Digital Design Project Report

ECE 212s

Team 19

Name	ID
Mohamed Abdou Ahmed	2200206
Yousef Abd El-Fattah Mahmoud	2200260
Ahmed Mahmoud Fathy	2200285
Marwan Abdallah Amin	2200560
Moaaz Tarek Mohamed	2200092
Mohamed Mahmoud Ibrahim	2200456
Omar Ahmed Gamal	2200196
Shihab Mohamed	2201366
Karim Hosam Ahmed	2201405

Table of contributions

Implementation of base fundamental gates	Ahmed Mahmoud-Youssef Abdelfattah Mohamed Abdou- Mohamed Mahmoud
Logical operations modules	Mohamed Abdou- Omar Ahmed
Arithmetic operations modules	Marwan Abdallah- Mohamed Mahmoud - Yousef Abdelfattah
Delay Calculations	Shihab Mohamed- Moaaz Tarek Mohamed Mahmoud
Power Calculations	Karim Hosam - Marwan Abdallah
ALU Assembly	Mohamed Abdou- Marwan Abdallah- Ahmed Mahmoud
Test-Benches	Ahmed Mahmoud- Yousef Abdelfattah
Verilog	Yousef Abdelfattah Mahmoud
Implementation of sequential circuit	Moaaz Tarek- Ahmed Mahmoud- Marwan Abdallah
Report	Karim Hosam - Shihab Mohamed- Omar Ahmed - Mohamed Abdou- Moaaz Tarek

Undersupervision of ENG. Amr Hamouda

Introduction

This project report documents the design and implementation of a custom Arithmetic and Logical Unit (ALU). The primary objective of the project was to design an efficient ALU capable of performing a variety of arithmetic and logical operations using both HDL and transistor-level design.

Project Description

The ALU designed in this project supports both arithmetic and logical operations based on a 4-bit control signal (sel<3:0>), with the most significant bit determining the operation category (arithmetic or logic), and the remaining bits selecting the specific operation. The unit operates on two signed 4-bit inputs (a<3:0> and b<3:0>) and outputs an 8-bit result.

Implemented Features:

1. Arithmetic Operations: Increment, Decrement, Transfer, Add, Subtract, multiply.

2. Logical Operations: AND, OR ,XOR, XNOR, NAND, NOR, and 1's Complement.

3. VHDL/Verilog Code: Complete HDL implementation of the ALU and flip-flops.

4. Testbench: Comprehensive verification covering all operations.

5. Transistor-Level Design: Schematic and simulation using Cadence tools with 130nm CMOS technology.

6. Performance Metrics: Delay and power consumption measured at worst-case and maximum-frequency conditions.

7. Flip-Flops: Designed at both input and output stages for synchronization.

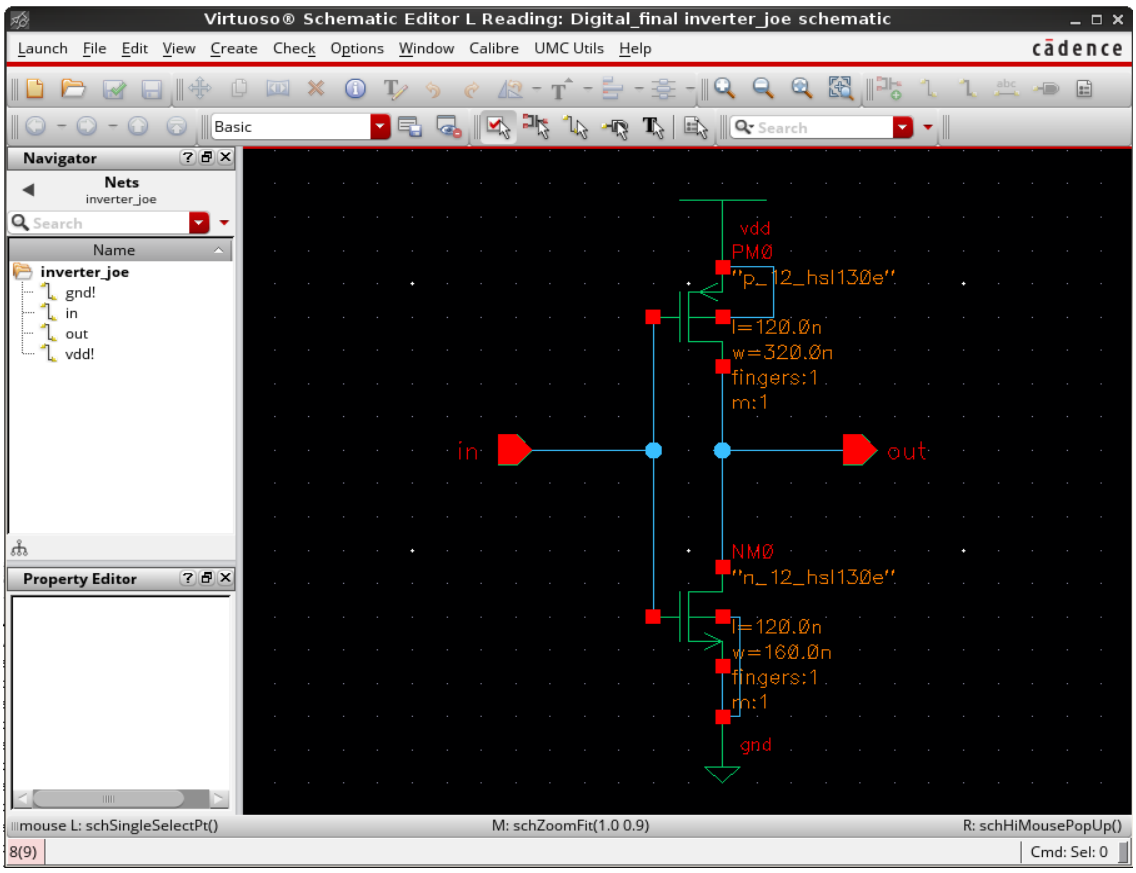
8. Simulation: Transient analysis confirming correct sampling at maximum operational frequency.

Links:

CADENCE Files:

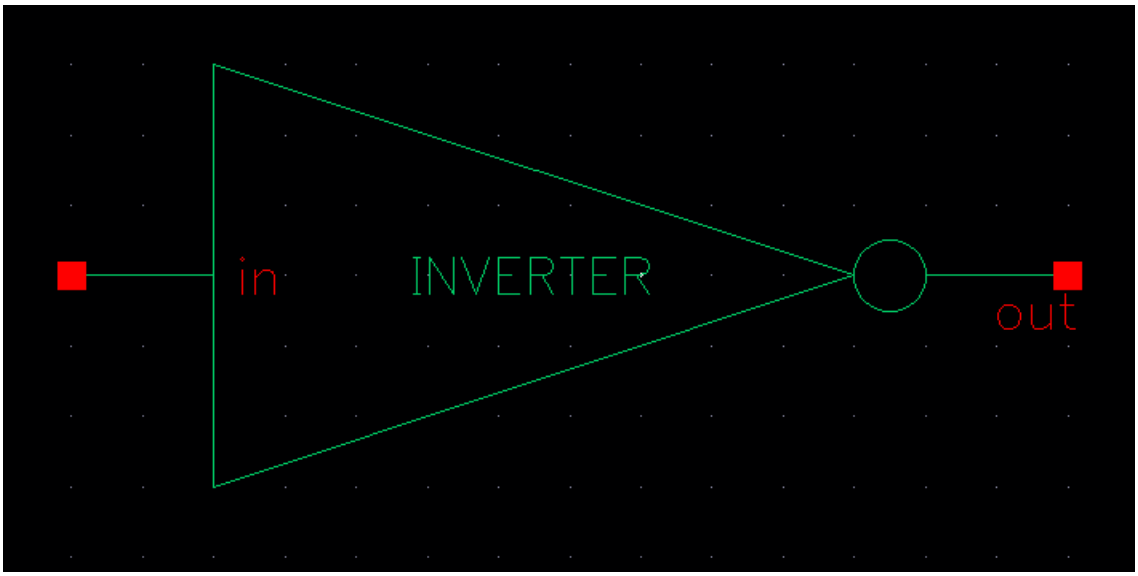
https://drive.google.com/drive/u/0/folders/1yvG5f3DQXBb9tfuMXrkIo69znCAbhpda

Inverter

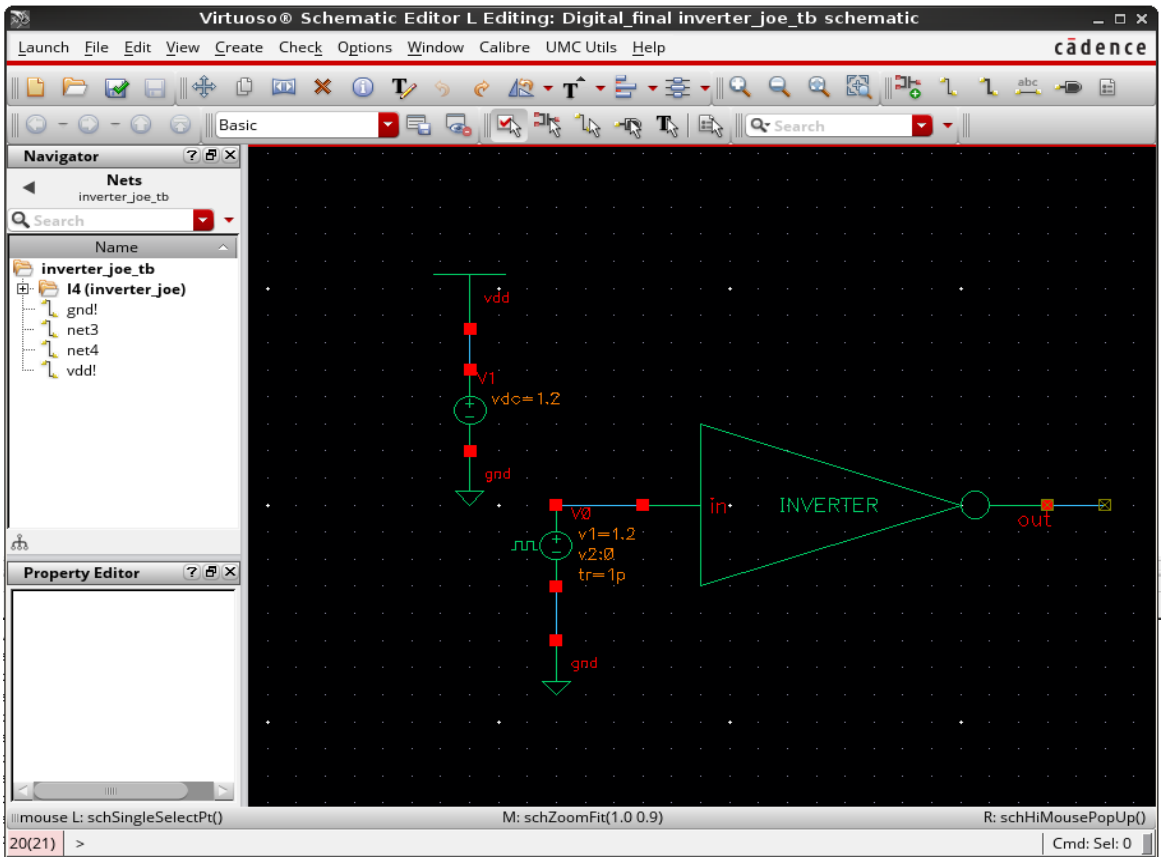


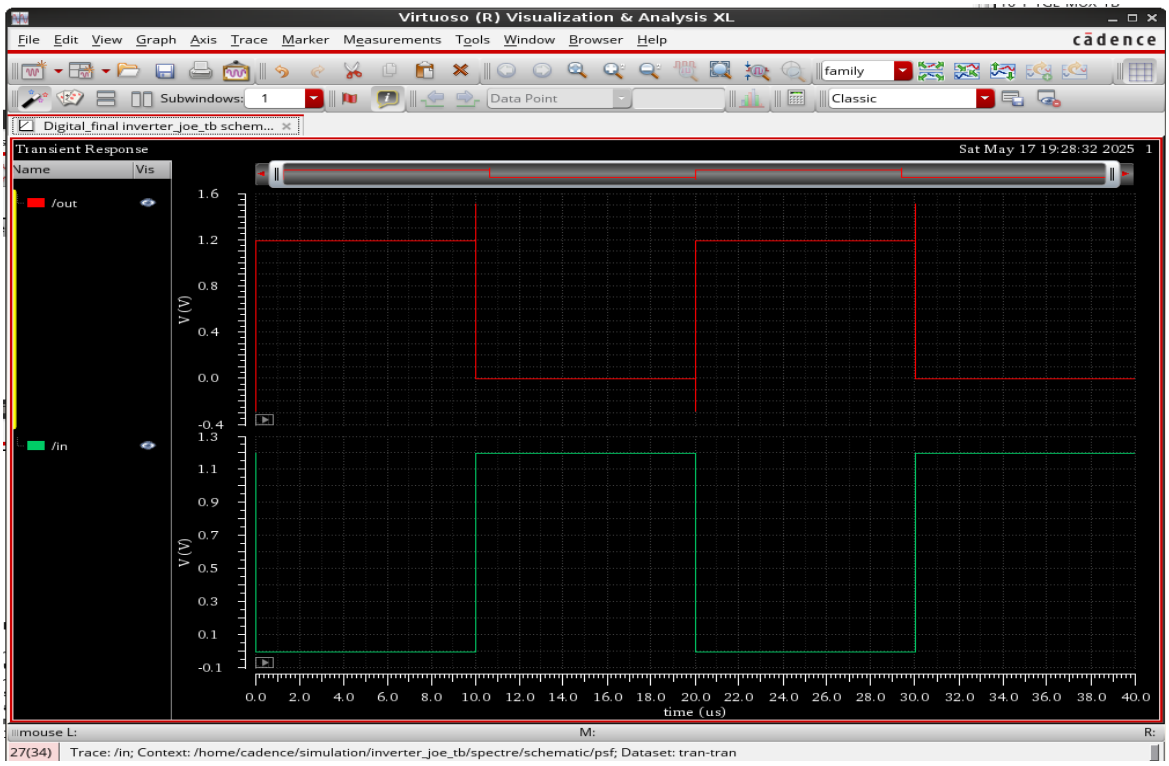
We sized the inverter to be symmetric inverter as we used the ratio Wp=2Wn

