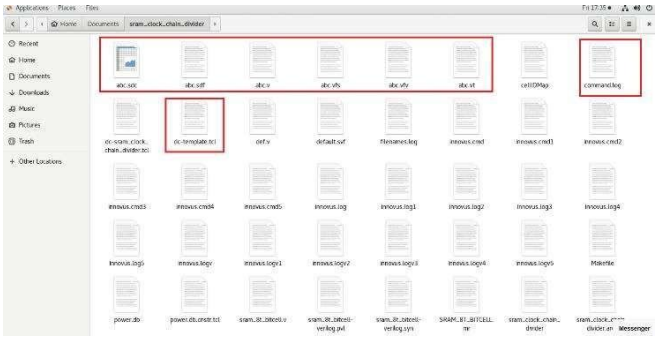


Figure: 15 SRAM Divider Synthesis Files Setup

Step 4: Upload all verilog log files from Computer to Cadence in Apporto and add them to Created ram_clock_chain_divider Folder and repeat this for other 2 folders

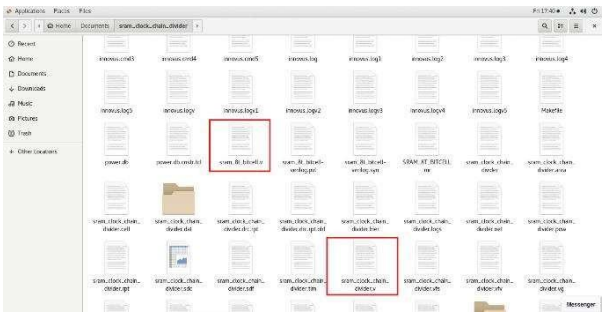


Figure: 16 SRAM Divider Netlist and Output Files

Step 5: Modify .vfs and .vfv files for 3 created folders by adding the Verilog module names and saving them with top module name as shown in the Figures below

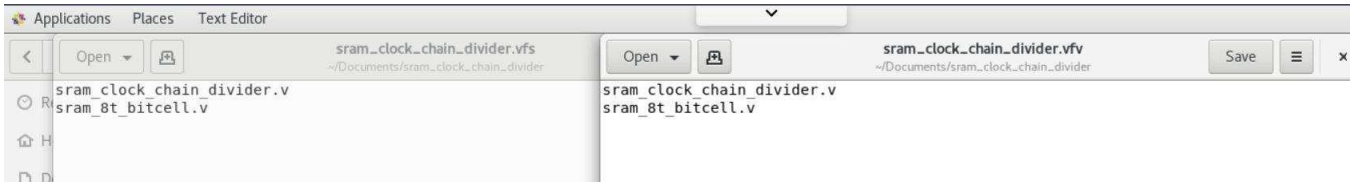


Figure: 17 Verilog Source and Setup Files for Synthesis View of .vfs and .vfv files

Step 6: Now, right click and tap "Open in Terminal" from the options and enter the command **Make NAME=sram_clock_chain_divider synth** for Synthesizing and obtaining Area, Power, Timing and .vg files and repeat this for other 2 implemented designs which are in separate folders.

Figure: 18 Synthesis Command Execution in Terminal

Step 7: Now, save all the reports for 3 implemented designs to proceed further comparison

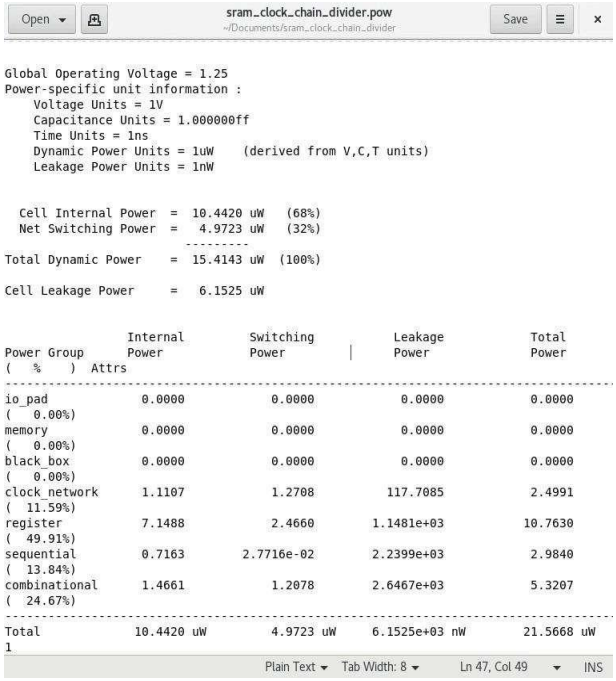


Figure: 19 Area and Power Report of SRAM-Based Clock Divider

Open | sram_clock_chain_divider.tim | ~/Documents/sram_clock_chain_divider | Save

```
****************************************

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

  Startpoint: stage_reg[0]
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: out[7] (output port clocked by clk)
  Path Group: clk
  Path Type: max

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

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clk (rise edge)                    0.00       0.00
  clock network delay (ideal)              0.00       0.00
  stage_reg[0]/CK (DFFR_X1)                0.00       0.00 r
  stage_reg[0]/Q (DFFR_X1)                 0.07       0.07 f
  U45/ZN (NAND2_X1)                        0.04       0.11 r
  U19/ZN (NOR2_X1)                         0.03       0.14 f
  U3/ZN (NAND2_X1)                         0.03       0.16 r
  U33/ZN (TINV_X1)                         0.09       0.25 f
  out[7] (out)                             0.01       0.26 f
  data arrival time                                   0.26

  clock clk (rise edge)                    4.00       4.00
  clock network delay (ideal)              0.00       4.00
  clock uncertainty                       -0.20       3.80
  output external delay                   -0.16       3.64
  data required time                                  3.64
  -----------------------------------------------------------
  data required time                                  3.64
  data arrival time                                  -0.26
  -----------------------------------------------------------
  slack (MET)                                         3.38
```

Plain Text | Tab Width: 8 | Ln 15, Col 1 | INS

Open | sram_clock_chain_divider.vg | ~/Documents/sram_clock_chain_divider | Save

```
/////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : S-2021.06-SP5-1
// Date      : Thu May 15 18:33:23 2025
/////////////////////////////////////////////////////////////


module sram_clock_chain_divider ( clk, rst, en_clk, en_pwr, out );
  output [7:0] out;
  input clk, rst, en_clk, en_pwr;
  wire   N0, N2, gated_clk, \bl[7] , N6, N7, N8, N9, N10, N11, N12, N13, N24,
         N25, n1, n4, n6, n8, n10, n12, n14, n16, n18, n22, n23, n24, n25, n26,
         n27, n28, n29, n30, n31, n32, n33, n34, n35;
  wire   [7:0] wwl;
  wire   [2:0] stage;
  assign N0 = en_clk;
  assign N2 = en_pwr;

  DFF_X1 stored_bit_reg7 ( .D(\bl[7] ), .CK(wwl[7]), .QN(n29) );
  DFFS_X1 bit_in_reg ( .D(n31), .CK(gated_clk), .SN(n30), .Q(\bl[7] ) );
  DFF_X1 stored_bit_reg ( .D(\bl[7] ), .CK(wwl[0]), .QN(n28) );
  DFF_X1 stored_bit_reg1 ( .D(\bl[7] ), .CK(wwl[1]), .QN(n26) );
  DFF_X1 stored_bit_reg2 ( .D(\bl[7] ), .CK(wwl[2]), .QN(n25) );
  DFF_X1 stored_bit_reg3 ( .D(\bl[7] ), .CK(wwl[3]), .QN(n27) );
  DFF_X1 stored_bit_reg4 ( .D(\bl[7] ), .CK(wwl[4]), .QN(n24) );
  DFF_X1 stored_bit_reg5 ( .D(\bl[7] ), .CK(wwl[5]), .QN(n23) );
  DFF_X1 stored_bit_reg6 ( .D(\bl[7] ), .CK(wwl[6]), .QN(n22) );
  NAND2_X1 U3 ( .A1(N2), .A2(N13), .ZN(n18) );
  NAND2_X1 U4 ( .A1(N12), .A2(N2), .ZN(n16) );
  NAND2_X1 U7 ( .A1(N11), .A2(N2), .ZN(n14) );
  NAND2_X1 U8 ( .A1(N10), .A2(N2), .ZN(n12) );
  NAND2_X1 U9 ( .A1(N9), .A2(N2), .ZN(n10) );
  NAND2_X1 U10 ( .A1(N8), .A2(N2), .ZN(n8) );
  NAND2_X1 U11 ( .A1(N7), .A2(N2), .ZN(n6) );
  NAND2_X1 U12 ( .A1(N6), .A2(N2), .ZN(n4) );
  XOR2_X1 U15 ( .A(n34), .B(n1), .Z(N25) );
  NAND2_X1 U16 ( .A1(stage[1]), .A2(stage[0]), .ZN(n1) );
  XOR2_X1 U17 ( .A(stage[1]), .B(stage[0]), .Z(N24) );
  DFFR_X1 \stage_reg[1]  ( .D(N24), .CK(gated_clk), .RN(n30), .Q(stage[1]),
        .QN(n33) );
  DFFR_X1 \stage_reg[0]  ( .D(n32), .CK(gated_clk), .RN(n30), .Q(stage[0]),
        .QN(n32) );
```

