8-Bit 8:1 Multiplexer: A Complete RTL to GDSII Implementation

# Introduction

A multiplexer (Mux) is a combinational digital circuit that selects one of several input signals and forwards the selected input into a single output line. A multiplexer is often referred to as a data selector. An 8:1 multiplexer, specifically, has eight data input lines (D0 to D7), three select lines (S2, S1, S0) to determine which input is routed to the output, and a single output line (Y).

This project implements an 8-bit 8:1 multiplexer. This means that for each of the eight data inputs, there are 8 bits, resulting in a total of 64 data input lines (D0[7:0] to D7[7:0]) and an 8-bit output (Y[7:0]). Additionally, the design incorporates an active-low RESET signal to asynchronously set the output to a known state (typically all zeros) and an ENABLE signal to control the overall operation of the multiplexer. When ENABLE is high, the multiplexer operates normally; when ENABLE is low, the output is typically held at a high-impedance state or a fixed low state, regardless of the select or data inputs.

## Operational Summary

|  |  |
| --- | --- |
| **Inputs** | **Description** |
| Data Inputs | D0[7:0] to D7[7:0]: Eight sets of 8-bit data inputs. |
| Select Lines | S2, S1, S0: Select which input is passed to the output. |
| ENABLE | Active-high signal. When low, output is disabled. |
| RESET | Active-low asynchronous reset that sets output to 0. |
| Output | Y[7:0]: The selected 8-bit data output. |

## Truth Table (Simplified for 1-bit)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| RESET | ENABLE | S2 | S1 | S0 | Output (Y) |
| 0 | X | X | X | X | 0 (Reset) |
| 1 | 0 | X | X | X | Z or 0 (Disabled) |
| 1 | 1 | 0 | 0 | 0 | D0 |
| 1 | 1 | 0 | 0 | 1 | D1 |
| 1 | 1 | 0 | 1 | 0 | D2 |
| 1 | 1 | 0 | 1 | 1 | D3 |
| 1 | 1 | 1 | 0 | 0 | D4 |
| 1 | 1 | 1 | 0 | 1 | D5 |
| 1 | 1 | 1 | 1 | 0 | D6 |
| RESET | ENABLE | S2 | S1 | S0 | Output (Y) |
| 0 | X | X | X | X | 0 (Reset) |
| 1 | 0 | X | X | X | Z or 0 (Disabled) |
| 1 | 1 | 0 | 0 | 0 | D0 |
| 1 | 1 | 0 | 0 | 1 | D1 |
| 1 | 1 | 0 | 1 | 0 | D2 |
| 1 | 1 | 0 | 1 | 1 | D3 |
| 1 | 1 | 1 | 0 | 0 | D4 |
| 1 | 1 | 1 | 0 | 1 | D5 |
| 1 | 1 | 1 | 1 | 0 | D6 |
| 1 | 1 | 1 | 1 | 1 | D7 |

# Applications

Multiplexers are fundamental components in digital design and have numerous applications, including:

* **Data Selection:** Routing multiple data streams through a single channel.
* **Data Routing:** Directing data from a single source to multiple destinations.
* **Parallel-to-Serial Conversion:** Converting parallel data into a serial stream.
* **Boolean Function Generation:** Implementing complex Boolean functions.
* **Memory Addressing:** Selecting specific memory locations.
* **Communication Systems:** Routing signals in telecommunication networks.
* **Test Equipment:** Selecting signals for measurement and analysis.

# Tools Used

## Design & Simulation

|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| Verilog HDL | Hardware Description Language used to design the RTL of the 8:1 Mux |
| Synopsys VCS | Compiles and simulates Verilog code at RTL level |
| Verdi | Waveform viewer for debugging simulation results and post-layout analysis |

## Synthesis

|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| Synopsys Design Compiler (dc\_shell / dc\_shell-xg-t) | Synthesizes RTL to gate-level netlist |
| Standard Design Cell Library (e.g., SAED32/45nm) | Target technology library for synthesis |

## Physical Design

|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| Synopsys ICC2 (icc2\_shell) | Performs floorplanning, placement, CTS, routing, and GDSII generation |
| start\_gui | Graphical interface for ICC2 and Design Compiler |

## Timing and Power Analysis

|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| Synopsys PrimeTime (pt\_shell) | Static timing and power analysis after synthesis and layout |

## Environment

|  |  |
| --- | --- |
| **Tool** | **Purpose** |
| Rocky Linux | Operating system environment used for running Synopsys tools |

# Project Structure

The project directory is organized as follows:

RTL2GDSII\_/

├── 1\_RTL/

│ ├── 8x1mux.v

│ └── testbench.v

├── 2\_Synthesis/

│ ├── 8x1mux\_synth.v

│ └── synthesis\_report.txt

├── 3\_Floorplan/

│ ├── floorplan.tcl

│ └── floorplan\_report.txt

├── 4\_PowerPlan/

│ ├── power\_plan.tcl

│ └── power\_plan\_report.txt

├── 5\_Place/

│ ├── placement.tcl

│ └── placement\_report.txt

├── 6\_CTS/

│ ├── cts.cl

│ └── cts\_report.txt

├── 7\_Routing/

│ ├── routing.tcl

│ └── routing\_report.txt

Below are the major steps followed using Synopsys tools:

1. RTL Simulation using VCS and waveform viewing in Verdi

In this step, the Register Transfer Level (RTL) design is validated by running simulations. Synopsys VCS compiles the Verilog code, and Verdi is used to visualize the simulation waveform which helps in debugging the logical behavior and timing of the design.

RTL code is translated to a gate-level representation using Synopsys Design Compiler. This involves logic optimization, mapping to standard cells, and applying design constraints such as timing and area.

Physical implementation includes several sequential stages:  
- Floorplanning: Define chip size and block locations.  
- Power Planning: Design the power distribution network.  
- Placement: Allocate physical locations for logic cells.  
- Clock Tree Synthesis (CTS): Build the clock distribution network with minimal skew.  
- Routing: Establish metal interconnections for the placed cells.

Static Timing Analysis (STA) checks for setup and hold violations without requiring test vectors. Power analysis calculates dynamic and static power consumption. PrimeTime is used after layout to ensure the design meets timing and power budgets.

1. Synthesis using Design Compiler
2. Physical Design in ICC2 (Floorplan, Powerplan, Placement, CTS, Routing)
3. Static Timing Analysis and Power using PrimeTime

# CODE & SNAPSHOTS

**Create Verilog Files**

**File: mux\_rtl.v**

module mux (

input [7:0] I0, I1, I2, I3, I4, I5, I6, I7,

input [2:0] Sel,

input Clock,

input Reset, // Reset signal

input Enable, // Enable signal

output reg [7:0] Y;

always @ (posedge Clock or posedge Reset) begin

if (Reset) begin

Y <= 8'b0; // Reset output to 0

end else if (Enable) begin

case (Sel)

3'b000: Y <= I0;

3'b001: Y <= I1;

3'b010: Y <= I2;

3'b011: Y <= I3;

3'b100: Y <= I4;

3'b101: Y <= I5;

3'b110: Y <= I6;

3'b111: Y <= I7;

default: Y <= 8'b0;

endcase

end

end

endmodule

**File: mux\_tb.v**

`timescale 1ns/1ns

`include "mux\_rtl.v" // includes the module definition for the 8-to-1 mux

module testbench;

reg [7:0] I0, I1, I2, I3, I4, I5, I6, I7; // 8-bit inputs

reg [2:0] Sel; // 3-bit select input

reg Clock; // Clock input

reg Reset; // Reset input

reg Enable; // Enable input

wire [7:0] Y; // 8-bit output

// Instantiate the module under test

mux dut (.I0(I0), .I1(I1), .I2(I2), .I3(I3), .I4(I4), .I5(I5), .I6(I6), .I7(I7), .Sel(Sel), .Clock(Clock), .Reset(Reset), .Enable(Enable), .Y(Y));

// Clock generation

always #5 Clock = ~Clock; // Clock signal with a period of 10 ns

initial begin

$fsdbDumpvars(); // Tool specific command for waveform generation

// Initialize inputs

I0 = 8'b00000000; I1 = 8'b00000001; I2 = 8'b00000010; I3 = 8'b00000011;

I4 = 8'b00000100; I5 = 8'b00000101; I6 = 8'b00000110; I7 = 8'b00000111;

Sel = 3'b000; // Start with the first input

Clock = 0; // Initialize clock

Reset = 1; // Start with reset

Enable = 0; // Disable initially

#10 Reset = 0; // Release reset

Enable = 1; // Enable the mux

// Apply test cases

#20 Sel = 3'b000; // Select I0

#10; // Wait for clock edge

$display("Sel = %b, Y = %b", Sel, Y);

#20 Sel = 3'b001; // Select I1

#10; // Wait for clock edge

$display("Sel = %b, Y = %b", Sel, Y);

#20 Sel = 3'b010; // Select I2

#10; // Wait for clock edge

$display("Sel = %b, Y = %b", Sel, Y);

#20 Sel = 3'b011; // Select I3

#10; // Wait for clock edge

$display("Sel = %b, Y = %b", Sel, Y);

#20 Sel = 3'b100; // Select I4

#10; // Wait for clock edge

$display("Sel = %b, Y = %b", Sel, Y);

#20 Sel = 3'b101; // Select I5

#10; // Wait for clock edge

$display("Sel = %b, Y = %b", Sel, Y);

#20 Sel = 3'b110; // Select I6

#10; // Wait for clock edge

$display("Sel = %b, Y = %b", Sel, Y);

#20 Sel = 3'b111; // Select I7

#10; // Wait for clock edge

$display("Sel = %b, Y = %b", Sel, Y);

#100 $finish; // End simulation

end

endmodule

A computer screen shot of a computer program

AI-generated content may be incorrect.**SnapShot of waveform and logic**A screenshot of a computer

AI-generated content may be incorrect.A computer screen shot of a computer program

AI-generated content may be incorrect.