Symbol



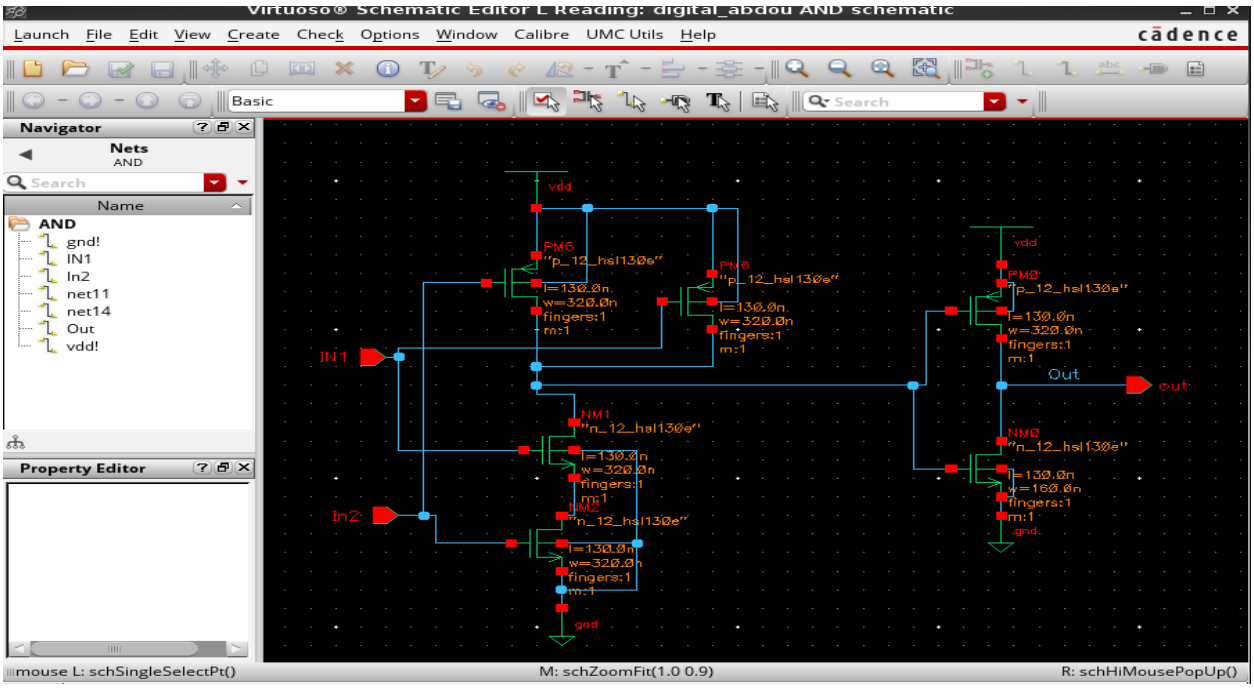
Inverter testbench





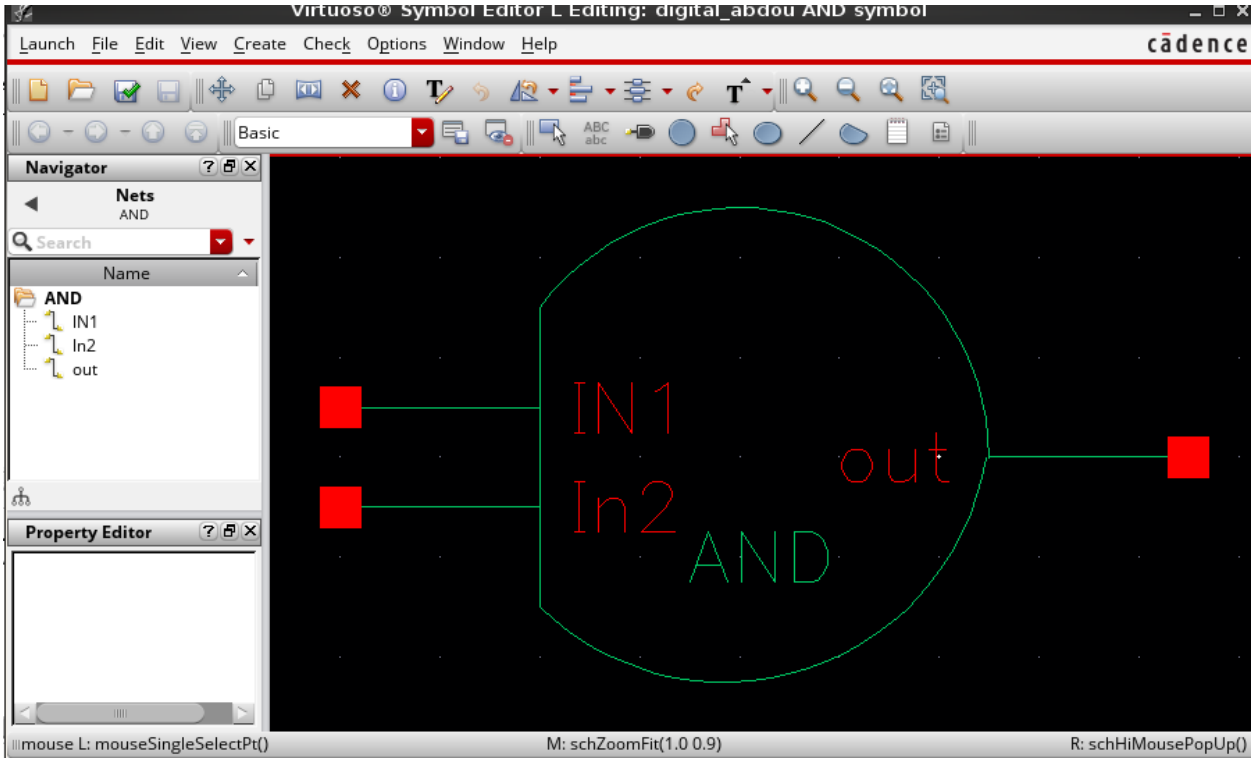
AND

schematic

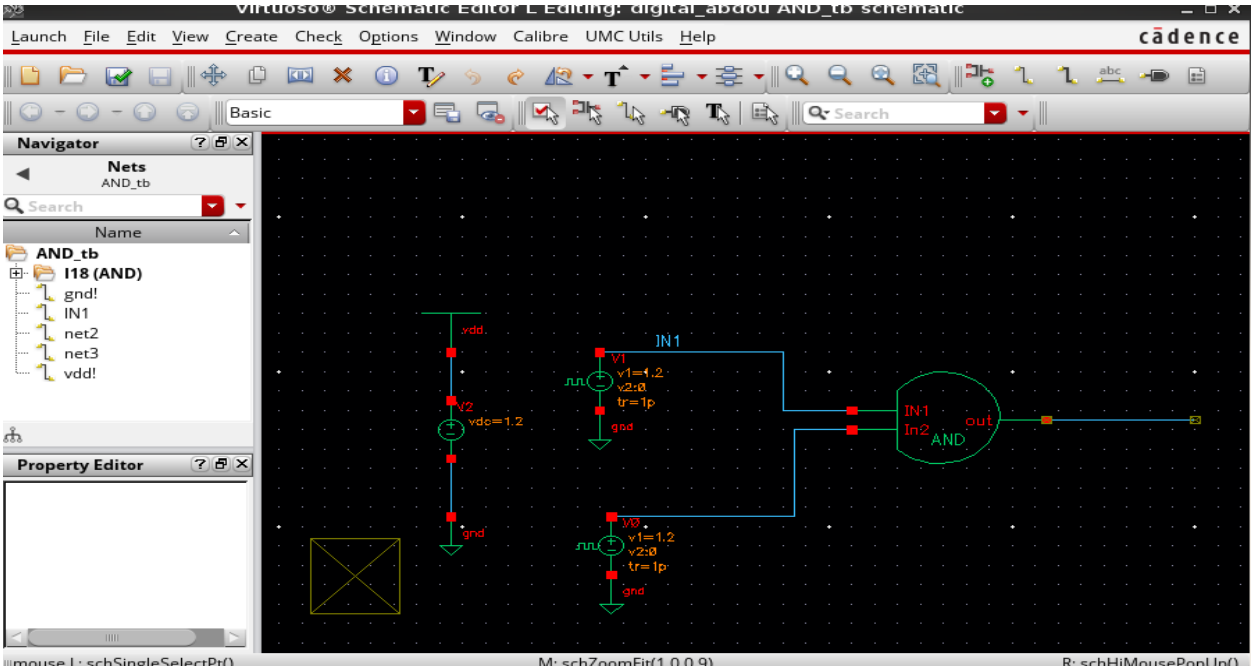


we sized each gate according to the reference inverter we made at first

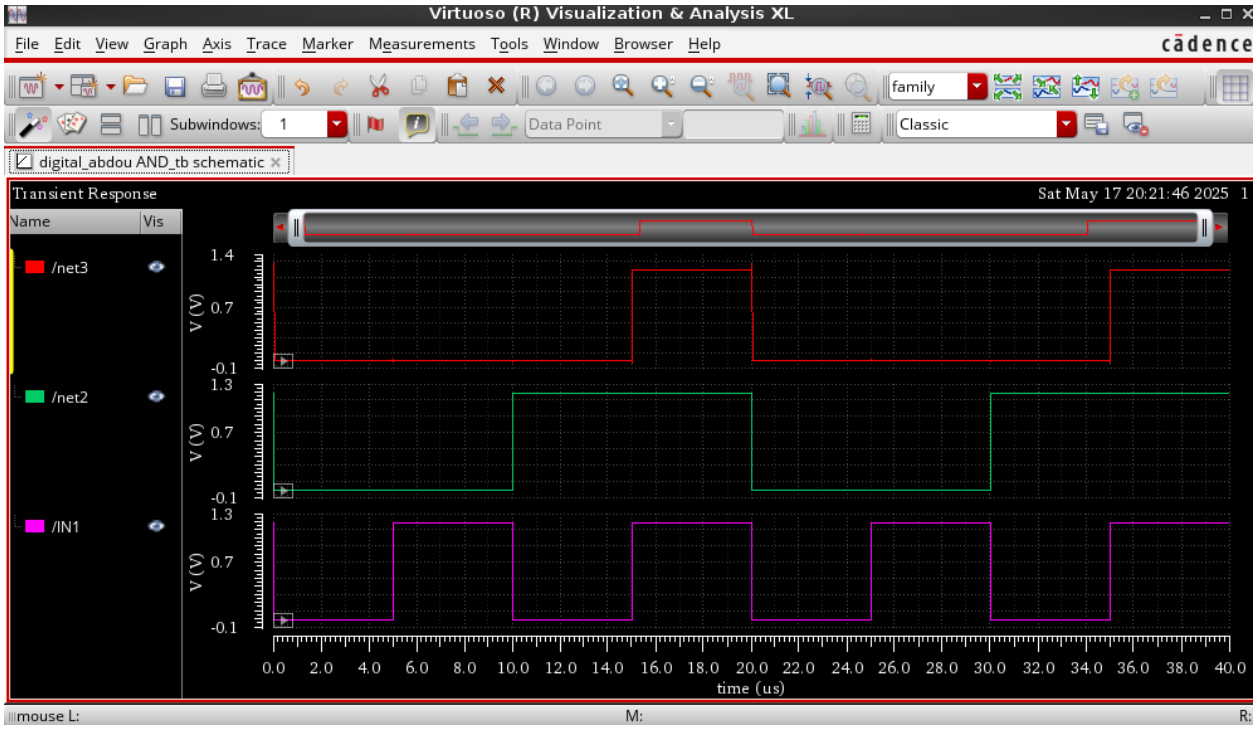
symbol



Test Bench

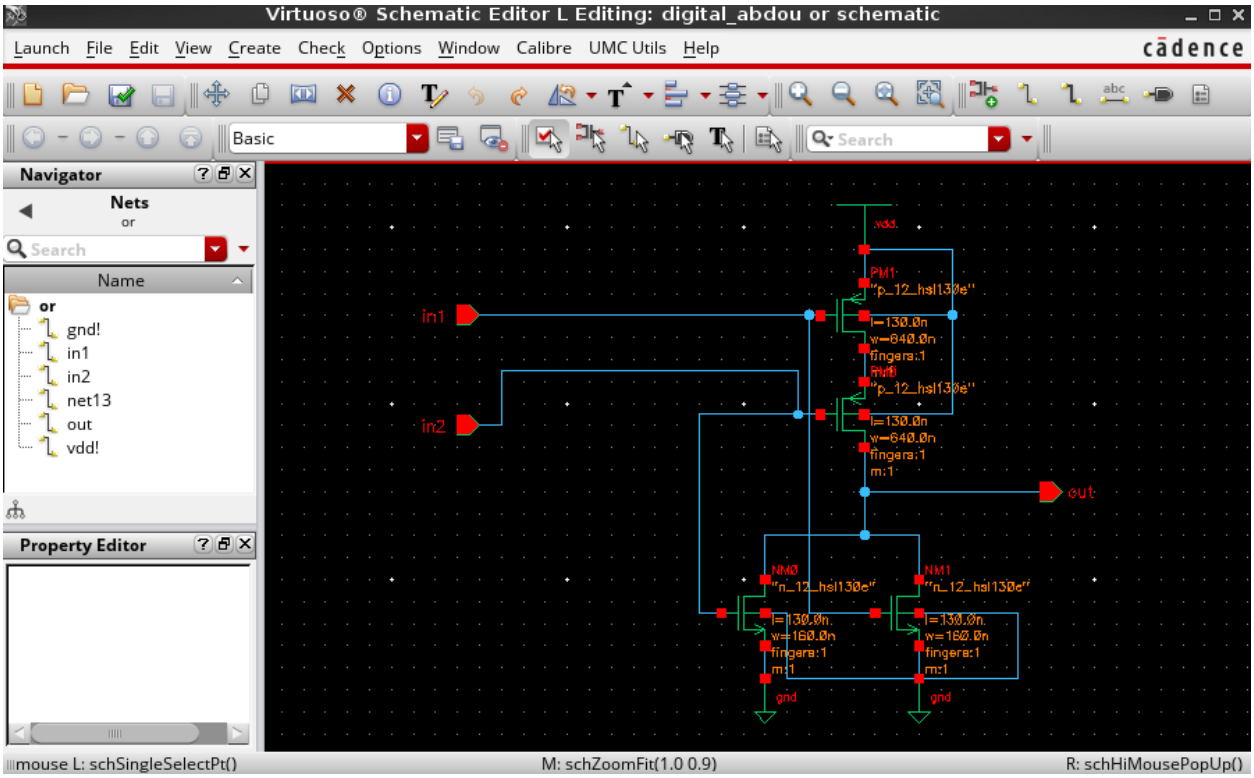


Wave Form