C | Tab Width: 8 | Ln 1, Col 1 | INS

Figure: 20 Timing and Gate-Level Netlist Report of SRAM-Based Divider

Open | dff_chain_divider.area | ~/Documents/dff_chain_divider | Save

```
****************************************
Report : area
Design : dff_chain_divider
Version: S-2021.06-SP5-1
Date   : Thu May 15 18:46:53 2025
****************************************

Library(s) Used:

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

Number of ports:                          13
Number of nets:                           35
Number of cells:                          22
Number of combinational cells:            13
Number of sequential cells:                8
Number of macros/black boxes:              0
Number of buf/inv:                         2
Number of references:                      6

Combinational area:                15.428000
Buf/Inv area:                       1.064000
Noncombinational area:             42.560001
Macro/Black Box area:               0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:                   57.988002
Total area:                 undefined
1
```

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

Applications Places Text Editor

Open | dff_chain_divider.pow | ~/Documents/dff_chain_divider

```
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
------------------------------------------------
dff_chain_divider     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  =  16.5741 uW   (73%)
  Net Switching Power  =  6.0608 uW   (27%)
                         ---------
Total Dynamic Power    =  22.6421 uW  (100%)

Cell Leakage Power     =  3.7416 uW


                 Internal         Switching         Leakage          Total
Power Group      Power            Power             Power            Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000           0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000           0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000           0.0000  (   0.00%)
clock_network      1.8629           4.8155          119.2149           5.7977  (  21.97%)
register          14.2743           0.6208         2.2828e+03        17.1771  (  65.10%)
sequential         0.0000           0.0000            0.0000           0.0000  (   0.00%)
combinational      0.6369           1.4325         1.3396e+03        3.4090  (  12.92%)
--------------------------------------------------------------------------------------------
Total             16.5741 uW        6.0608 uW     3.7416e+03 nW      26.3837 uW
1
```

Figure: 21 Area and Power Report of DFF-Based Clock Divider

```
Applications  Places  Text Editor
Open                                   dff_chain_divider.tim
                                       ~/Documents/dff_chain_divider

        -path full
        -delay max
        -nworst 10
        -max_paths 10
Design : dff_chain_divider
Version: S-2021.06-SP5-1
Date   : Thu May 15 18:46:53 2025
****************************************


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

  Startpoint: test_mode (input port clocked by clk)
  Endpoint: dc_reg[0] (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

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

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clk (rise edge)                    0.00       0.00
  clock network delay (ideal)              0.00       0.00
  input external delay                     0.12       0.12 f
  test_mode (in)                           0.00       0.12 f
  U16/ZN (NOR2_X1)                         0.08       0.20 r
  U28/ZN (INV_X1)                          0.06       0.27 f
  U24/ZN (OAI22_X1)                        0.05       0.32 r
  dc_reg[0]/D (DFFR_X1)                    0.01       0.33 r
  data arrival time                                   0.33

  clock clk (rise edge)                    4.00       4.00
  clock network delay (ideal)              0.00       4.00
  clock uncertainty                       -0.20       3.80
  dc_reg[0]/CK (DFFR_X1)                   0.00       3.80 r
  library setup time                      -0.03       3.77
  data required time                                  3.77
  -----------------------------------------------------------
  data required time                                  3.77
  data arrival time                                  -0.33
  -----------------------------------------------------------
  slack (MET)                                         3.44

dff_chain_divider   [c21870...csufresno@nc-csual4-01]...   dff_chain_divider.tim (~/Documents...
```

```
nc-hyperstream.apportto.com/hyperstream/#/client/MTI0NDUxNTEAYwBlbmNyaXB0ZWR1cmwtamRYes=?qs=PKFDcjHPDOayd1m1fy7774hEeQDIFjmAEzEvqfU
Applications  Places  Text Editor
Open                                   dff_chain_divider.vg
                                       ~/Documents/dff_chain_divider
//////////////////////////////////////////////////////////////
// Created by: Synopsys DC Expert(TM) in wire load mode
// Version   : S-2021.06-SP5-1
// Date      : Thu May 15 18:46:53 2025
//////////////////////////////////////////////////////////////


module dff_chain_divider ( clk, rst, en_clk, en_pwr, test_mode, dc );
  output [7:0] dc;
  input clk, rst, en_clk, en_pwr, test_mode;
  wire   N0, gated_clk, n1, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14,
         n15, n16, n17, n18, n20, n21, n22;

  DFFR_X1 \dc_reg[0]  ( .D(n20), .CK(gated_clk), .RN(n21), .Q(dc[0]), .QN(n10)
          );
  DFFR_X1 \dc_reg[1]  ( .D(n18), .CK(gated_clk), .RN(n21), .Q(dc[1]), .QN(n9)
          );
  DFFR_X1 \dc_reg[2]  ( .D(n17), .CK(gated_clk), .RN(n21), .Q(dc[2]), .QN(n8)
          );
  DFFR_X1 \dc_reg[3]  ( .D(n16), .CK(gated_clk), .RN(n21), .Q(dc[3]), .QN(n7)
          );
  DFFR_X1 \dc_reg[4]  ( .D(n15), .CK(gated_clk), .RN(n21), .Q(dc[4]), .QN(n6)
          );
  DFFR_X1 \dc_reg[5]  ( .D(n14), .CK(gated_clk), .RN(n21), .Q(dc[5]), .QN(n5)
          );
  DFFR_X1 \dc_reg[6]  ( .D(n13), .CK(gated_clk), .RN(n21), .Q(dc[6]), .QN(n4)
          );
  DFFR_X1 \dc_reg[7]  ( .D(n12), .CK(gated_clk), .RN(n21), .Q(dc[7]), .QN(n11)
          );
  NOR2_X1 U16 ( .A1(en_pwr), .A2(test_mode), .ZN(n1) );
  OAI22_X1 U17 ( .A1(n4), .A2(n1), .B1(n11), .B2(n22), .ZN(n12) );
  OAI22_X1 U18 ( .A1(n5), .A2(n1), .B1(n4), .B2(n22), .ZN(n13) );
  OAI22_X1 U19 ( .A1(n6), .A2(n1), .B1(n5), .B2(n22), .ZN(n14) );
  OAI22_X1 U20 ( .A1(n7), .A2(n1), .B1(n6), .B2(n22), .ZN(n15) );
  OAI22_X1 U21 ( .A1(n8), .A2(n1), .B1(n7), .B2(n22), .ZN(n16) );
  OAI22_X1 U22 ( .A1(n9), .A2(n1), .B1(n8), .B2(n22), .ZN(n17) );
  OAI22_X1 U23 ( .A1(n10), .A2(n1), .B1(n9), .B2(n22), .ZN(n18) );
  OAI22_X1 U24 ( .A1(n1), .A2(dc[7]), .B1(n10), .B2(n22), .ZN(n20) );
  OR2_X1 U25 ( .A1(en_clk), .A2(test_mode), .ZN(N0) );
  MUX2_X1 U26 ( .A(1'b0), .B(clk), .S(N0), .Z(gated_clk) );
  INV_X1 U27 ( .A(rst), .ZN(n21) );
  INV_X1 U28 ( .A(n1), .ZN(n22) );
endmodule

dff_chain_divider   [c21870...csufresno@nc-csual4-01]...   dff_chain_divider.vg (~/Documents/...
```

Figure: 23 Timing and Gate-Level Netlist Report of DFF-Based Divider

```
Open                                   gated_lfsr_clock_divider.area
                                       ~/Documents/gated_lfsr_clock_divider                    Save
****************************************
Report : area
Design : gated_lfsr_clock_divider
Version: S-2021.06-SP5-1
Date   : Thu May 15 18:56:21 2025
****************************************

Library(s) Used:

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

Number of ports:                          13
Number of nets:                           36
Number of cells:                          23
Number of combinational cells:            14
Number of sequential cells:                8
Number of macros/black boxes:              0
Number of buf/inv:                         2
Number of references:                      7

Combinational area:                17.024000
Buf/Inv area:                       1.064000
Noncombinational area:             42.560001
Macro/Black Box area:               0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:                   59.584002
Total area:                 undefined
1

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

```
Open                                   gated_lfsr_clock_divider.pow
                                       ~/Documents/gated_lfsr_clock_divider                    Save
gated_lfsr_clock_divider
                    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  =  17.9637 uW   (74%)
  Net Switching Power  =   6.3034 uW   (26%)
                         ---------
Total Dynamic Power    =  24.2671 uW  (100%)

Cell Leakage Power     =   3.5941 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power
(   %    )  Attrs
--------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000            0.0000
(   0.00%)
memory             0.0000           0.0000            0.0000            0.0000
(   0.00%)
black_box          0.0000           0.0000            0.0000            0.0000
(   0.00%)
clock_network      1.6626           3.9657          119.2149            5.7475
(  20.63%)
register          15.3528           0.8711        2.2878e+03           18.5117
(  66.44%)
sequential         0.0000           0.0000            0.0000            0.0000
(   0.00%)
combinational      0.9482           1.4667        1.1872e+03            3.6020
(  12.93%)
--------------------------------------------------------------------------------
Total             17.9637 uW        6.3034 uW     3.5941e+03 nW        27.8612 uW
1