# DC SHELL

source -echo -verbose ./rm\_setup/dc\_setup.tcl

set RTL\_SOURCE\_FILES ./../rtl/mux\_rtl.v

define\_design\_lib WORK -path ./WORK

set\_dont\_use [get\_lib\_cells \*/FADD\*]

set\_dont\_use [get\_lib\_cells \*/HADD\*]

set\_dont\_use [get\_lib\_cells \*/AO\*]

set\_dont\_use [get\_lib\_cells \*/OA\*]

#set\_dont\_use [get\_lib\_cells \*/NAND\*]

#set\_dont\_use [get\_lib\_cells \*/AND\*]

set\_dont\_use [get\_lib\_cells \*/XOR\*]

#set\_dont\_use [get\_lib\_cells \*/OR\*]

set\_dont\_use [get\_lib\_cells \*/NOR\*]

set\_dont\_use [get\_lib\_cells \*/XNOR\*]

set\_dont\_use [get\_lib\_cells \*/MUX\*]

analyze -format verilog ${RTL\_SOURCE\_FILES}

elaborate ${DESIGN\_NAME}

current\_design

read\_sdc ./../CONSTRAINTS/mux.sdc

#compile

compile\_ultra

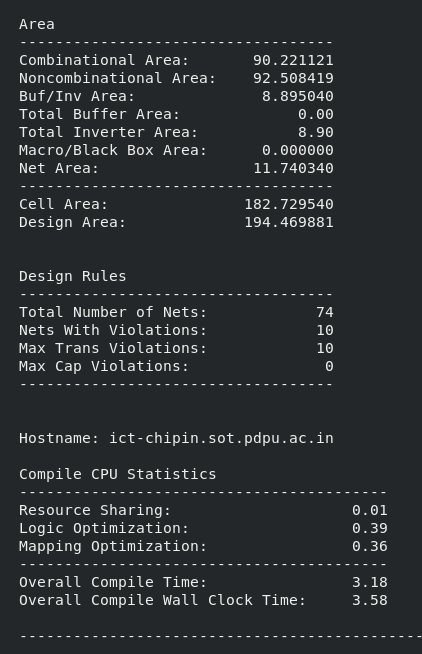
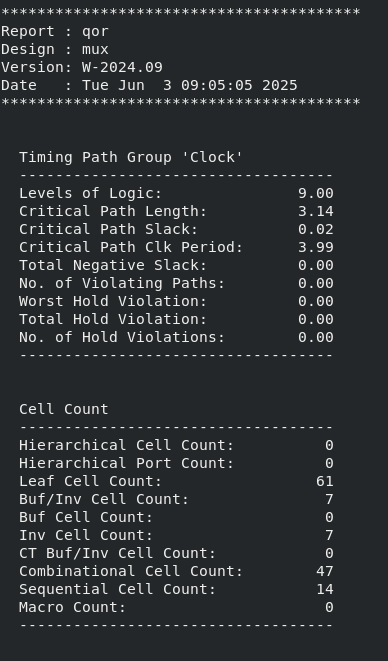
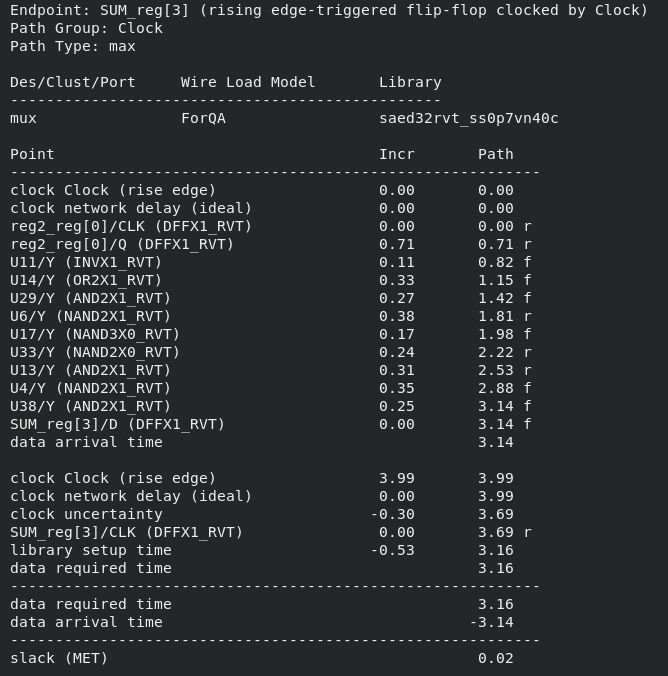
#report\_timing

write -format verilog -hierarchy -output ${RESULTS\_DIR}/${DCRM\_FINAL\_VERILOG\_OUTPUT\_FILE}