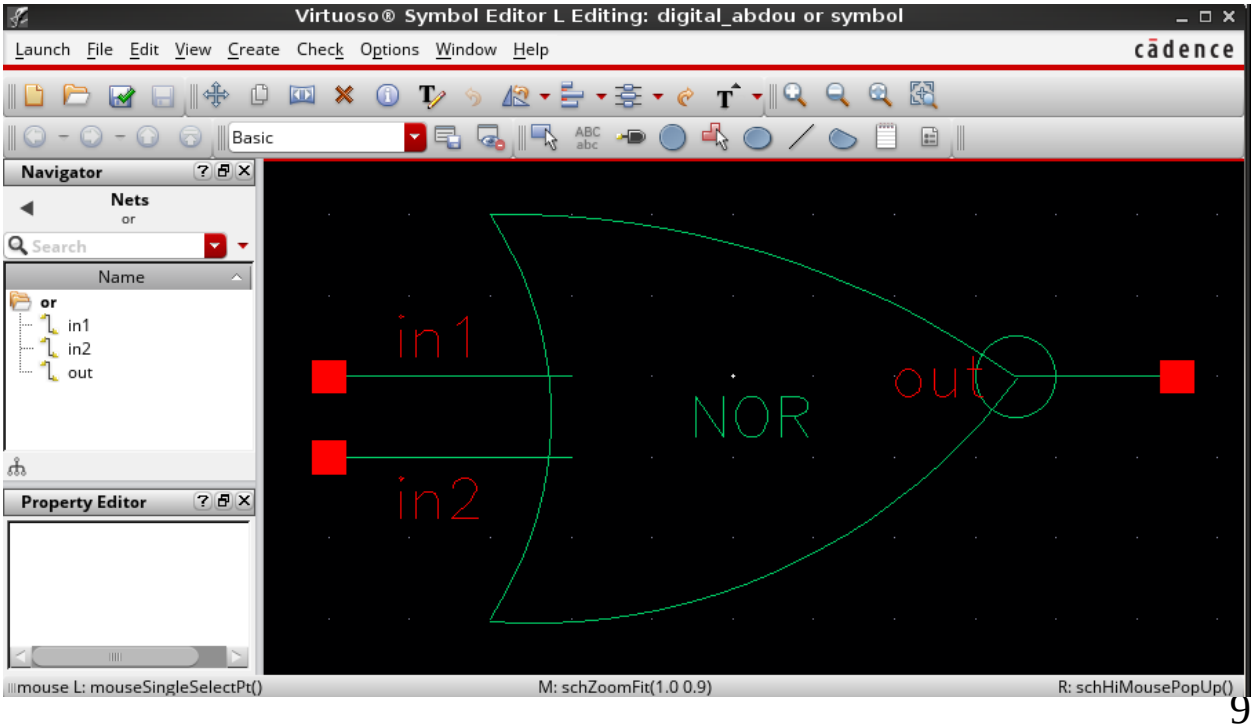


NOR

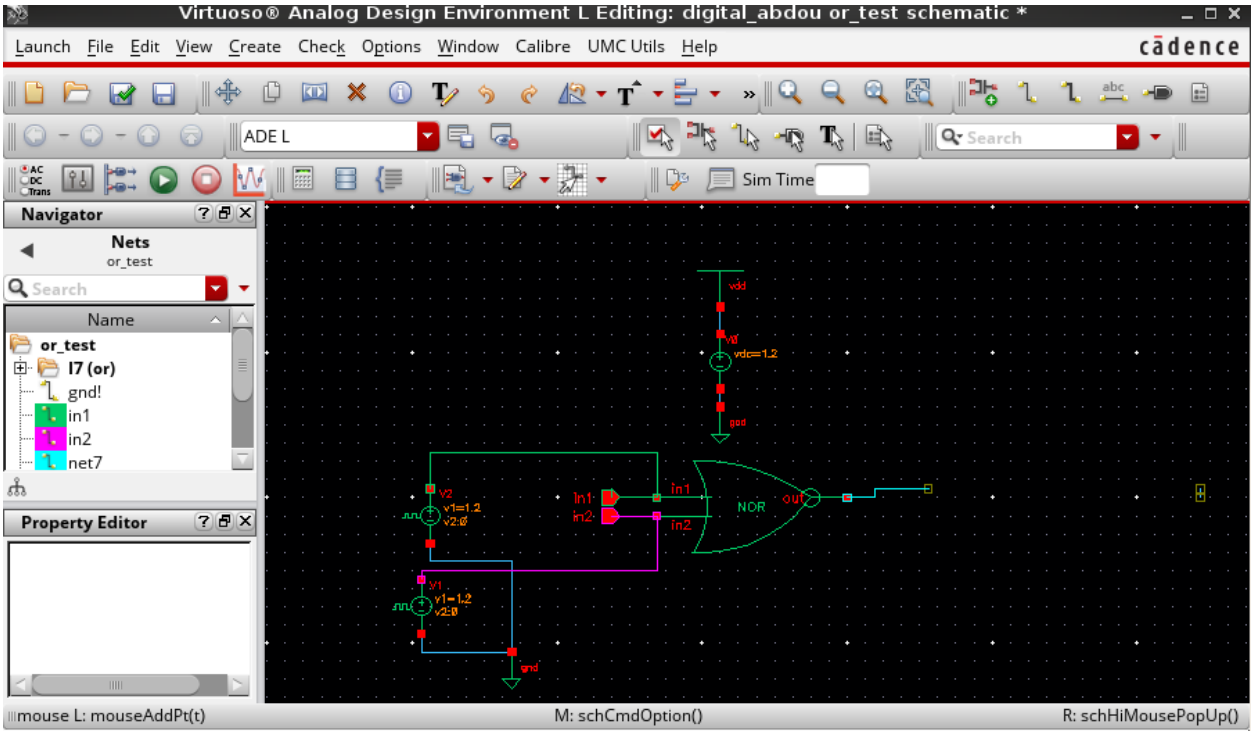
Schematic



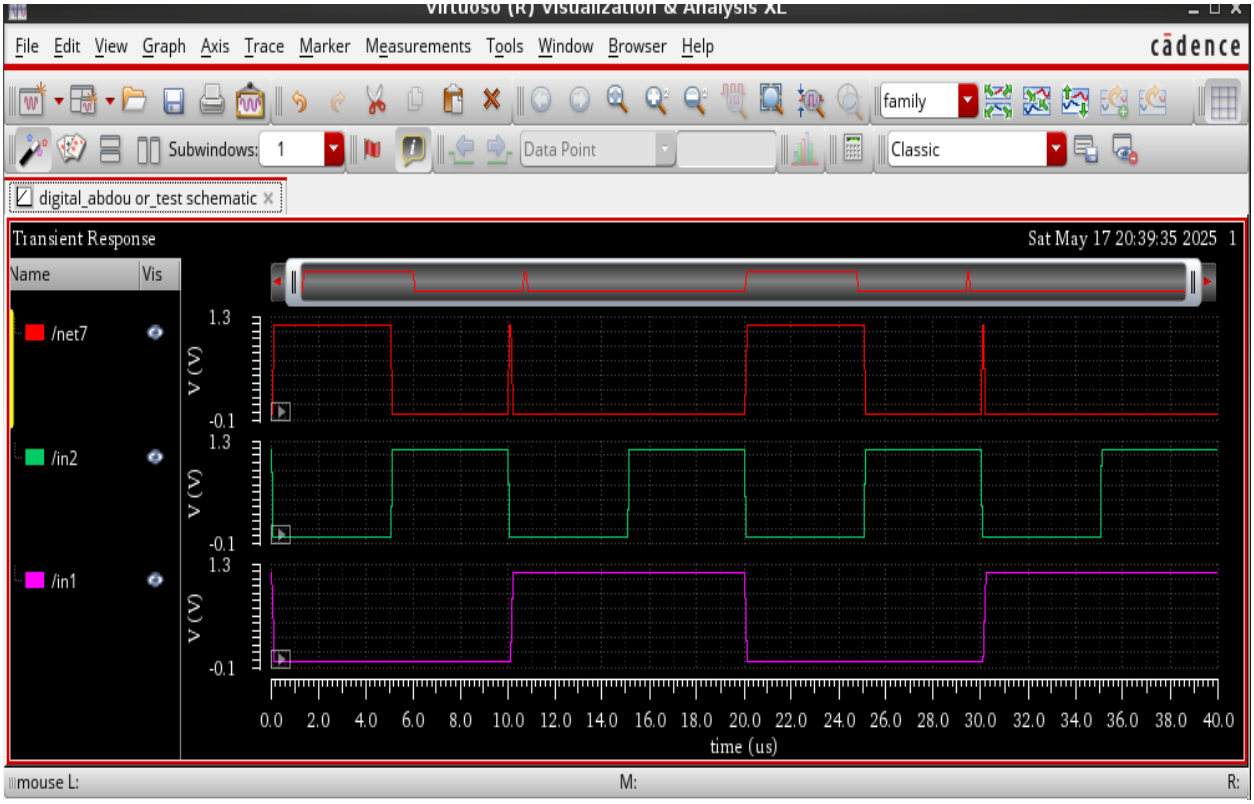
Symbol



Test Bench

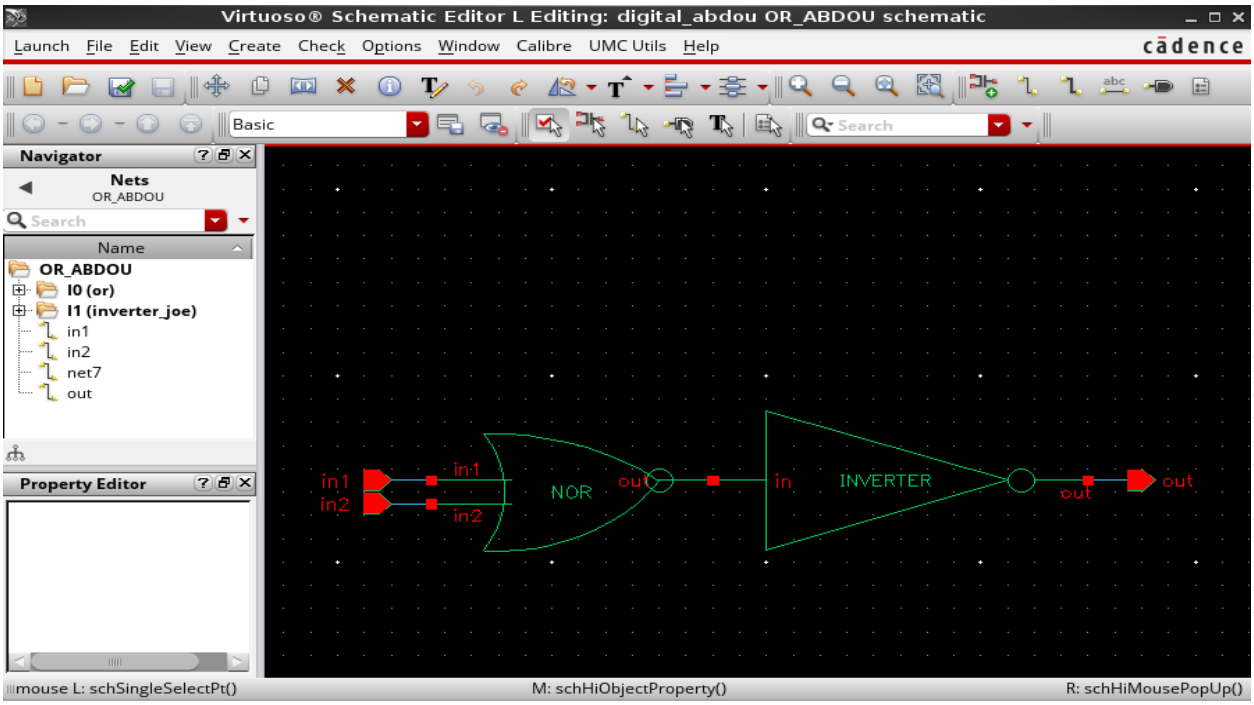


Wave form

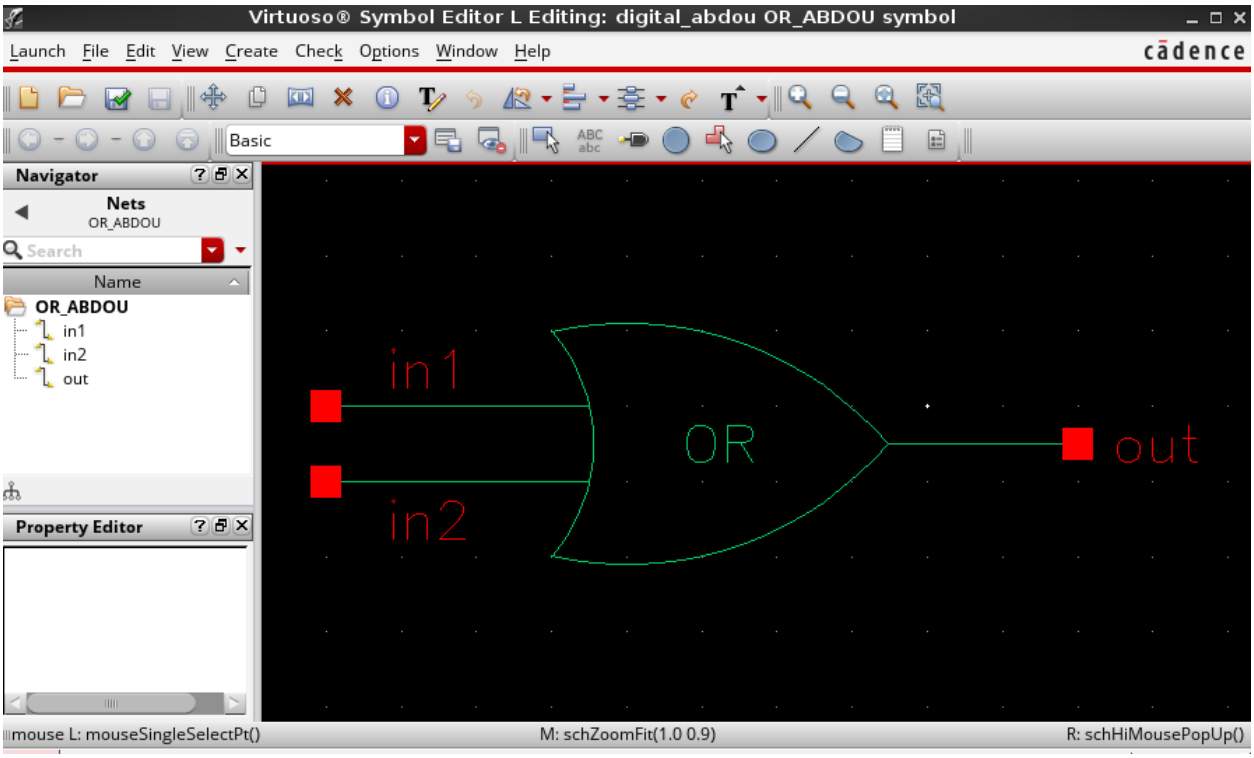


OR

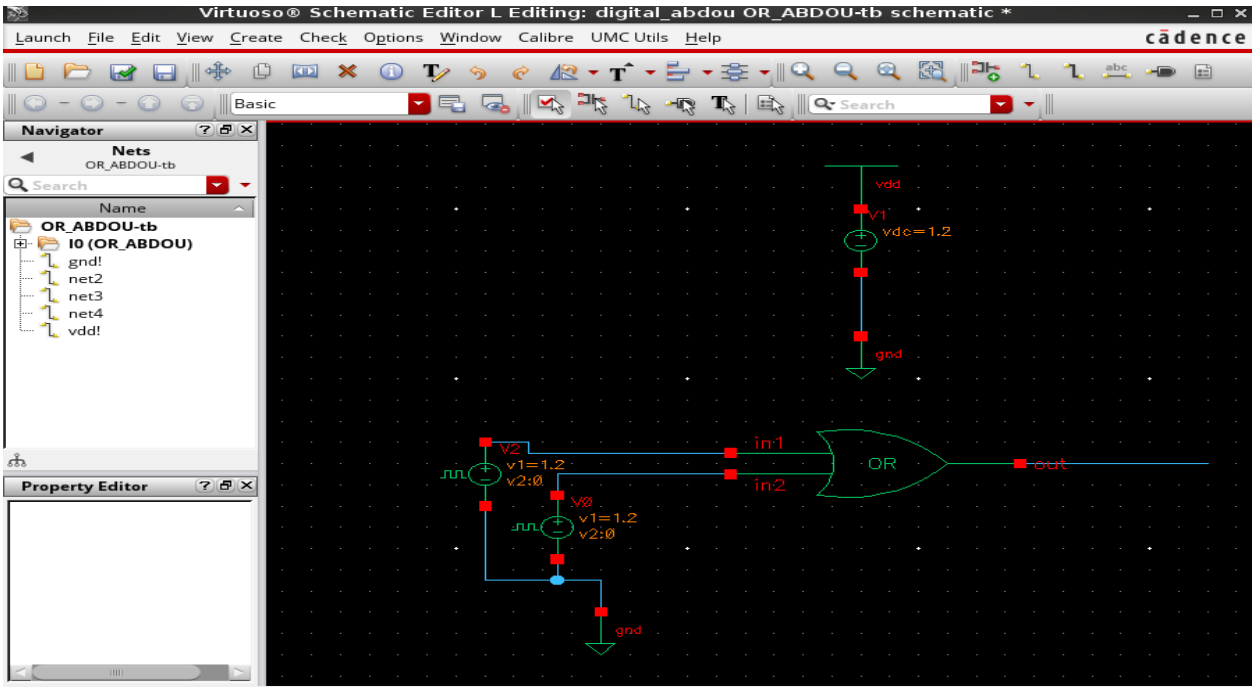
schematic



symbol



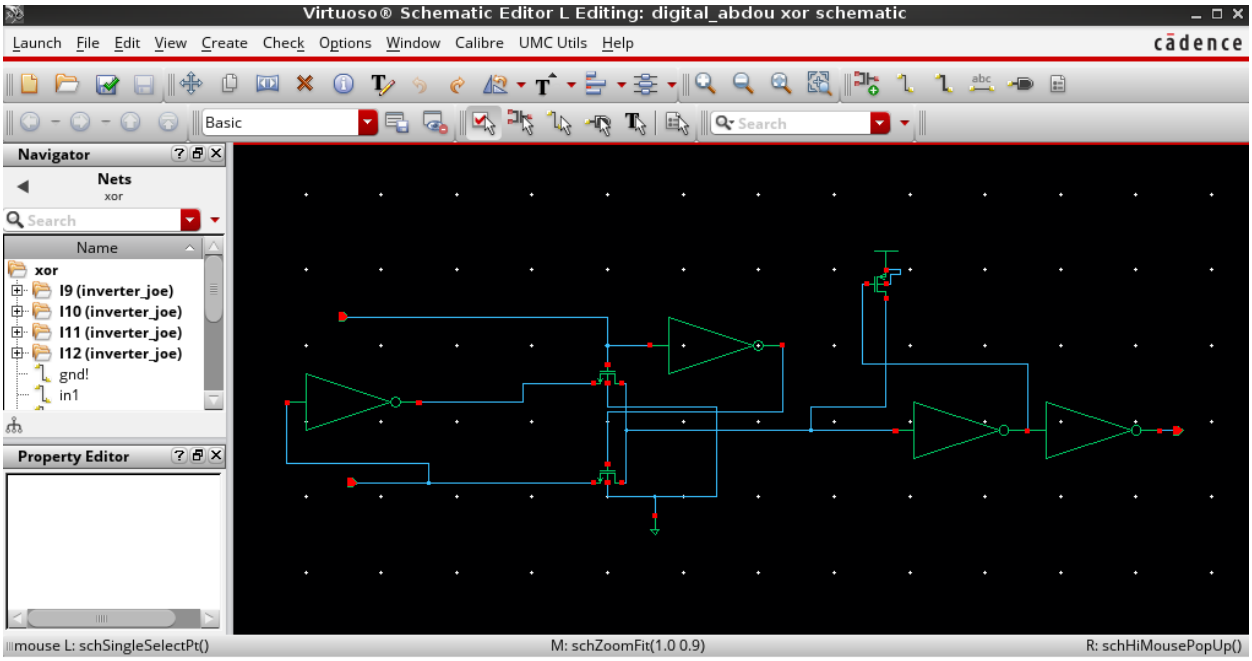