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

Figure: 24 Area and Power Report of Gated LFSR-Based Clock Divider

Open gated_lfsr_clock_divider.vg Save
~/Documents/gated_lfsr_clock_divider

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


module gated_lfsr_clock_divider ( clk, rst, en_clk, en_pwr, test_mode,
      lfsr_out );
  output [7:0] lfsr_out;
  input clk, rst, en_clk, en_pwr, test_mode;
  wire   N0, gated_clk, n1, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13,
         n14, n15, n16, n17, n18, n20, n21, n22;

  DFFS_X1 \lfsr_out_reg[0]  ( .D(n20), .CK(gated_clk), .SN(n21), .Q(
        lfsr_out[0]), .QN(n9) );
  DFFS_X1 \lfsr_out_reg[1]  ( .D(n18), .CK(gated_clk), .SN(n21), .Q(
        lfsr_out[1]), .QN(n8) );
  DFFS_X1 \lfsr_out_reg[2]  ( .D(n17), .CK(gated_clk), .SN(n21), .Q(
        lfsr_out[2]), .QN(n7) );
  DFFS_X1 \lfsr_out_reg[3]  ( .D(n16), .CK(gated_clk), .SN(n21), .Q(
        lfsr_out[3]), .QN(n6) );
  DFFS_X1 \lfsr_out_reg[4]  ( .D(n15), .CK(gated_clk), .SN(n21), .Q(
        lfsr_out[4]), .QN(n5) );
  DFFS_X1 \lfsr_out_reg[5]  ( .D(n14), .CK(gated_clk), .SN(n21), .Q(
        lfsr_out[5]), .QN(n10) );
  DFFS_X1 \lfsr_out_reg[6]  ( .D(n13), .CK(gated_clk), .SN(n21), .Q(
        lfsr_out[6]), .QN(n4) );
  DFFS_X1 \lfsr_out_reg[7]  ( .D(n12), .CK(gated_clk), .SN(n21), .Q(
        lfsr_out[7]), .QN(n11) );
  XNOR2_X1 U11 ( .A(n11), .B(n10), .ZN(n3) );
  NOR2_X1 U16 ( .A1(en_pwr), .A2(test_mode), .ZN(n1) );
  OAI22_X1 U17 ( .A1(n4), .A2(n1), .B1(n11), .B2(n22), .ZN(n12) );
  OAI22_X1 U18 ( .A1(n10), .A2(n1), .B1(n4), .B2(n22), .ZN(n13) );
  OAI22_X1 U19 ( .A1(n5), .A2(n1), .B1(n10), .B2(n22), .ZN(n14) );
  OAI22_X1 U20 ( .A1(n6), .A2(n1), .B1(n5), .B2(n22), .ZN(n15) );
  OAI22_X1 U21 ( .A1(n7), .A2(n1), .B1(n6), .B2(n22), .ZN(n16) );
  OAI22_X1 U22 ( .A1(n8), .A2(n1), .B1(n7), .B2(n22), .ZN(n17) );
  OAI22_X1 U23 ( .A1(n9), .A2(n1), .B1(n8), .B2(n22), .ZN(n18) );
  OAI22_X1 U24 ( .A1(n9), .A2(n22), .B1(n1), .B2(n3), .ZN(n20) );
  OR2_X1 U25 ( .A1(en_clk), .A2(test_mode), .ZN(N0) );
  MUX2_X1 U26 ( .A(1'b0), .B(clk), .S(N0), .Z(gated_clk) );
  INV_X1 U27 ( .A(rst), .ZN(n21) );
  INV_X1 U28 ( .A(n1), .ZN(n22) );
endmodule
```

C Tab Width: 8 Ln 1, Col 1 INS

Open gated_lfsr_clock_divider.tim Save
~/Documents/gated_lfsr_clock_divider

```
Version: S-2021.06-SP5-1
Date   : Thu May 15 18:56:22 2025
****************************************

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

  Startpoint: test_mode (input port clocked by clk)
  Endpoint: lfsr_out_reg[1]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

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

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clk (rise edge)                    0.00       0.00
  clock network delay (ideal)              0.00       0.00
  input external delay                     0.12       0.12 f
  test_mode (in)                           0.00       0.12 f
  U16/ZN (NOR2_X1)                         0.08       0.20 r
  U28/ZN (INV_X1)                          0.06       0.27 f
  U23/ZN (OAI22_X1)                        0.05       0.32 r
  lfsr_out_reg[1]/D (DFFS_X1)              0.01       0.33 r
  data arrival time                                   0.33