**Snapshot of Schematic of DC shell , report\_qor & report\_timing**

**A diagram of a circuit

AI-generated content may be incorrect.**



# 

# ICC2 SHELL

Floorplan

set PDK\_PATH ./../ref

create\_lib -ref\_lib $PDK\_PATH/lib/ndm/saed32rvt\_c.ndm MUX\_LIB

read\_verilog {./../DC/results/mux.mapped.v} -library MUX\_LIB -design mux -top mux  
  
open\_lib MUX\_LIB

open\_block MUX

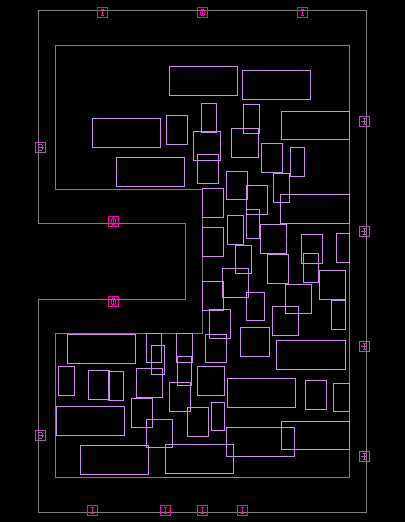
initialize\_floorplan -core\_utilization 0.5 -coincident\_boundary true -core\_offset {1 2} -shape U -orientation W

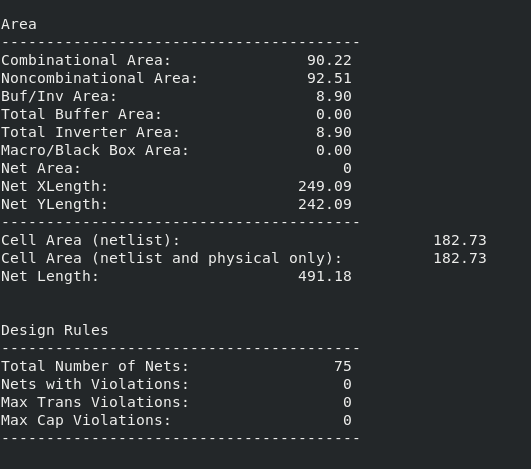
place\_pins -self

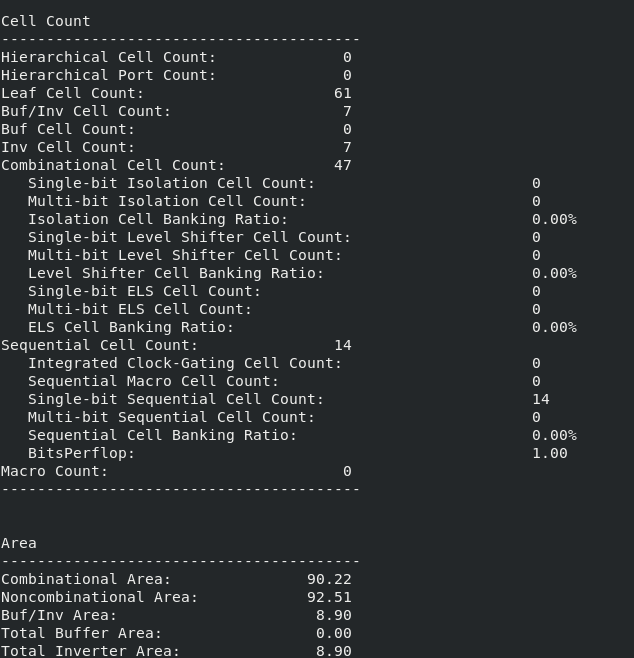
create\_placement -floorplan

save\_block -as MUX\_

save\_lib







PowerPlan

### Step 1: to\_create power/ground nets and to\_connect power/ground nets :-

#to create power nets

create\_net -power {VDD}

create\_net -ground {VSS}

## to connect power/ground\_nets

connect\_pg\_net -all\_blocks -automatic

### step 2: to create power and ground ring patterns

#scenario1:

create\_pg\_ring\_pattern core\_ring\_pattern -horizontal\_layer M7 -horizontal\_width .4 -horizontal\_spacing .3 -vertical\_layer M8 -vertical\_width .4 -vertical\_spacing .3

set\_pg\_strategy core\_power\_ring -core -pattern {{name : core\_ring\_pattern}{nets : {VDD VSS}}{offset : {.5 .5}}}

compile\_pg -strategies core\_power\_ring

### step 3: to create pg mesh pattern

create\_pg\_mesh\_pattern mesh -layers { {{vertical\_layer: M6}{width: .34} {spacing: interleaving}{pitch: 5} {offset: .5}} {{horizontal\_layer: M7}{width: .38} {spacing: interleaving} {pitch: 5} {offset: .5}} {{vertical\_layer: M8}{width: .38} {spacing: interleaving}{pitch: 5} {offset: .5}}}

set\_pg\_strategy core\_mesh -pattern { {pattern:mesh} {nets: VDD VSS}} -core -extension {stop: innermost\_ring}