Test Bench



Wave Form

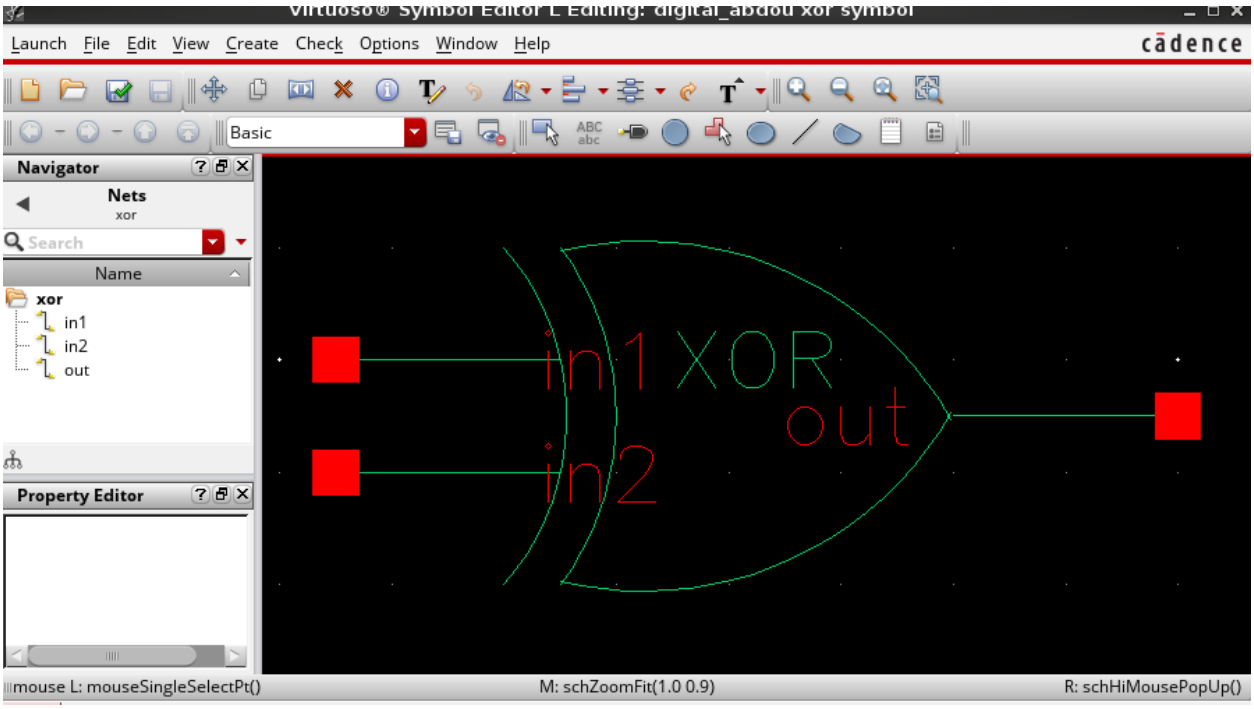
XOR

schematic

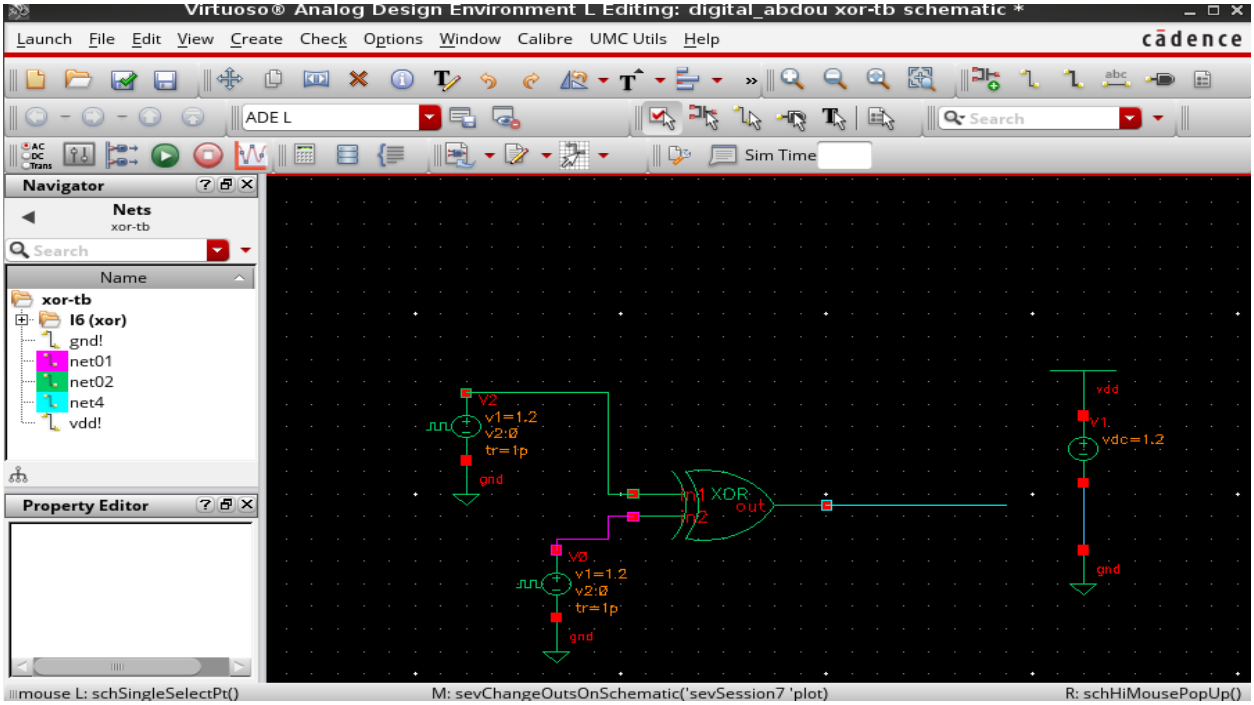


We used PTL (pass-Transistor Logic) because it's so much easier than C-mos implementation and uses less transistors and we solved the voltage drop problem by adding keeper transistor and we made its size so small because it's ratioed