  clock clk (rise edge)                    4.00       4.00
  clock network delay (ideal)              0.00       4.00
  clock uncertainty                       -0.20       3.80
  lfsr_out_reg[1]/CK (DFFS_X1)             0.00       3.80 r
  library setup time                      -0.02       3.78
  data required time                                  3.78
  -----------------------------------------------------------
  data required time                                  3.78
  data arrival time                                  -0.33
  -----------------------------------------------------------
  slack (MET)                                         3.44
```

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

Figure: 25 Timing and Gate-Level Netlist Report of Gated LFSR-Based Divider

Design Metrics Evaluation and Preferred Architecture Selection

To determine the most power- and area-efficient architecture for a clock chain divider, three RTL-level implementations — DFF-based, gated LFSR-based, and symbolic 8T SRAM-based — were designed and functionally verified through simulation. Post-synthesis analysis was carried out using Synopsys Design Compiler, and all designs were evaluated based on their cell area, power consumption, and timing performance under the same library and operating conditions.

The synthesis results are summarized below:

Sno.	Metric	DFF-Based Divider	Gated LFSR Divider	8T SRAM-

				Based Divider
1	Total Cell Area (μm²)	57.99	59.58	102.94
2	Combinational Area (μm²)	15.43	17.02	36.97
3	Sequential Cells	8	8	20
4	Total Dynamic Power (μW)	20.63	24.27	15.41
5	Total Power (μW)	26.38	27.86	21.57
6	Worst Case Slack (ns)	3.44	3.44	3.38

Table 1: Comparison os all divider designs

Among the three architectures, the SRAM-based clock chain divider shows clear advantages in dynamic power consumption while maintaining acceptable area and timing performance. Though slightly higher in total area due to the unrolled memory-like structure, it offers a well-balanced trade-off for low-power system design.

Considering its internal gating behavior, efficient state retention, and reduced switching activity, the SRAM-based architecture stands out as the most energy-efficient and scalable option for advanced SoC environments such as RISC-V–based processors.

As the next step, this optimized design is intended for physical implementation in Cadence Innovus, where critical stages such as floor planning, power planning, placement, and special and detailed routing will be performed. This will be followed by power analysis to assess layout-level power integrity and validate the design's feasibility for low-power physical design workflows.

5.3 Obtaining RTL Schematic Design for SRAM Based Clock chain divider using Design Vision:

Step 1: Right click on the created sram_clock_chain_divider folder and click "Open in Terminal" and enter "design_vision" command to open gui

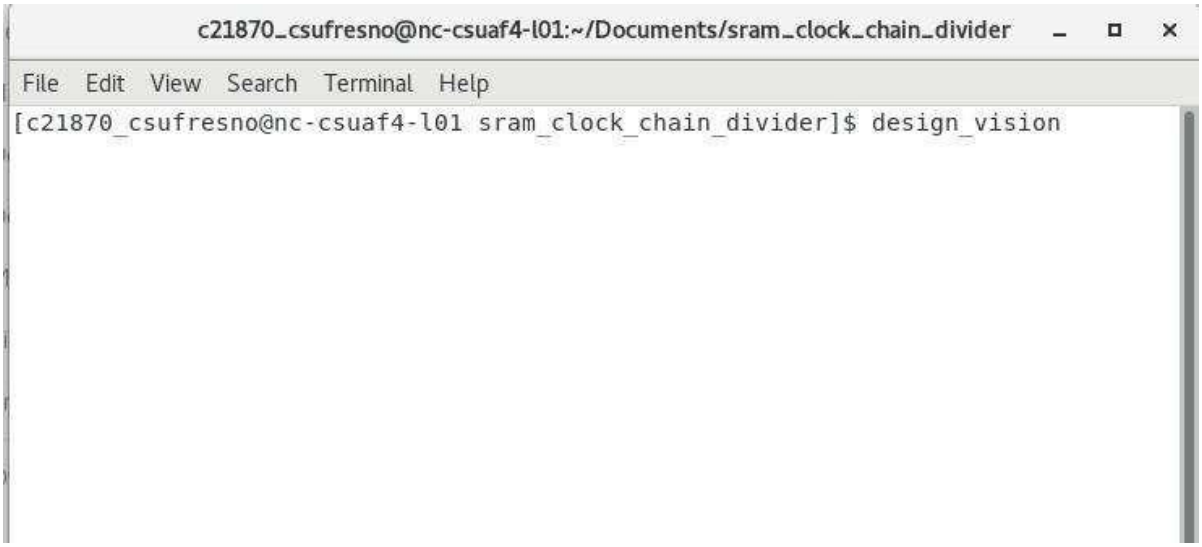


Figure: 26 Launching Design Vision for RTL Schematic Analysis

Step 2: Nextly, an interface is opened. Click on File-> Read> Read Design-> sram_clock_chain_divider.v-> Open

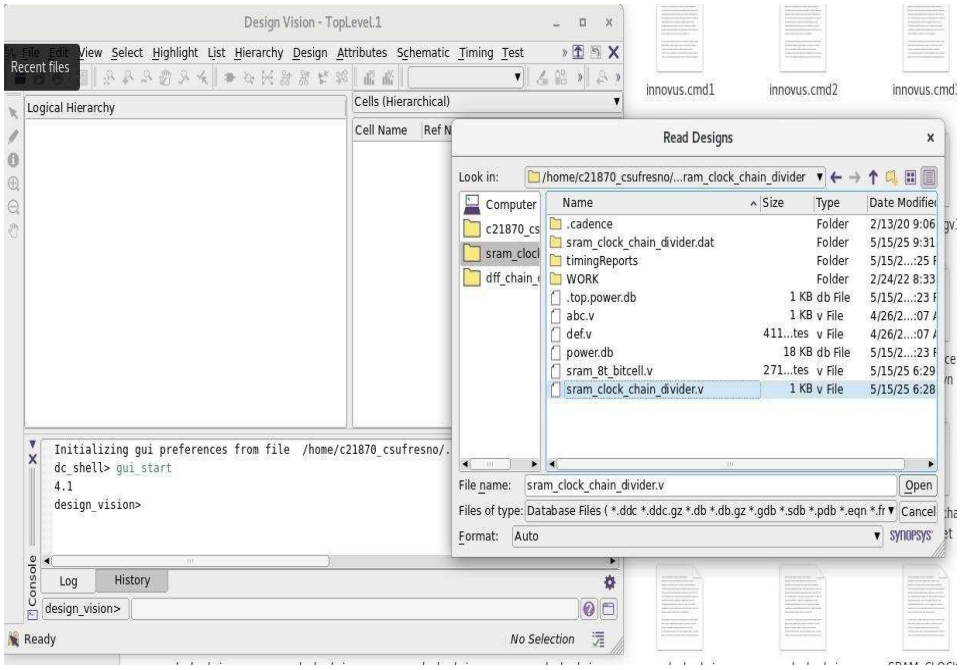