#set\_pg\_strategy core\_mesh -pattern { {pattern:mesh} {nets: VDD VSS}} -core -extension {{{side: 1 2} {direction: L T} {stop: innermost\_ring}}}

compile\_pg -strategies core\_mesh

###step 4 : to create std cell power rail pattern

create\_pg\_std\_cell\_conn\_pattern std\_cell\_rail -layers {M1} -rail\_width 0.06

set\_pg\_strategy rail\_strat -core -pattern {{name: std\_cell\_rail} {nets: VDD VSS} }

compile\_pg -strategies rail\_strat

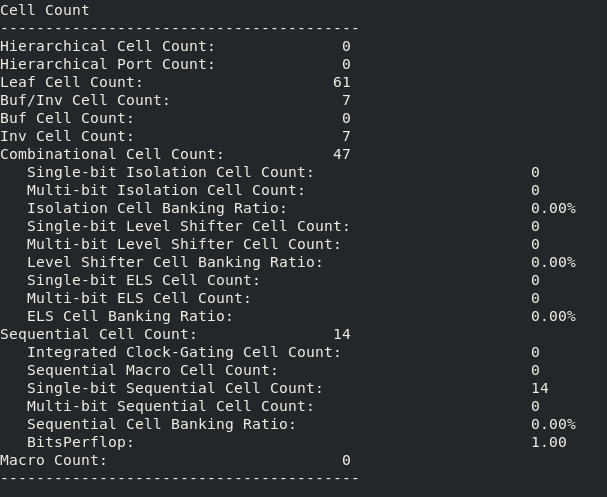
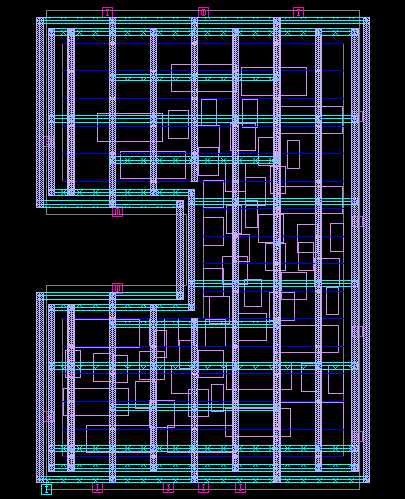
connect\_pg\_net -all\_blocks -automatic

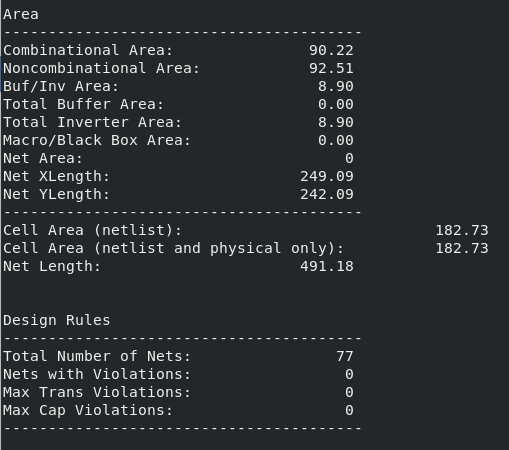
#compile\_pg

### step 5 : To save block and Save lib

save\_block

save\_lib





Placement

set mode1 "func"

set corner1 "nom"

set scenario1 "${mode1}::${corner1}"

remove\_modes -all; remove\_corners -all; remove\_scenarios -all

create\_mode $mode1

create\_corner $corner1

create\_scenario -name func::nom -mode func -corner nom

current\_mode func

current\_scenario func::nom

source ./../CONSTRAINTS/mux.sdc

set\_dont\_use [get\_lib\_cells \*/FADD\*]

set\_dont\_use [get\_lib\_cells \*/HADD\*]

set\_dont\_use [get\_lib\_cells \*/AO\*]

set\_dont\_use [get\_lib\_cells \*/OA\*]

#set\_dont\_use [get\_lib\_cells \*/NAND\*]

#set\_dont\_use [get\_lib\_cells \*/AND\*]

set\_dont\_use [get\_lib\_cells \*/XOR\*]