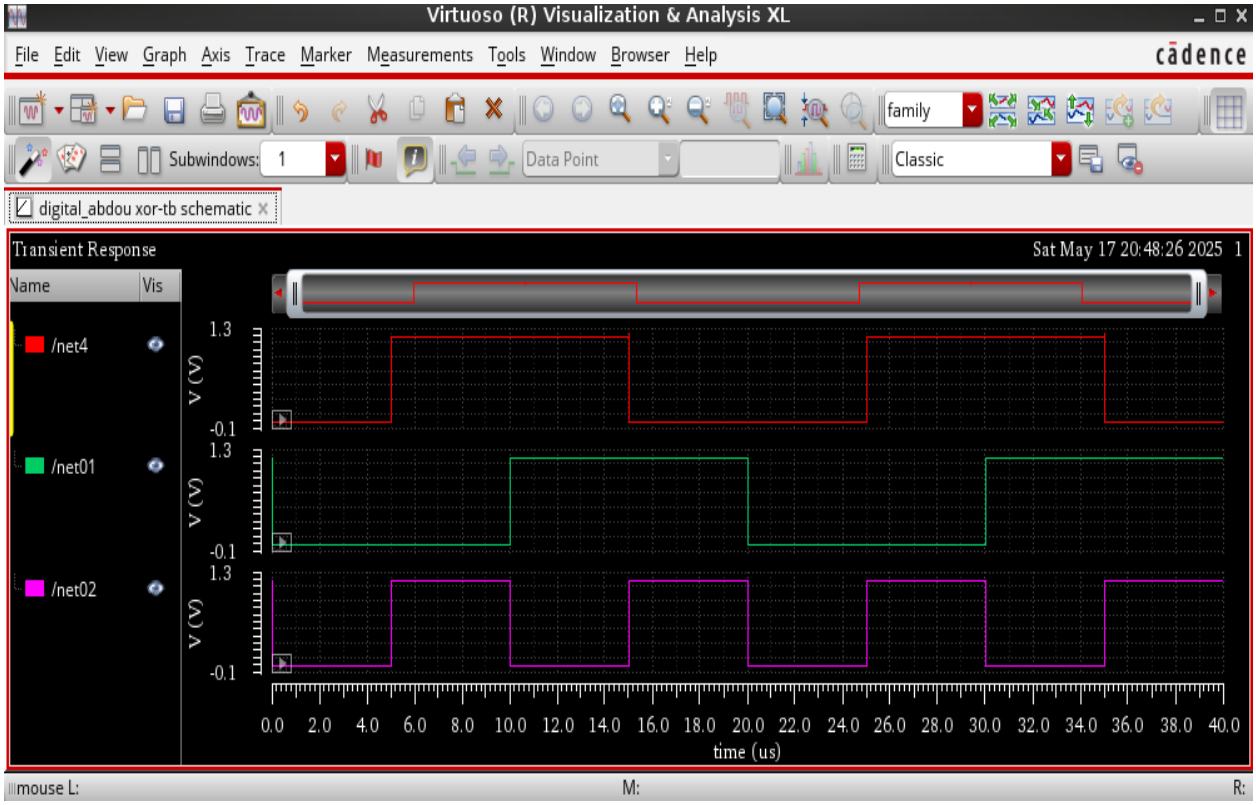
Symbol



Test Bench

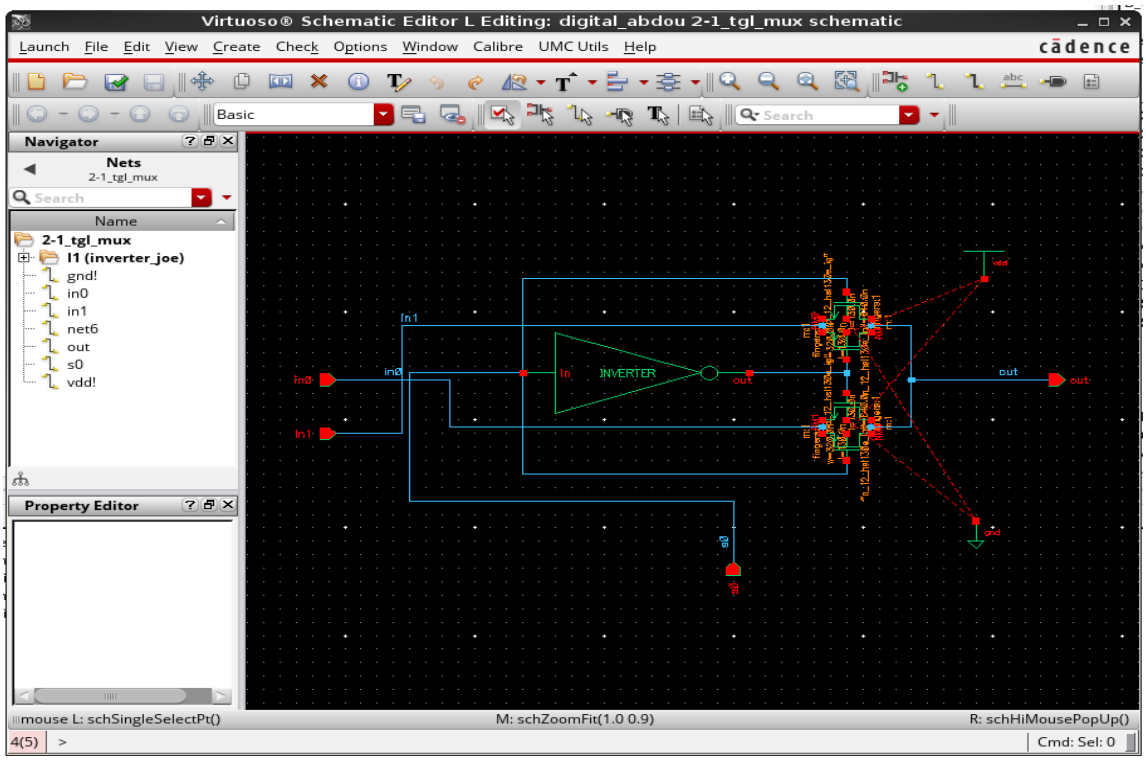


Wave Form



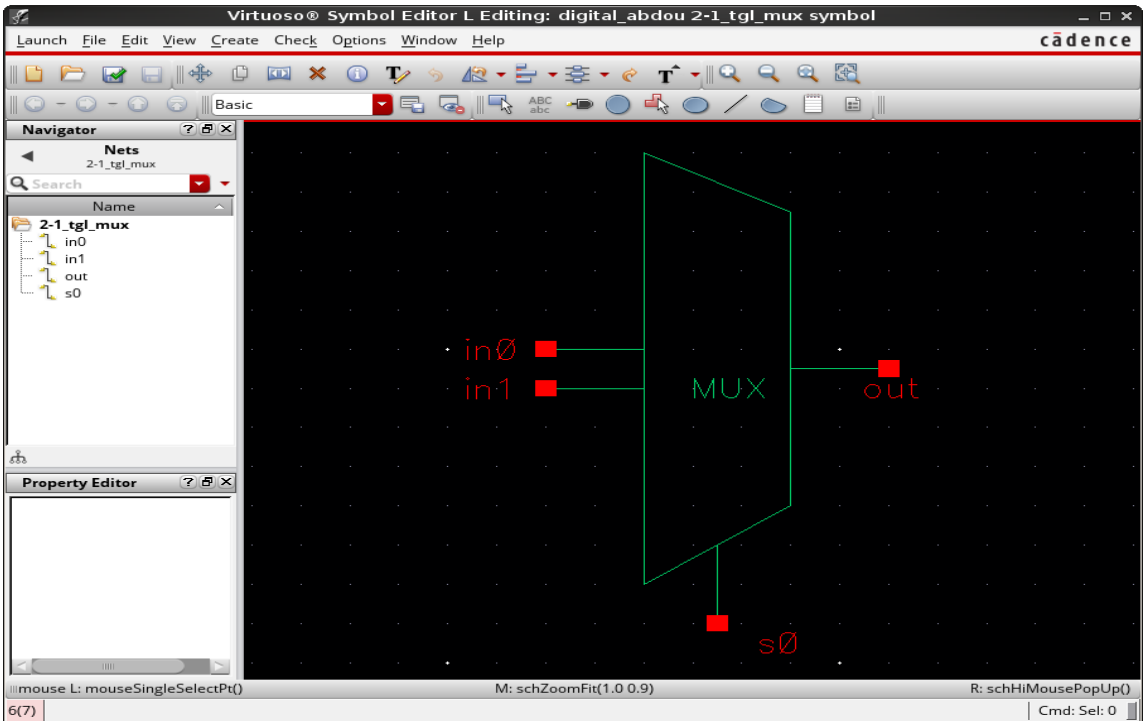
TGL Mux 2-1

Schematic

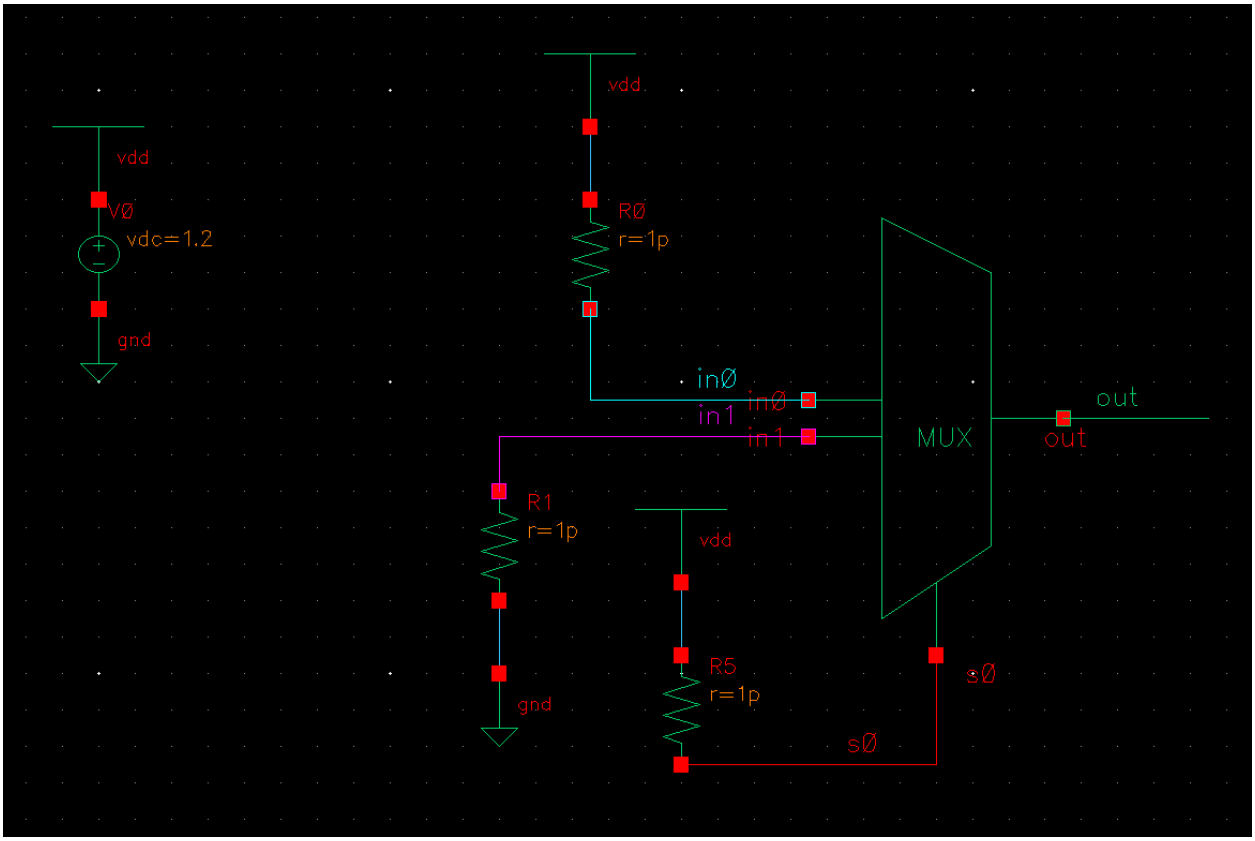


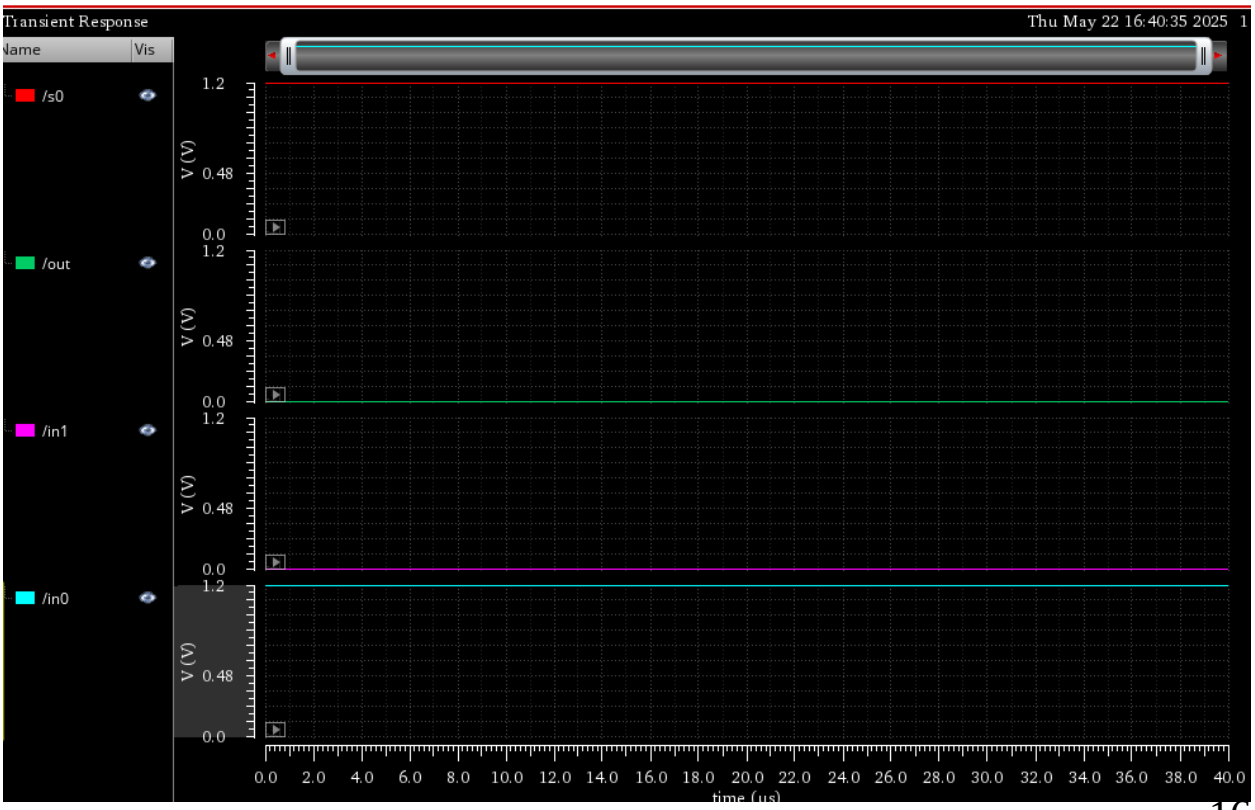
We used TGL technology because it is the most famous and easiest family for implementation of MUX