Figure: 27 Loading Synthesized Design into Design Vision

Step 3: Nextly, all the cell(s) present in the logical hierarchy of the design will be displayed on the left pane as shown in the figure below

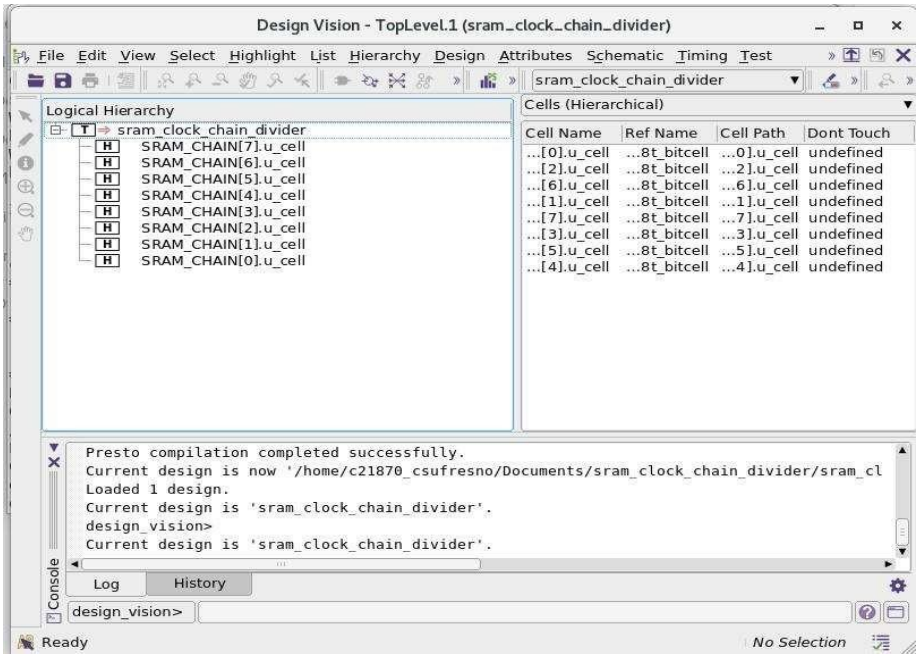


Figure: 28 RTL Hierarchy View in Design Vision

Step 4: Right click on sram_clock_chain_divider on logical hierarchy and select schematic view from the options menu. And click on zoom-in to obtain clear schematic view for the desired implementation

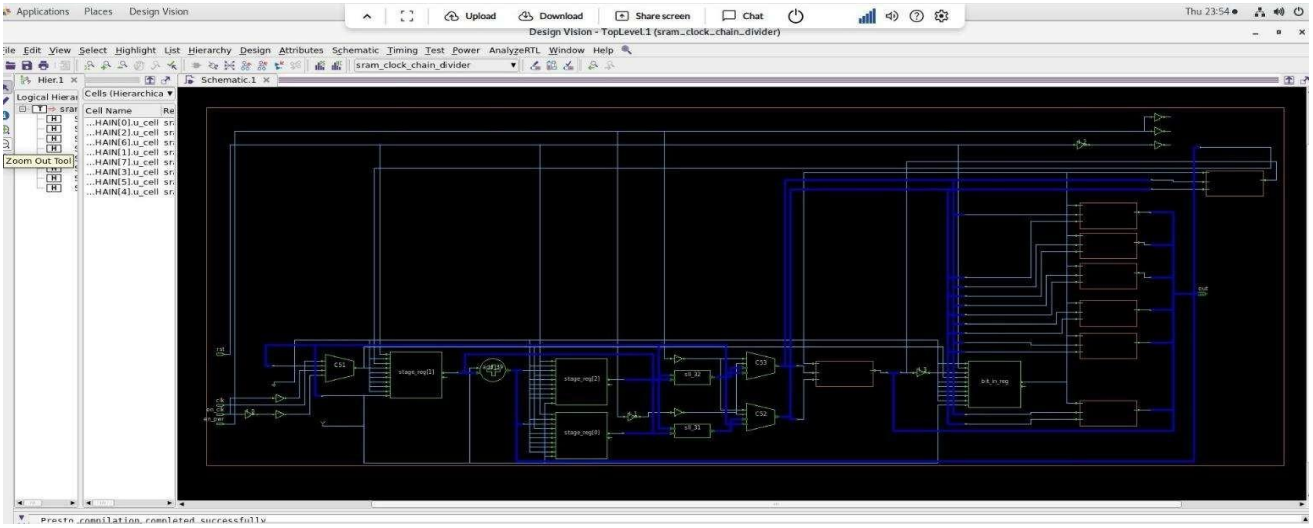


Figure: 29 RTL Schematic in Design Vision

5.4 Post Synthesis of SRAM Based Clock Chain Divider:

Step 1: After Synthesis, head over to InnovusSetupFiles Folder in the documents and copy template.globals and template.view files and paste them in the sram_clock_chain_divider folder

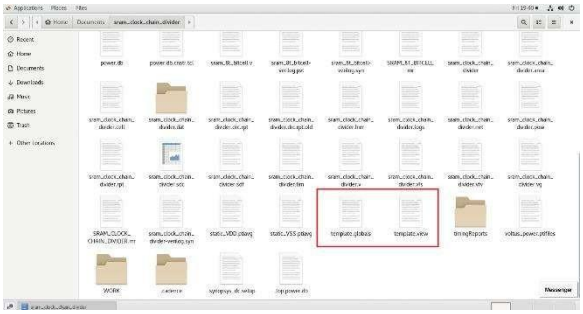


Figure: 29 Floorplan Setup

Step 2: Open both the files, By using Find and Replace from the options bar, Replace "abc" with "sram_clock_chain_divider" and enter Replace all

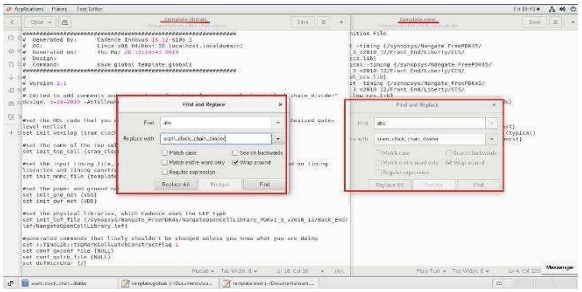
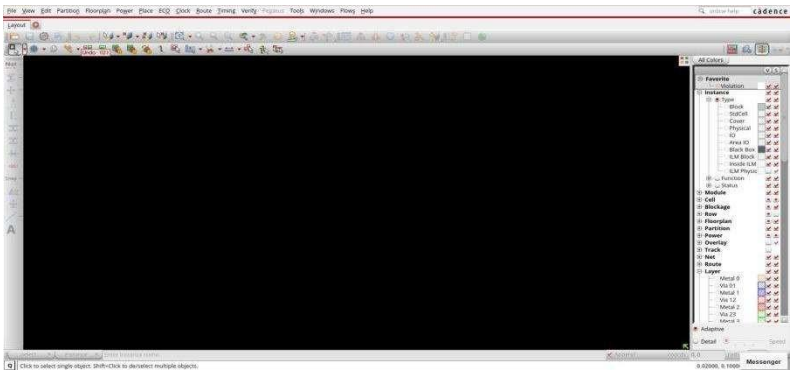


Figure: 30 Editing Constraint Files for Physical Design Setup

Step 3: Now, Right click on the sram_clock_chain_divider Folder and select "Open in Terminal" and



type "innovus" and click Enter

Figure: 31 Cadence Innovus GUI Launched for Physical Design

Step 4: Go to File-> Import Design-> Load ->Template.globals -> OK

After importing the gate-level netlist and constraint file into Cadence Innovus, the tool initialized the

layout environment. The design area was loaded with default rows automatically generated by the tool.

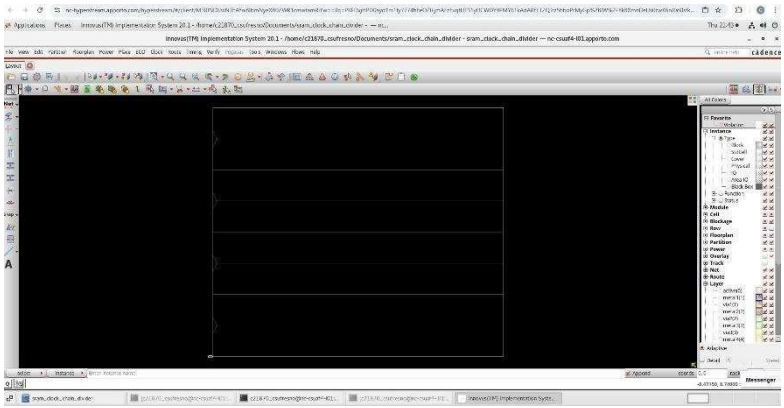


Figure: 32 Initial Floorplan Layout in Cadence Innovus

Step 5: Connect Global nets VSS, VDD Power Option in the Menu and Perform Floor Planing

The next step was to define the floorplan. The Floorplan menu was accessed, and the Specify Floorplan and Generate Floorplan options were used to define the core area, margins, and number of rows. This step prepared the layout space for proper standard cell placement and I/O pin alignment, as per the tutorial instructions.

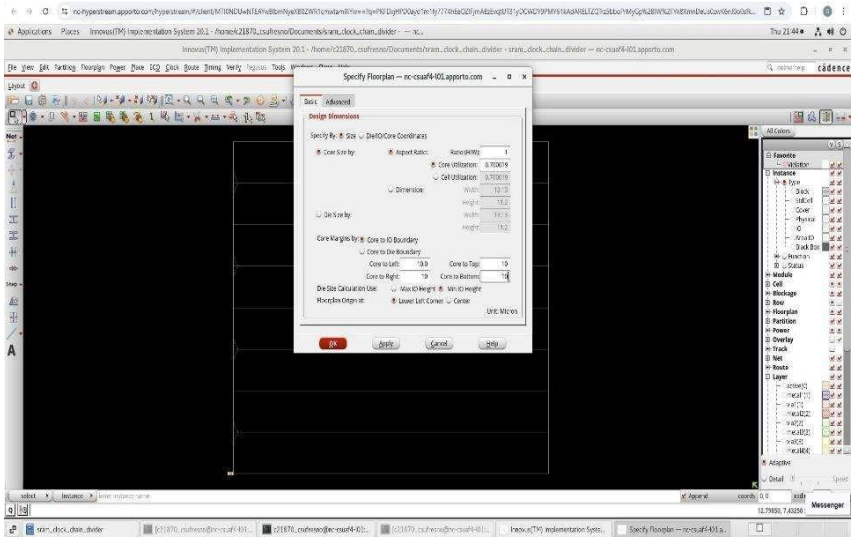


Figure: 33 Specifying Floorplan Parameters in Innovus

Step 6: Power planning- Add Rings:

After floor planning, the Power menu was accessed to begin the power planning process. Power rings were added around the core area to ensure proper power distribution throughout the design. In the "Add Rings" configuration window, the power ring parameters were defined. The ring configuration used metal9 for top and bottom, and metal10 for left and right, with a uniform width and spacing of 1.8 microns. These settings followed the tutorial's suggested default values for effective power grid creation.

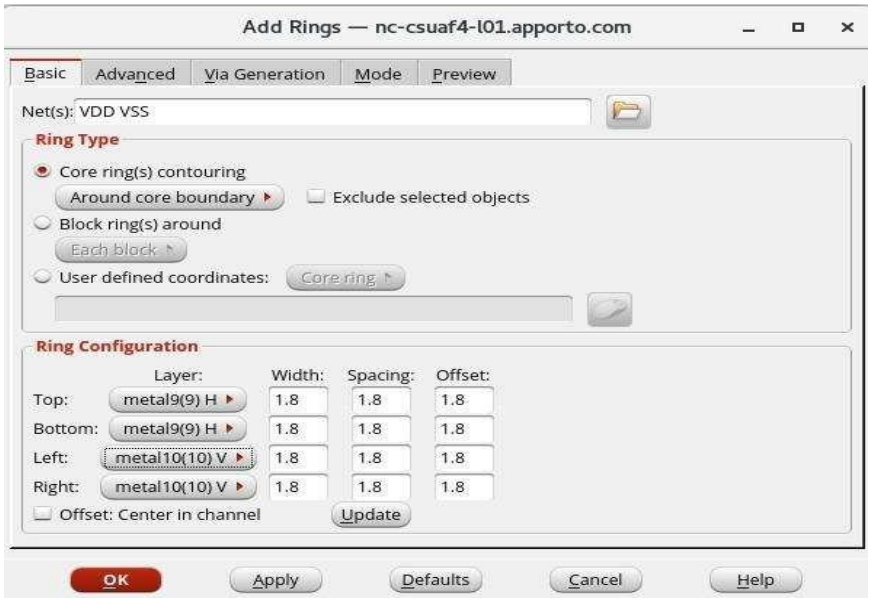


Figure: 34 Power Ring Configuration in Cadence Innovus

Step 7: Floor plan view with power rings

Once the power ring configuration was applied, the layout view updated to show the generated power rails. VDD and VSS rails were wrapped around the design core using the specified metal layers, enabling robust power delivery during placement and routing.