#set\_dont\_use [get\_lib\_cells \*/OR\*]

set\_dont\_use [get\_lib\_cells \*/NOR\*]

set\_dont\_use [get\_lib\_cells \*/XNOR\*]

set\_dont\_use [get\_lib\_cells \*/MUX\*]

current\_corner nom

current\_scenario func::nom

set parasitic1 "p1"

set tluplus\_file$parasitic1 "$PDK\_PATH/tech/star\_rcxt/saed32nm\_1p9m\_Cmax.tluplus"

set layer\_map\_file$parasitic1 "$PDK\_PATH/tech/star\_rcxt/saed32nm\_tf\_itf\_tluplus.map"

set parasitic2 "p2"

set tluplus\_file$parasitic2 "$PDK\_PATH/tech/star\_rcxt/saed32nm\_1p9m\_Cmin.tluplus"

set layer\_map\_file$parasitic2 "$PDK\_PATH/tech/star\_rcxt/saed32nm\_tf\_itf\_tluplus.map"

read\_parasitic\_tech -tlup $tluplus\_filep1 -layermap $layer\_map\_filep1 -name p1

read\_parasitic\_tech -tlup $tluplus\_filep2 -layermap $layer\_map\_filep2 -name p2

set\_parasitic\_parameters -late\_spec $parasitic1 -early\_spec $parasitic2

set\_app\_options -name place.coarse.continue\_on\_missing\_scandef -value true

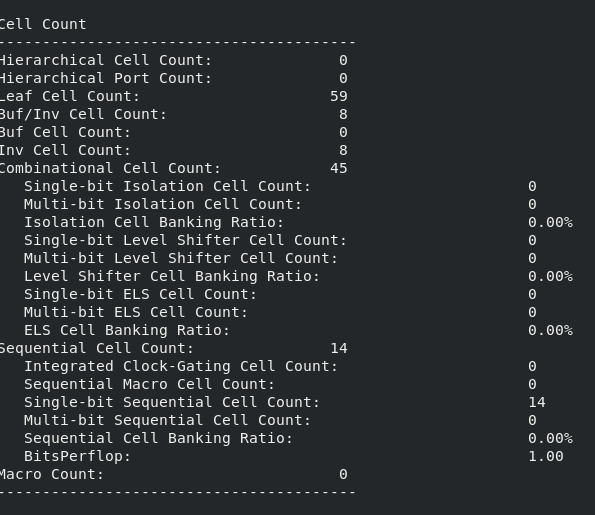
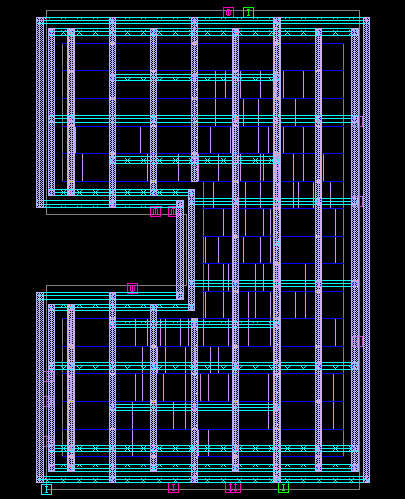
place\_pins -self

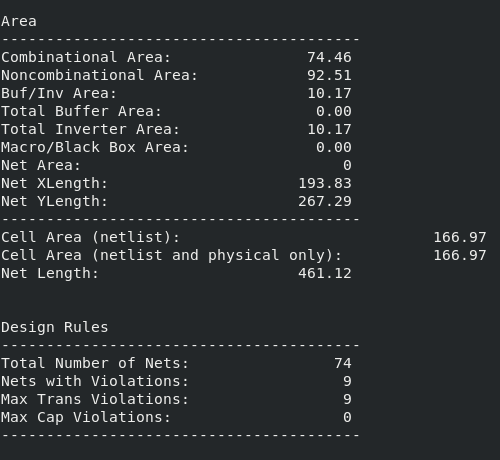
place\_opt

legalize\_placement

save\_block -as mux\_placement

save\_lib





CTS ( Clock Tree Synthesis )

check\_design -checks pre\_clock\_tree\_stage

##stage 1:

synthesize\_clock\_tree

##stage 2:

set\_app\_options -name cts.optimize.enable\_local\_skew -value true

set\_app\_options -name cts.compile.enable\_local\_skew -value true

set\_app\_options -name cts.compile.enable\_global\_route -value false

set\_app\_options -name clock\_opt.flow.enable\_ccd -value true

clock\_opt -to build\_clock

#stage 3:

clock\_opt -from route\_clock -to route\_clock

clock\_opt

#report\_clock\_qor

#report\_clock\_qor -largest 2 -show\_verbose\_paths

#report\_clock\_qor -largest 2 -show\_verbose\_paths > cts\_ccd.rpt

save\_block -as mux\_cts\_CCD

save\_lib

# set app options

set\_app\_options -name route.global.timing\_driven -value true

set\_app\_options -name route.global.crosstalk\_driven -value false

#track assignment

set\_app\_options -name route.track.timing\_driven -value true

set\_app\_options -name route.track.crosstalk\_driven -value true

#detail route

set\_app\_options -name route.detail.timing\_driven -value true

set\_app\_options -name route.detail.save\_after\_iterations -value false

set\_app\_options -name route.detail.force\_max\_number\_iterations -value false

set\_app\_options -name route.detail.antenna -value true

set\_app\_options -name route.detail.antenna\_fixing\_preference -value use\_diodes

set\_app\_options -name route.detail.diode\_libcell\_names -value \*/ANTENNA\_RVT

route\_global

#save\_block route\_global\_database

route\_track

#save\_block route\_track\_database

route\_detail

#save\_block route\_detail\_database

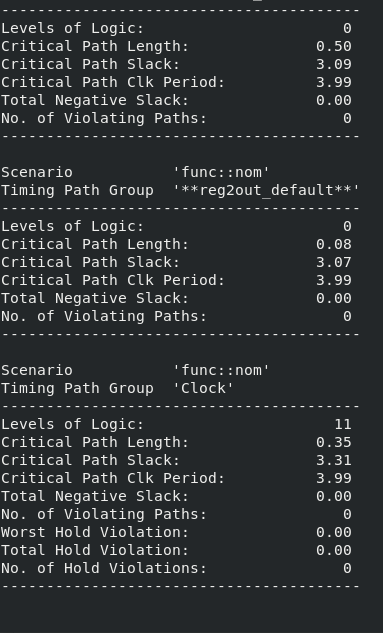
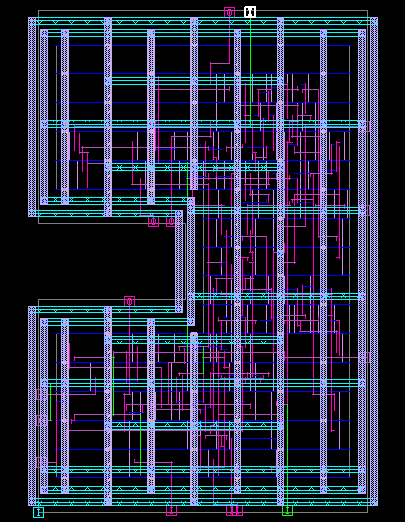
#route\_auto