Symbol



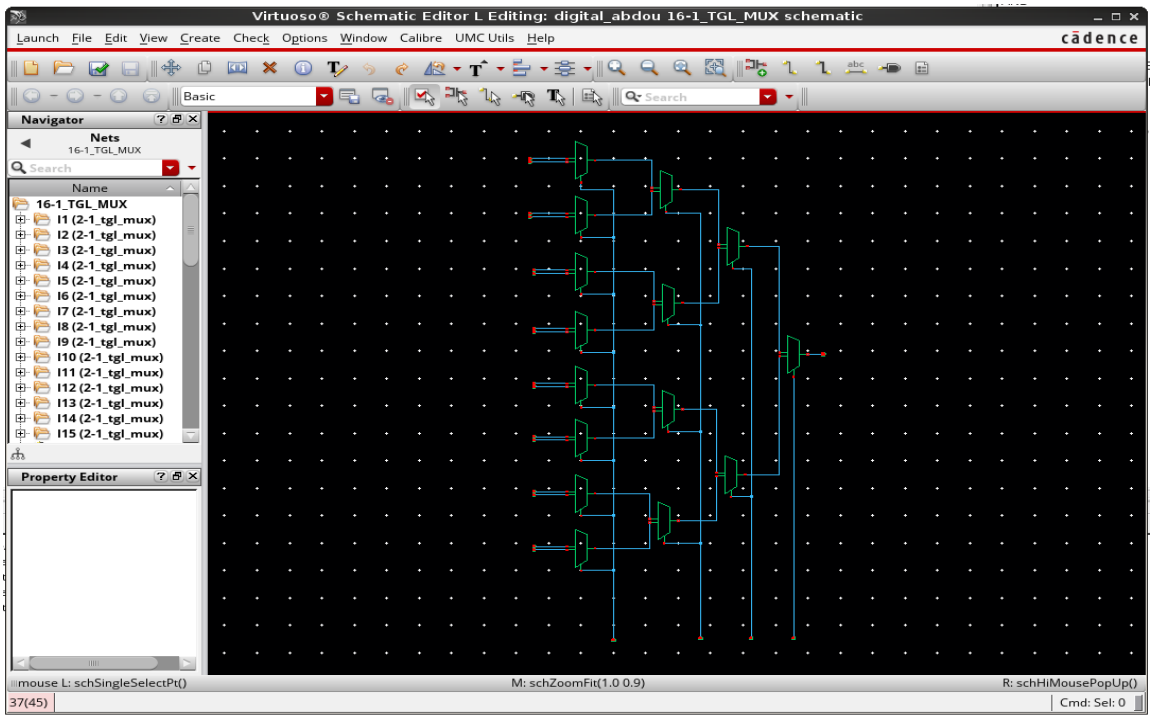
Tgl_2in_Mux testbench





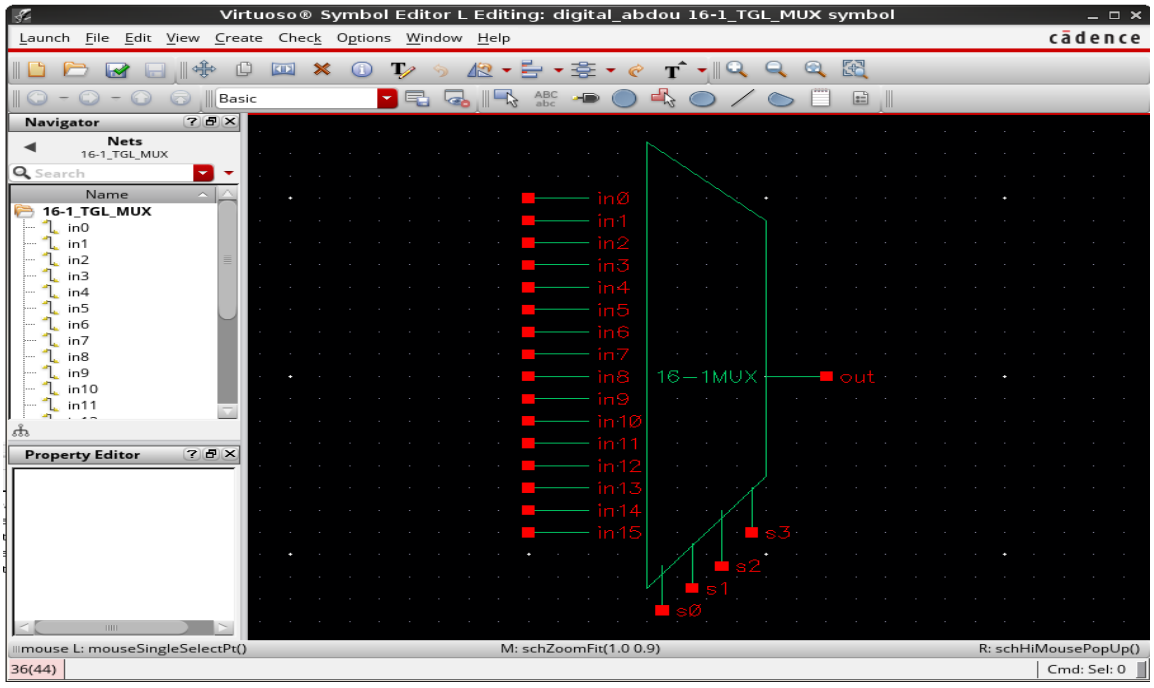
TGL Mux 16-1

schematic

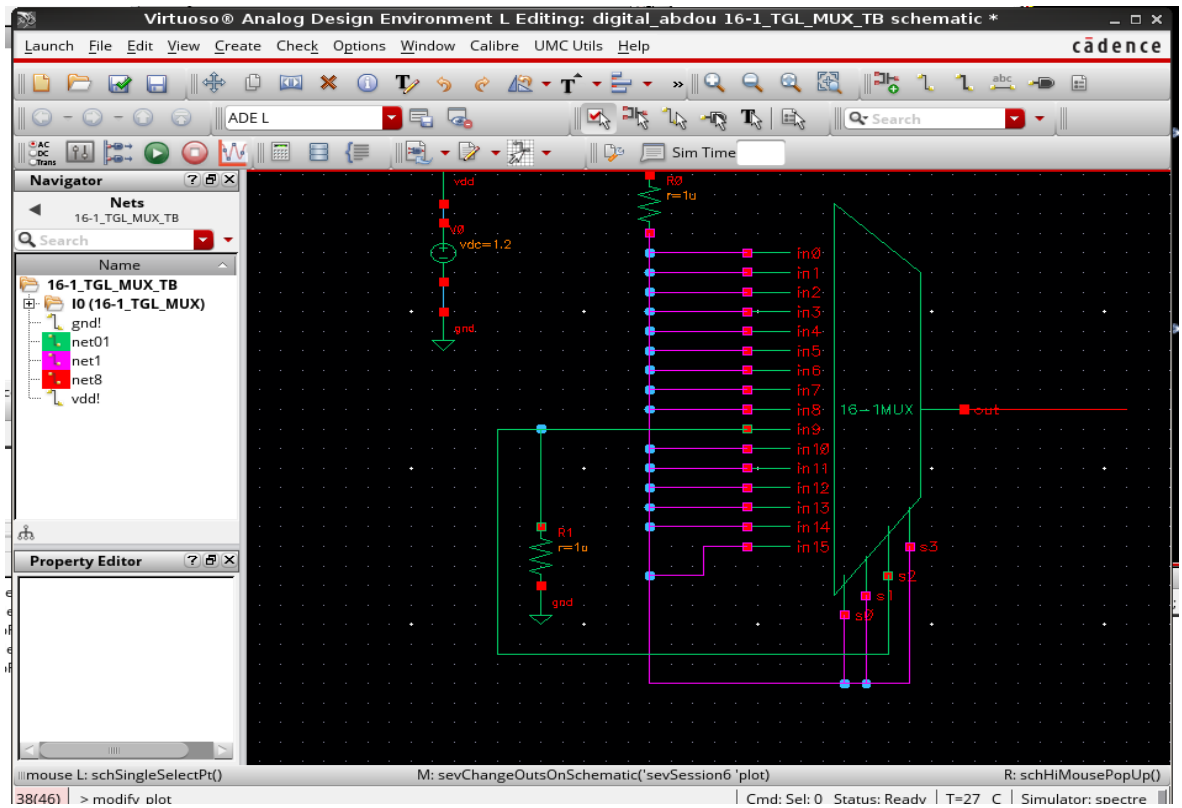


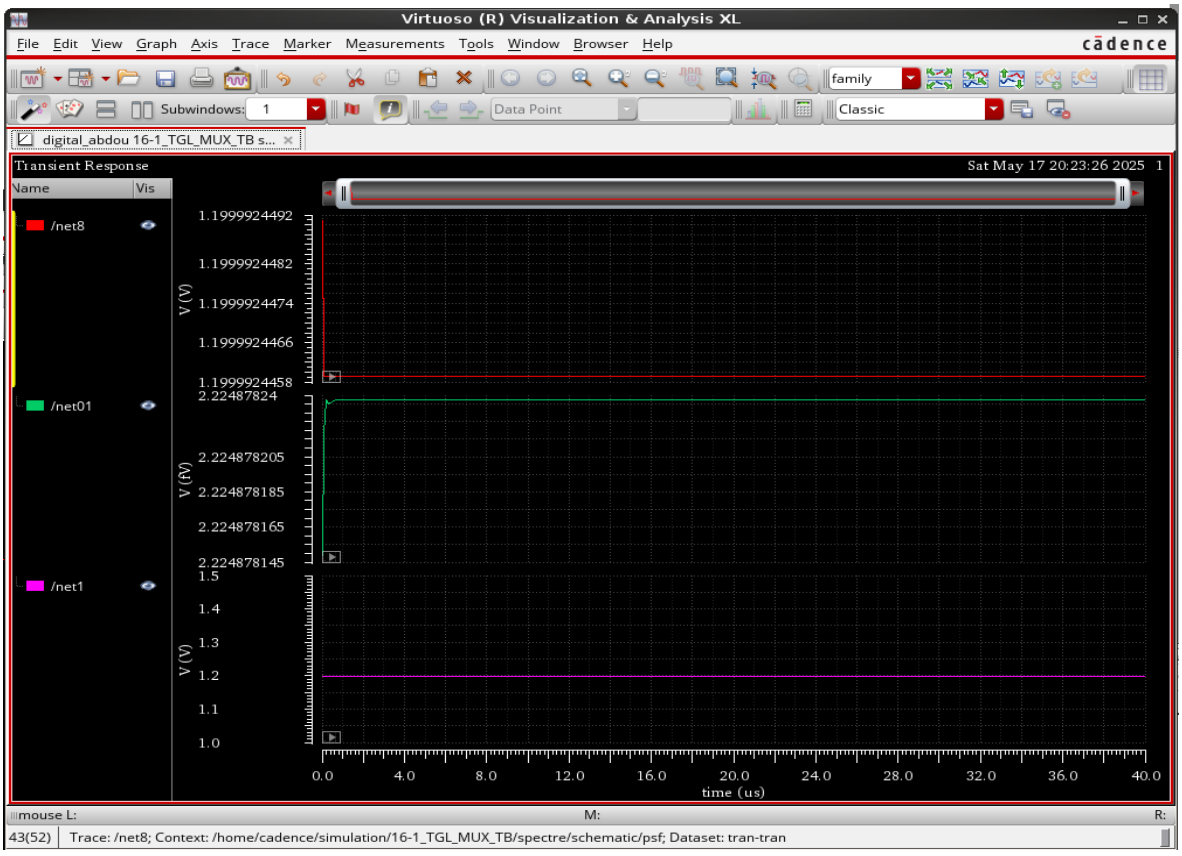
We cascaded 2-1 MUX to create the 16-1 Mux to reduce complications