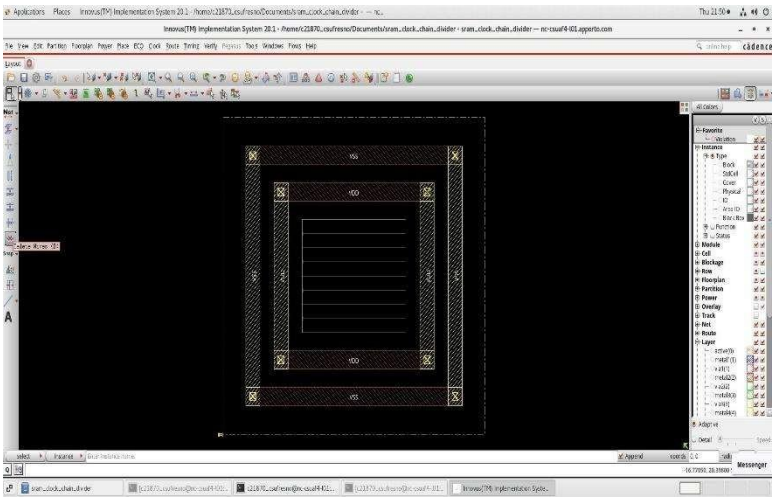


Figure: 35 Power Ring Generation in Innovus Layout View

Step 8: Route- >Special Route

After completing floorplanning and power planning, the routing process was initiated. In the Route menu, the Special Route option was selected to create dedicated routing paths for global power nets like VDD and VSS. This step ensures that power delivery is established through the predefined metal layers before starting regular signal routing.

The special routing defines high-priority connections that support power integrity and timing.

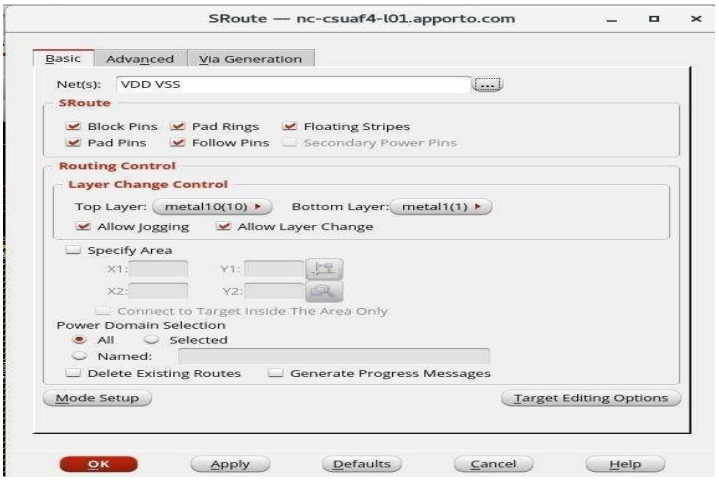
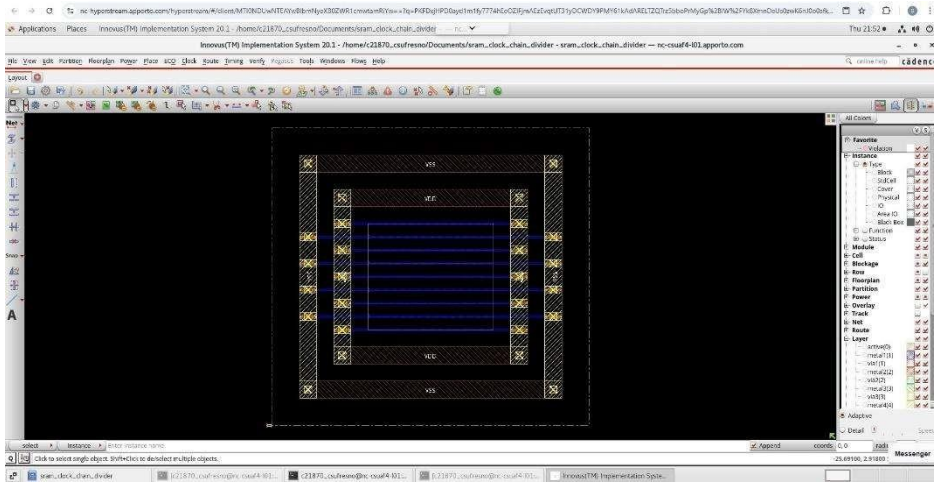


Figure: 36 Special Route (SRoute) Configuration for Power Connections

Step 9: Specify Route

During special routing, advanced settings were applied to configure power net routing. In the SRoute dialog box, VDD and VSS nets were selected for routing. Top and bottom metal layers were defined as metal10 and metal1 respectively, allowing vertical and horizontal connectivity across layers. Options such as Pad Rings, Pad Pins, and Floating Stripes were enabled to ensure proper power connectivity across the core and I/O. Jogging and layer change were also allowed to provide routing flexibility where needed. This setup helped the tool efficiently connect power rails before proceeding



with full signal using nanoroute.

Figure: 37 Power Stripe Routing Completed in Innovus

Step 10: Add Stripes

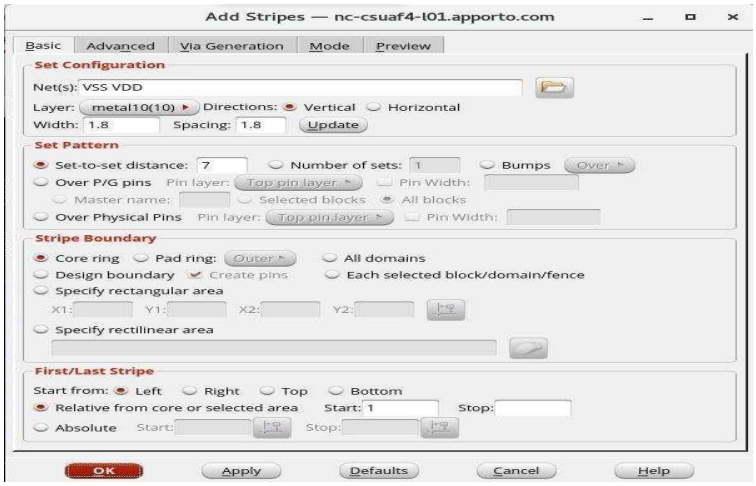
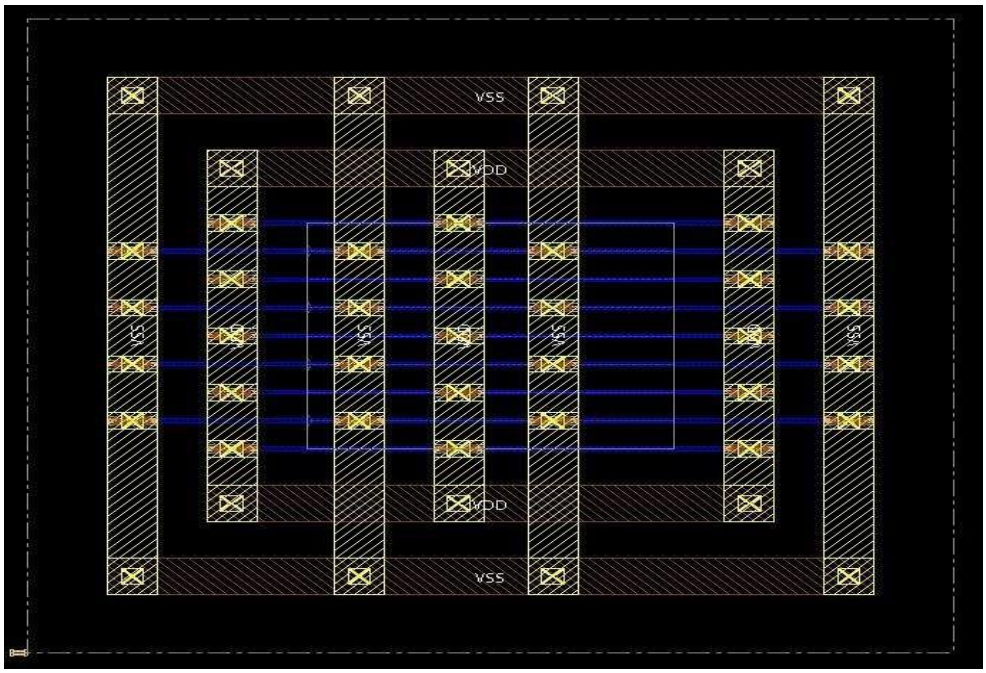


Figure: 38 Add Stripes Configuration for Power Routing



In this step, VDD and VSS power stripes were added using the Add Stripes option. This helped strengthen the power grid before final.

Figure: 39 Final Power Grid View with Rings and Stripes**Step 11: Placement**

Standard cells were placed using the **Place → Place Standard Cell** option in Innovus. This ensured

all logic cells were arranged optimally before routing

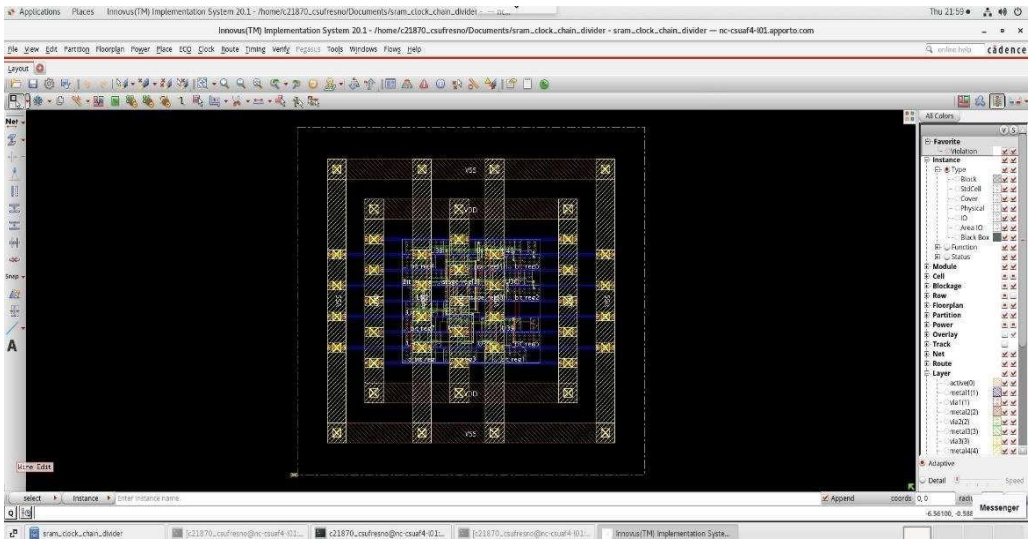


Figure: 40 Standard Cell Placement within Power Grid in Innovus

Step 12: Verify DRC

The "Verify DRC" option was selected under the “Verify” menu to check design rule violations such as spacing, enclosure, and layer constraints.

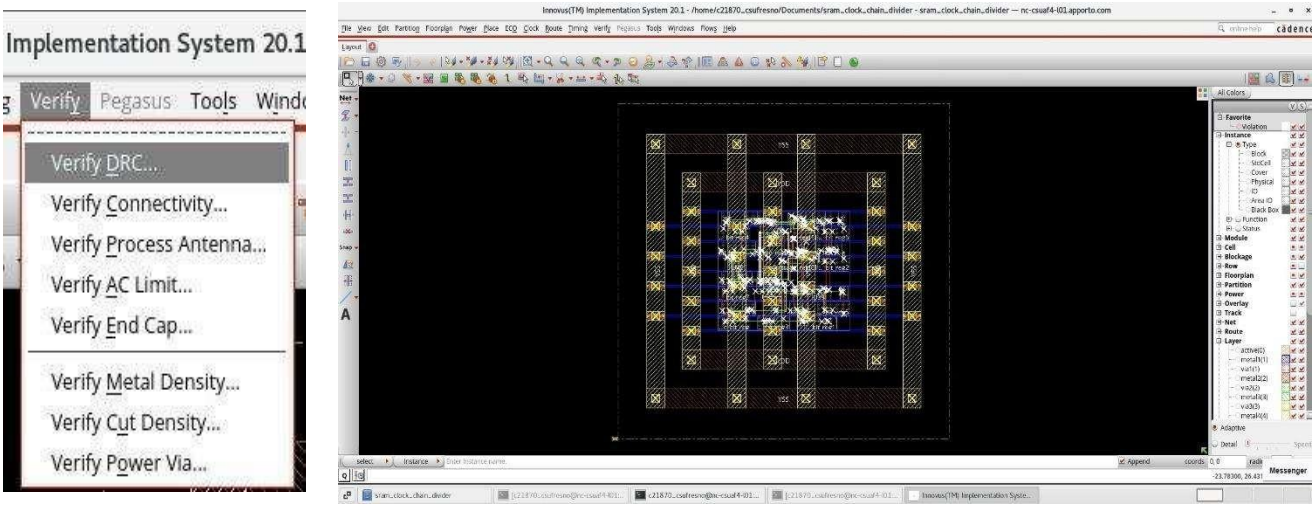


Figure: 41 Post-Placement Layout and DRC Verification in Innovus

The white color lines in the design highlights these violations. These violations might be eliminated using by performing Early global routing and nano routing

Step 13: Performing Clock Tree Synthesis

We need to open the terminal and type following commands one-by-one

1. set_ccopt_property cts_buffer_cells {BUF_X1 BUF_X2}
2. create_ccopt_clock_tree_spec
3. ccopt_design

Click on clock-> ClockDebugger to obtain relevant clock tree

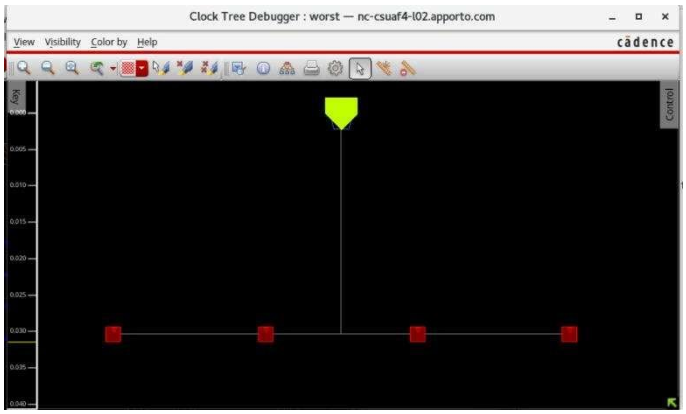


Figure: 42 Clock Tree Debugger View in Innovus