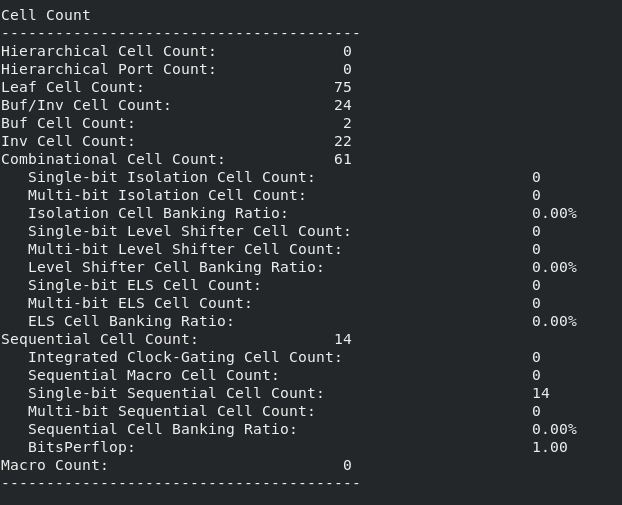
route\_opt

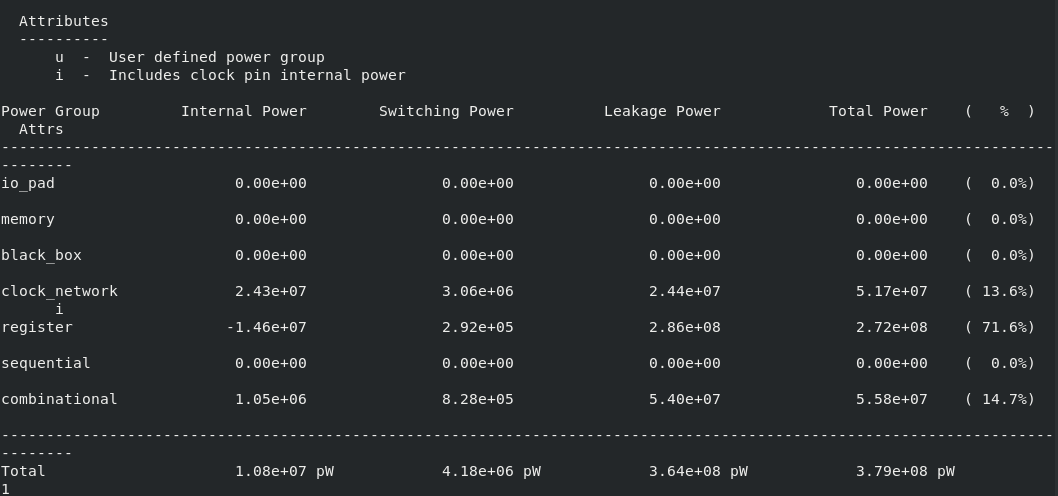
write\_verilog ./results/mux.routed.v

write\_sdc -output ./results/mux.routed.sdc

write\_parasitics -format spef -output ./results/mux\_${scenario1}.spef

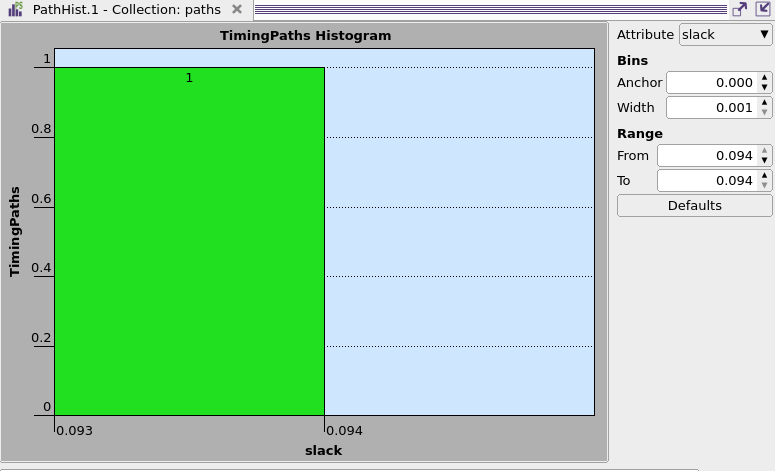






# PT SHELL

The pt sleck time cam out to be 0.094 as it is shown in the graph below .



# Final Summary Conclusion: A Journey from RTL to GDSII – Learnings and Reflections

This project provided an invaluable, end-to-end experience in digital design—starting from the abstract world of RTL coding to the tangible physical layout ready for fabrication (GDSII). Implementing an 8-bit 8:1 multiplexer with RESET and ENABLE functionality was more than just a design task—it was an educational journey that deepened my understanding of both the theoretical and practical aspects of VLSI design.

At the RTL design stage, I strengthened my command over Verilog HDL, learning how to model functional behavior of digital circuits accurately and modularly. Writing a well-structured testbench and observing waveform outputs using VCS and Verdi taught me how simulation bridges the gap between code and logic.

During synthesis with Design Compiler, I realized the importance of optimization and constraint-driven development. The way high-level RTL code is translated into a gate-level netlist helped me appreciate the importance of timing, area, and power in real-world design.

Entering the physical design phase using ICC2, I learned about chip planning at the silicon level. Concepts like floorplanning, power distribution networks, standard cell placement, clock tree synthesis (CTS), and routing were no longer abstract terms—they became real challenges to address and resolve using specific commands, flows, and strategies.

The power planning steps made me more conscious of how crucial power integrity is, especially in deep-submicron technologies. Meanwhile, the placement and routing phases taught me about physical constraints and congestion handling. Clock Tree Synthesis particularly stood out as a critical balance between skew, latency, and power—a concept I had only briefly understood before this project.

Using PrimeTime for timing and power analysis, I saw firsthand how static analysis tools verify whether the design will function reliably across different operating conditions. Understanding concepts like slack, parasitic extraction, and path delays was key in realizing the real challenges of closing timing.

In short, this project taught me how various stages of the VLSI flow are intricately connected. Each step builds on the last, and mistakes or oversights in early phases often propagate downstream. This instilled in me a sense of discipline and carefulness while designing and scripting at each stage.

Most importantly, I gained practical exposure to industry-standard EDA tools from Synopsys—VCS, Design Compiler, ICC2, and PrimeTime—which are essential in modern semiconductor design workflows. I now feel more confident and prepared to handle larger and more complex ASIC design projects in the future

# References

- https://www.allaboutcircuits.com/  
- https://www.vlsiexpert.com/  
- https://www.chipgrad.com/  
- SAED 32nm PDK Documentation – Synopsys University Program  
- Synopsys Tool User Guides  
- VLSI Lab Tutorials by Mr. Puneet Mittal

# Credits

This project was completed as part of the RTL to GDSII Flow Lab at Pandit Deendayal Energy University (PDEU).

We express our sincere gratitude to:  
- Mr. Puneet Mittal, VLSI Expert, for his exceptional guidance  
- Faculty members of PDEU for their support and instruction