Symbol



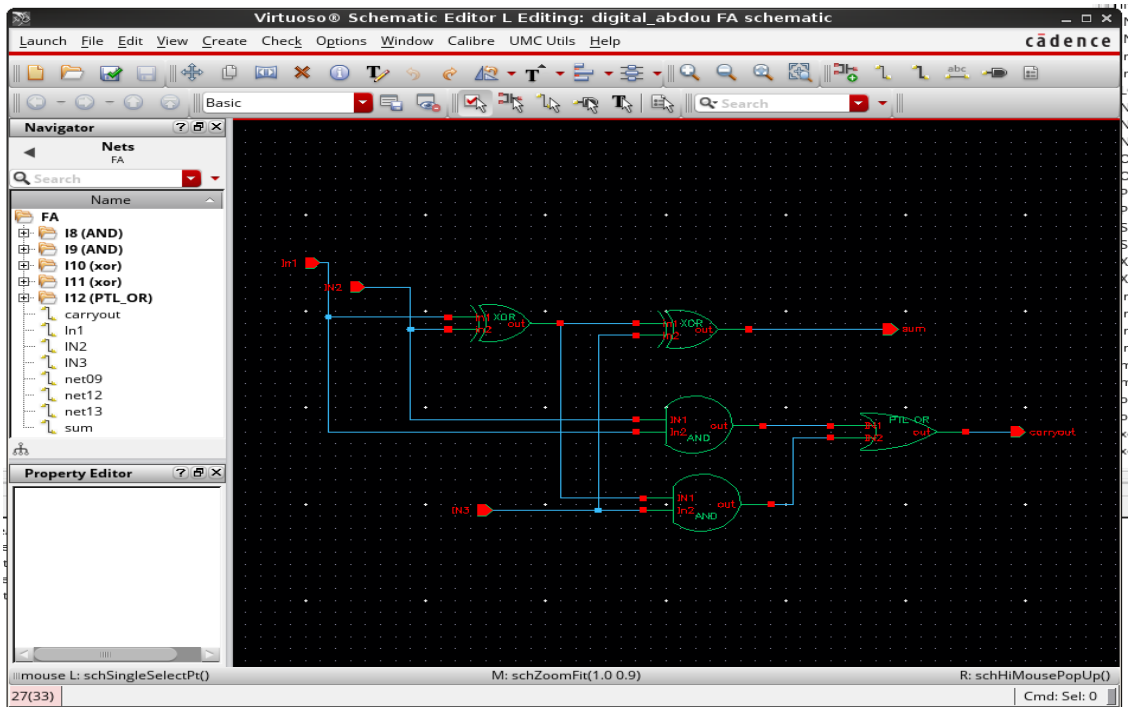
TGL Mux 16-1-TB



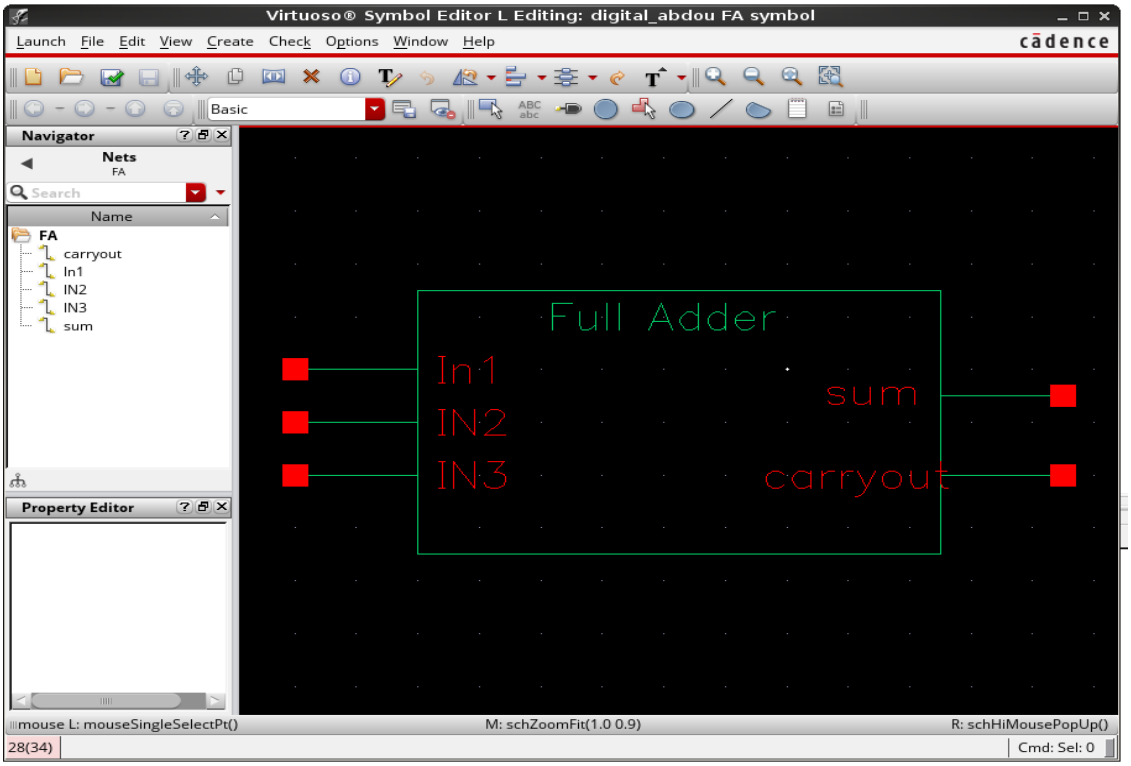


Full Adder

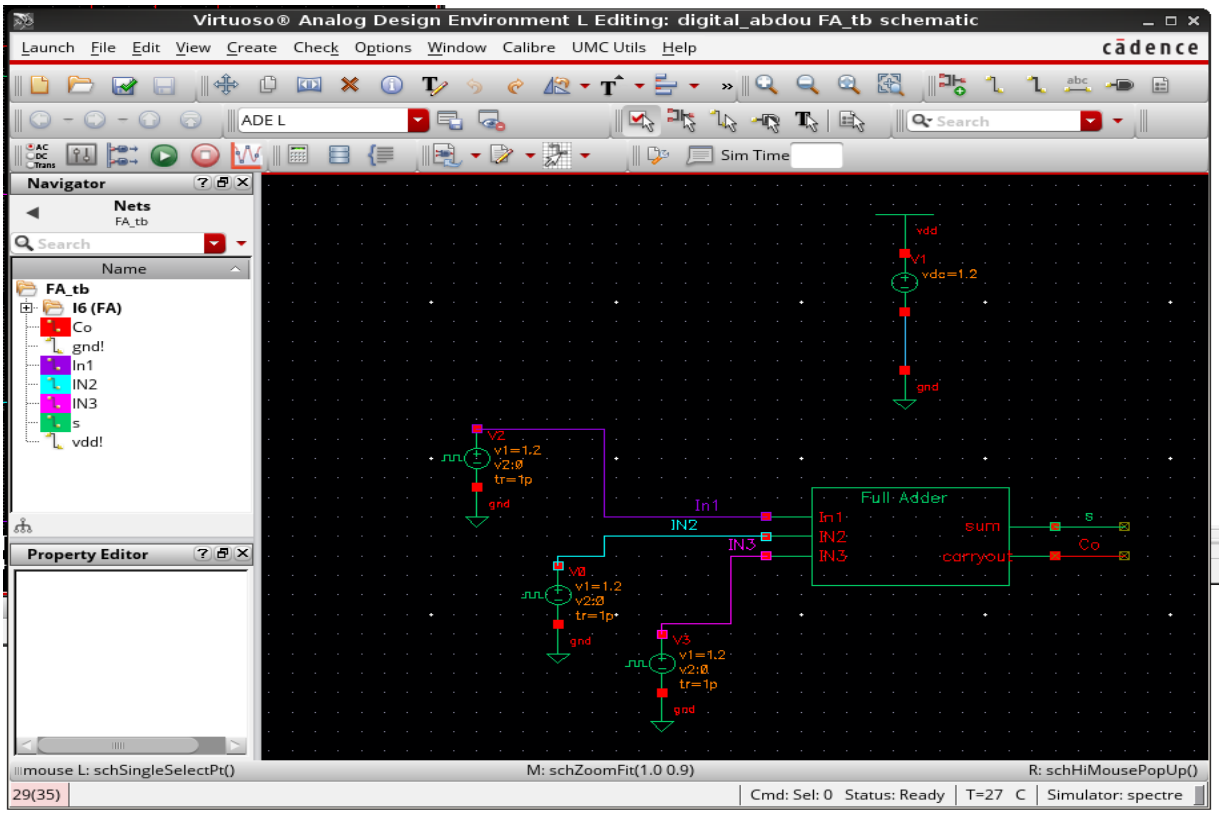
schematic

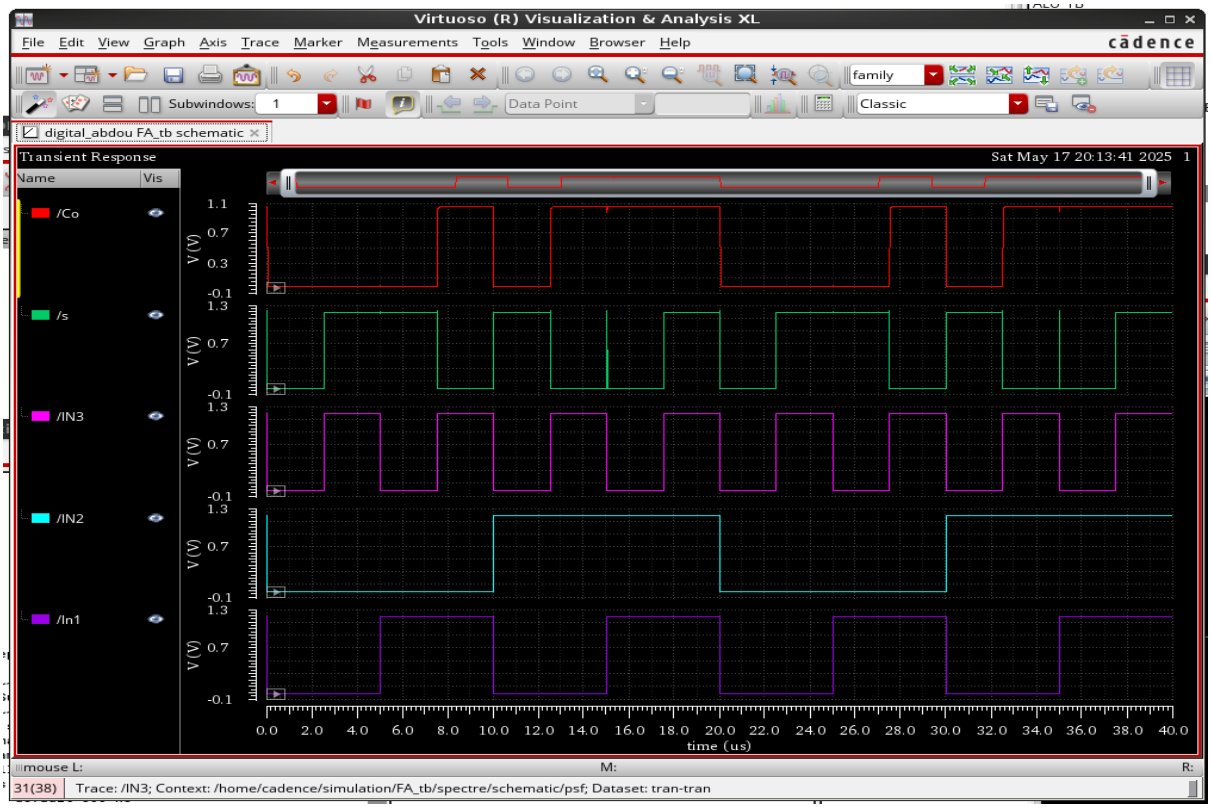


Symbol



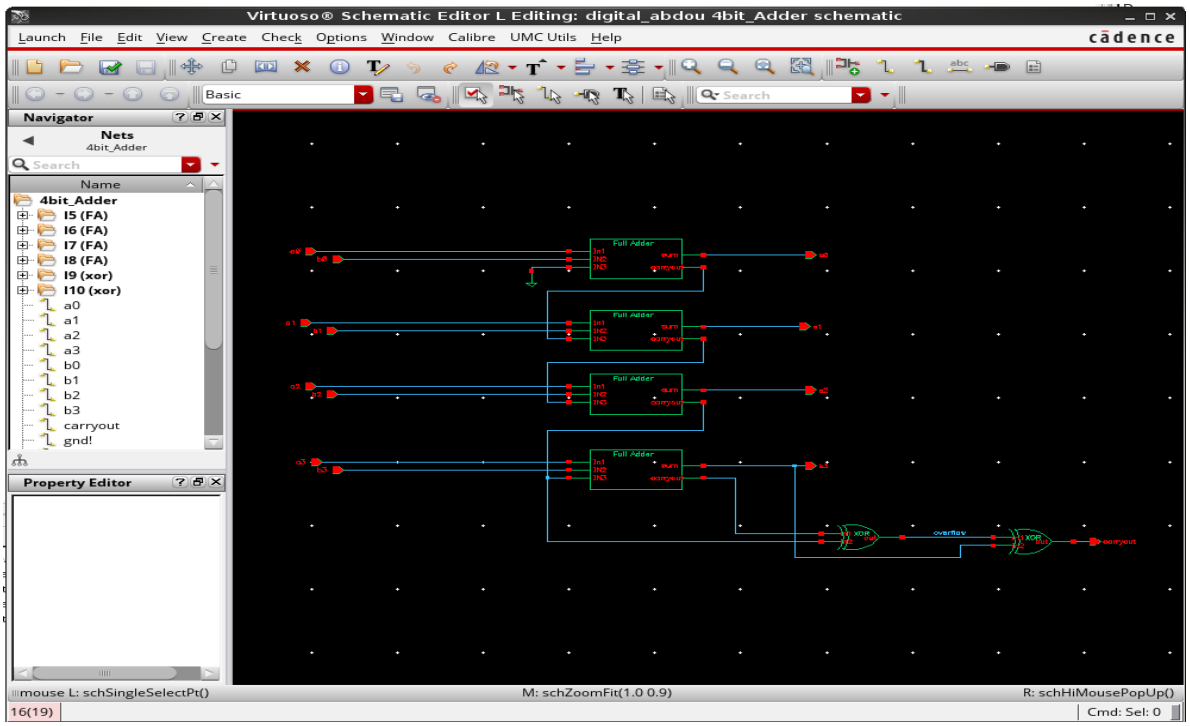
Full_Adder-TB





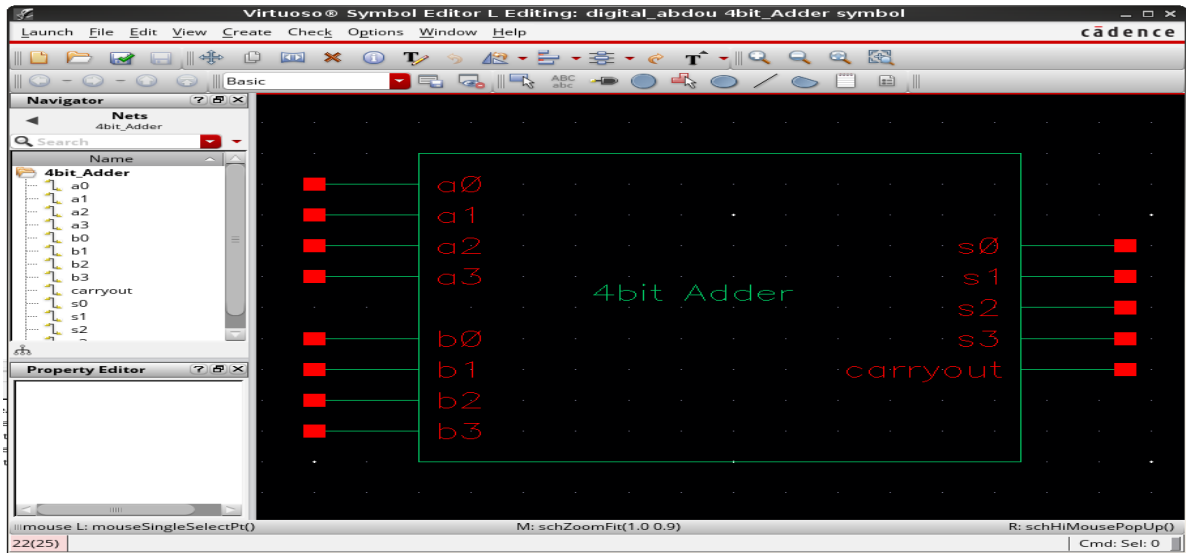