Observation: The Clock Tree Debugger view in Cadence Innovus illustrates a simple clock routing topology from a single clock source to four clock sinks without any intermediate clock buffers. This flat routing structure is a result of the limited number of clocked elements in the design and the absence of compatible buffer cells in the loaded technology library. Despite the absence of buffer-driven CTS, the design maintains minimal skew due to its small size, and the direct routing satisfies functional and timing requirements.

Step 14: Early Global Routing & Nano Routing

Go to Route-> Early Global Route

Early global routing is done to quickly estimate routing congestion, wirelength, and timing impacts after placement but before detailed routing, allowing designers to make early optimizations and avoid routing problems later in the physical design flow.

Go to Route-> Nano Route: It is selected to perform detailed routing and finalize metal connections.

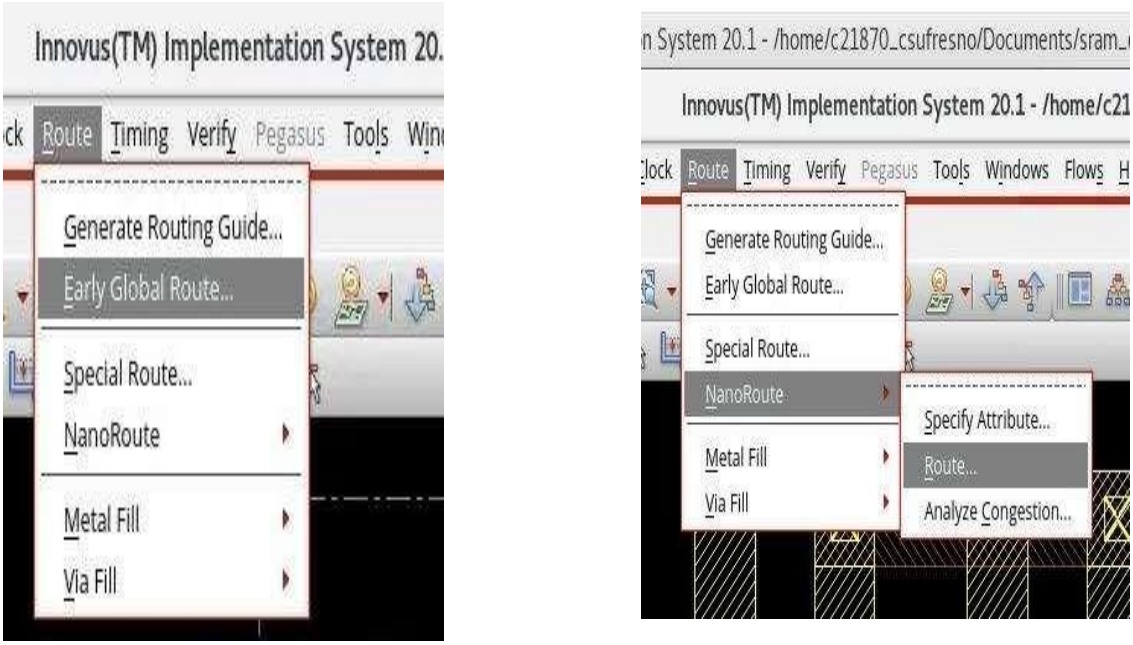


Figure: 43 Routing Options in Innovus: Early Global and NanoRoute

Step 15: Re-Verify DRC

Now, reverify the drc to obtain the implemented design without white lines or violations and save the obtained image

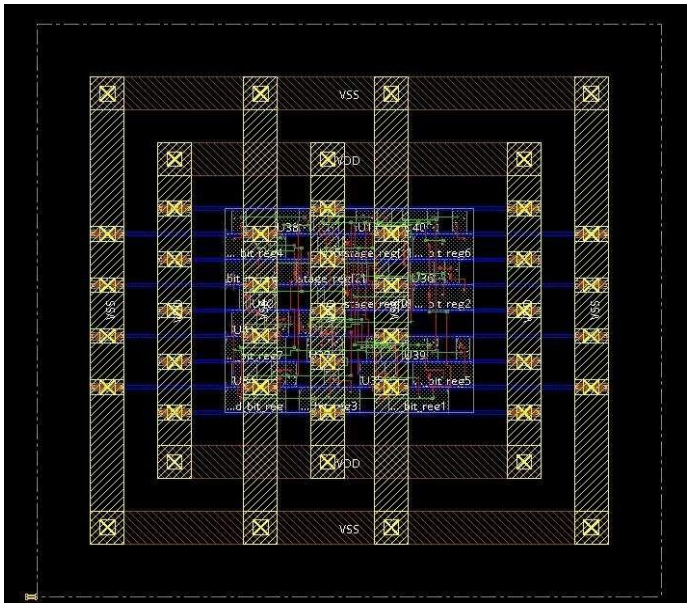


Figure: 44 Final Routed Layout of SRAM-Based Clock Divider

Step 16: ADD Fillers

Go to **Place → Physical Cell → Add I/O Filler**.

These fillers are critical for maintaining continuity between I/O cells. They help prevent routing violations and ensure proper metal connectivity. This step typically follows final placement and precedes tape out preparation.

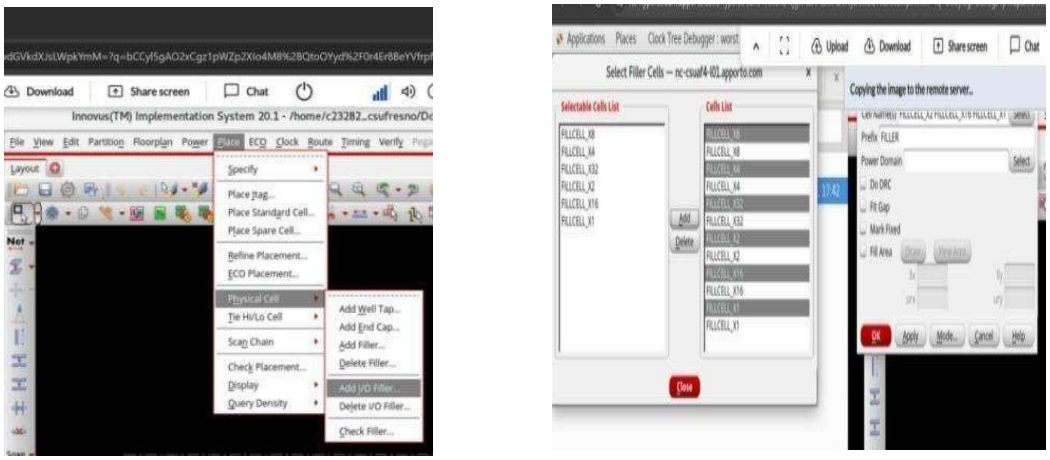


Figure: 45 Filler Cell Insertion in Cadence Innovus

Step 17: After adding the fillers, the final physical design can be obtained as shown in the figure below.

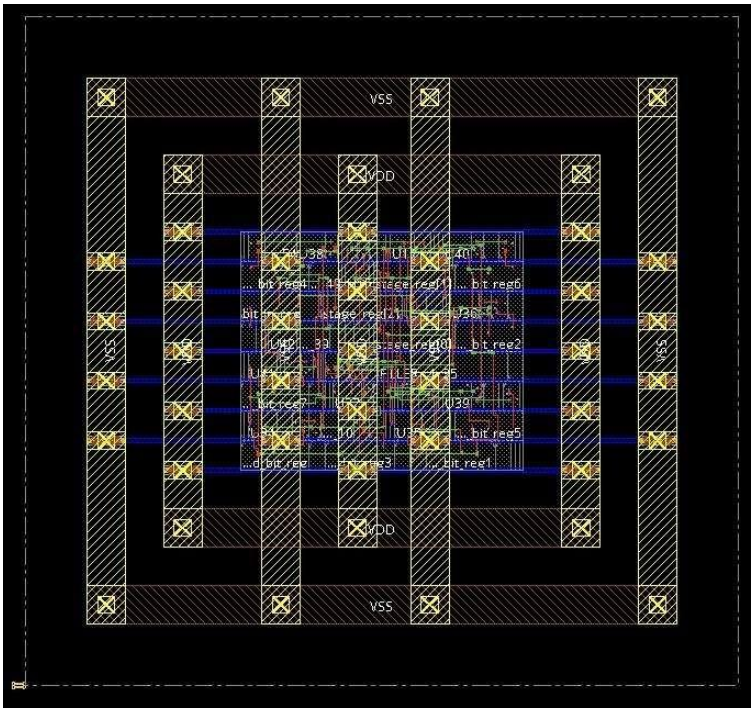


Figure: 46 Final Layout after Filler Cell Insertion

Step 18: Now, power analysis is performed to check the power-based metrics for the implemented design

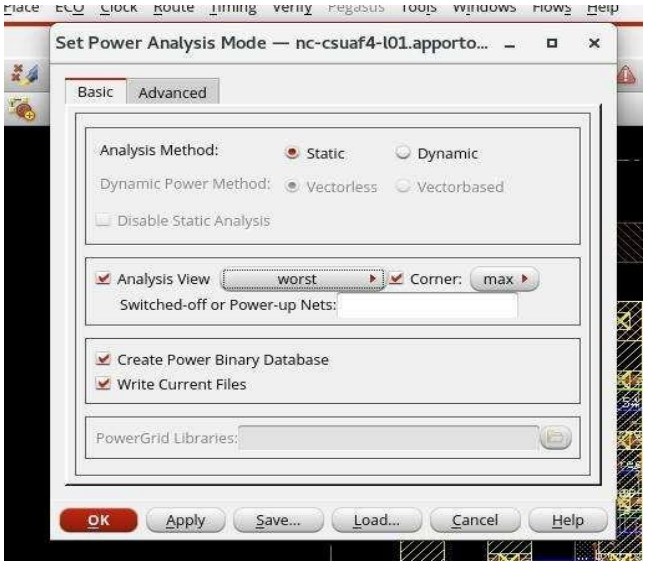


Figure: 47 Power Analysis Setup in Cadence Innovus

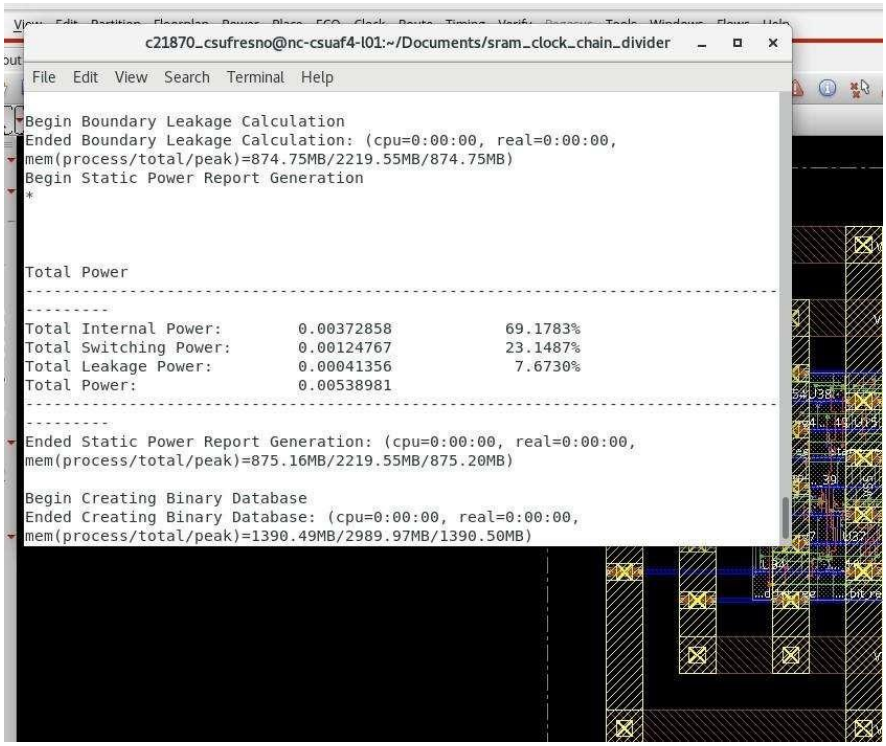


Figure: 48 Static Power Analysis Report Output

6.Future Scope

In the next phase of this project, the SRAM-based clock chain divider can undergo full post-route static timing analysis and power verification using Cadence Innovus to ensure its reliability at the physical design level. Additionally, the design can be scaled to support wider bit-lengths such as 16-bit or 32-bit versions, making it suitable for integration in larger digital systems.

Another important direction is the integration of the divider into a RISC-V processor subsystem, where it can serve real clock management functions in a low-power environment. Lastly, the design and implementation flow can be further optimized through scripting and automation, while also exploring advanced power-saving techniques like multi-Vt cell usage and dynamic gating for better energy efficiency.

Challenges Faced

Throughout the course of this project, we encountered several technical and design-related challenges. One of the key challenges was modeling the 8T SRAM-based architecture in Verilog. Since SRAM cells are not standard sequential elements, we had to creatively simulate their read/write behavior while preserving correct timing and control using symbolic logic.

Integrating clock gating and power gating at the RTL level across all three architectures also required careful signal handling to ensure that functional correctness was maintained during gating transitions. Another significant challenge was ensuring compatibility between our synthesized designs and backend tools like Cadence Innovus, which demanded correct constraint setup and clean netlists.

Additionally, interpreting synthesis and simulation reports to extract meaningful power and timing data involved multiple iterations and fine-tuning of testbenches and scripts. These challenges ultimately helped us gain deeper insight into the full digital design flow, from RTL to physical design planning.

7. Conclusion

This project successfully explored the design, implementation, and analysis of three different clock chain divider architectures: DFF-based, gated LFSR-based, and symbolic 8T SRAM-based. Each design was

developed in Verilog with integrated clock gating, power gating, and test mode to enable low-power operation and functional observability.

All three architectures were functionally verified using ModelSim and synthesized using Synopsys Design Compiler to evaluate area, power, and timing performance. Through comparative analysis, the SRAM-based divider demonstrated the most favorable trade-off, offering significantly lower dynamic power while maintaining acceptable area and timing characteristics.

The SRAM-based design was selected for further implementation and prepared for physical design using Cadence Innovus. This work highlights the benefits of combining memory-inspired logic with gating techniques for efficient clock division, and it sets the stage for future integration into low-power SoC designs such as RISC-V processors.

8. References

[1] https://www.efinixinc.com/products-riscv-en.html

[2] https://www.researchgate.net/figure/The-schematic-diagram- of-8T-SRAM-cell_fig3_283862501

[3] Ahmed, S., et al. "Clock Division and Randomization using LFSRs for Power Optimization," IEEE Access, vol. 8, pp. 204799–204808, 2020.

[4] Weste, Neil H.E., and David Harris. CMOS VLSI Design: A Circuits and Systems Perspective. Pearson.

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

[6] Cadence Innovus Implementation System User Guide, Cadence Design Systems.

[7] ModelSim User Manual, Mentor Graphics.

[8] Synopsys Design Vision Documentation, Synopsys Inc.

Appendix:

Top module for DFF Clock chain divider

```
module dff_clock_chain_divider (
   input clk,
   input rst,
   output [7:0] out
);

   reg [7:0] dc;

   assign out = dc;

   always @(posedge clk or posedge rst) begin
      if (rst)
         dc <= 8'b00000000;
      else
         dc <= {dc[6:0], ~dc[7]};
   end

endmodule
```

Testbench Module for DFF Clock chain divider

```
`timescale 1ns / 1ps
module tb_dff_clock_chain_divider;

   reg clk, rst;
   wire [7:0] out;
   reg [7:0] prev_out;

   dff_clock_chain_divider uut (
      .clk(clk),
      .rst(rst),
      .out(out)
   );

   initial clk = 0;
   always #5 clk = ~clk;

   initial begin
      $dumpfile("dff_divider.vcd");
      $dumpvars(0, tb_dff_clock_chain_divider);

      rst = 1;
      prev_out = 0;
```