4bit_Adder

schematic

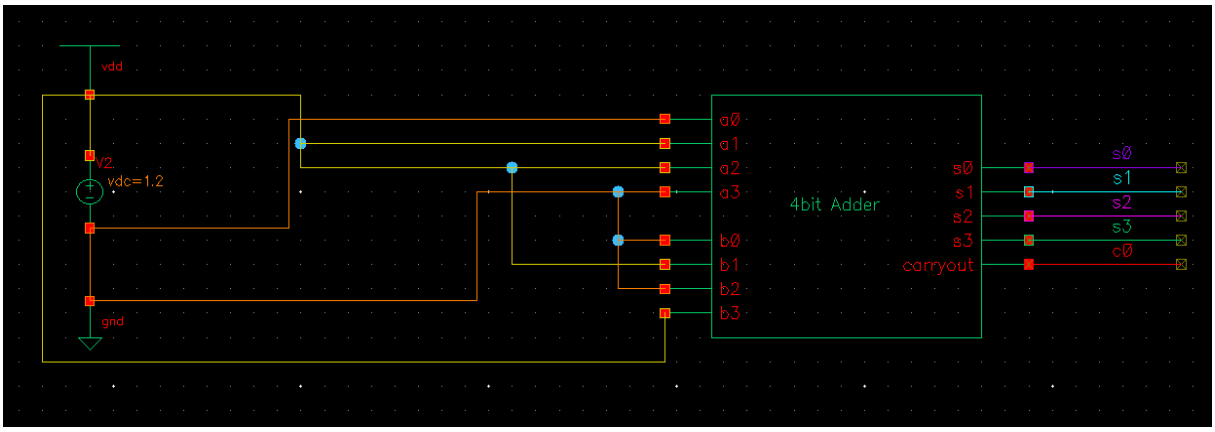


solved the overflow by x-oring last carry in with last carry out to implement overflow flag and we x-or the overflow with the last sum as and extended the output as it is the right bit

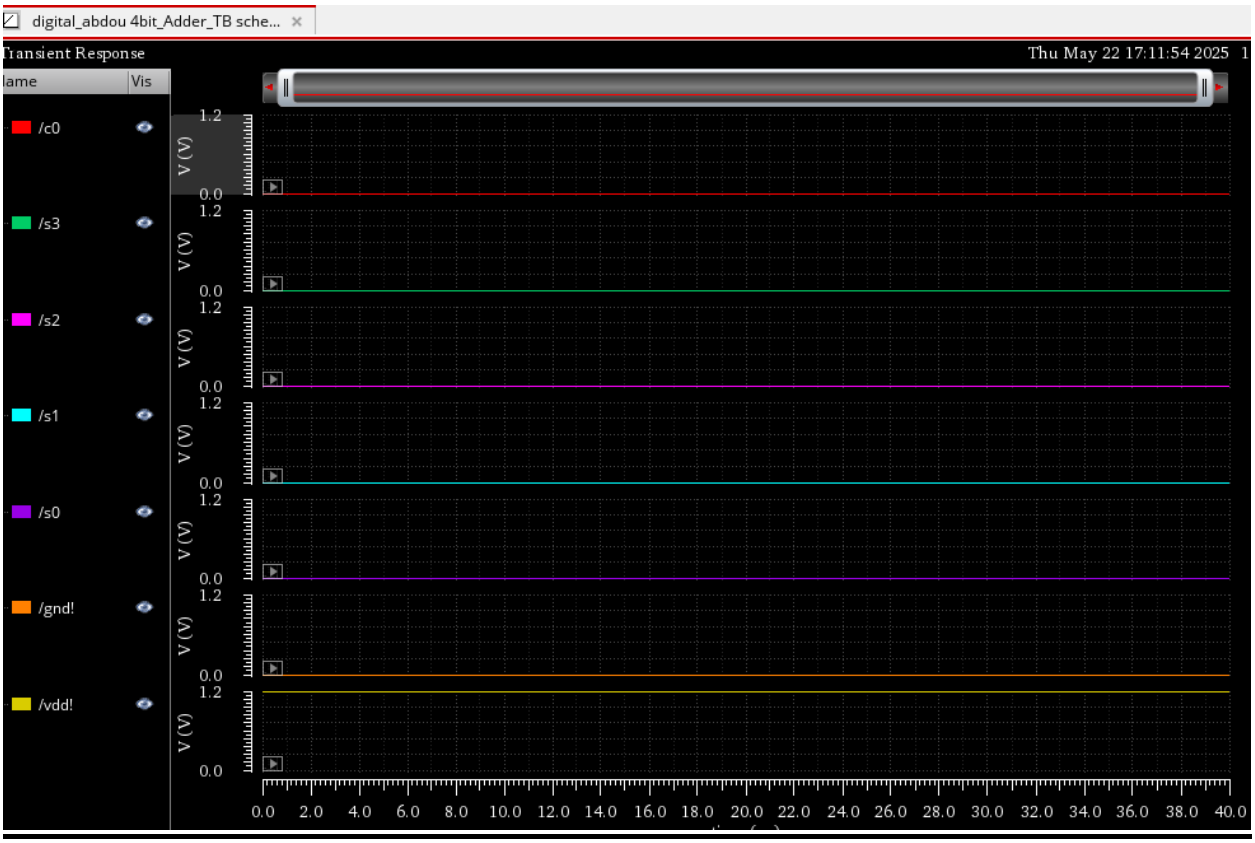
Symbol



4bit_Adder-TB

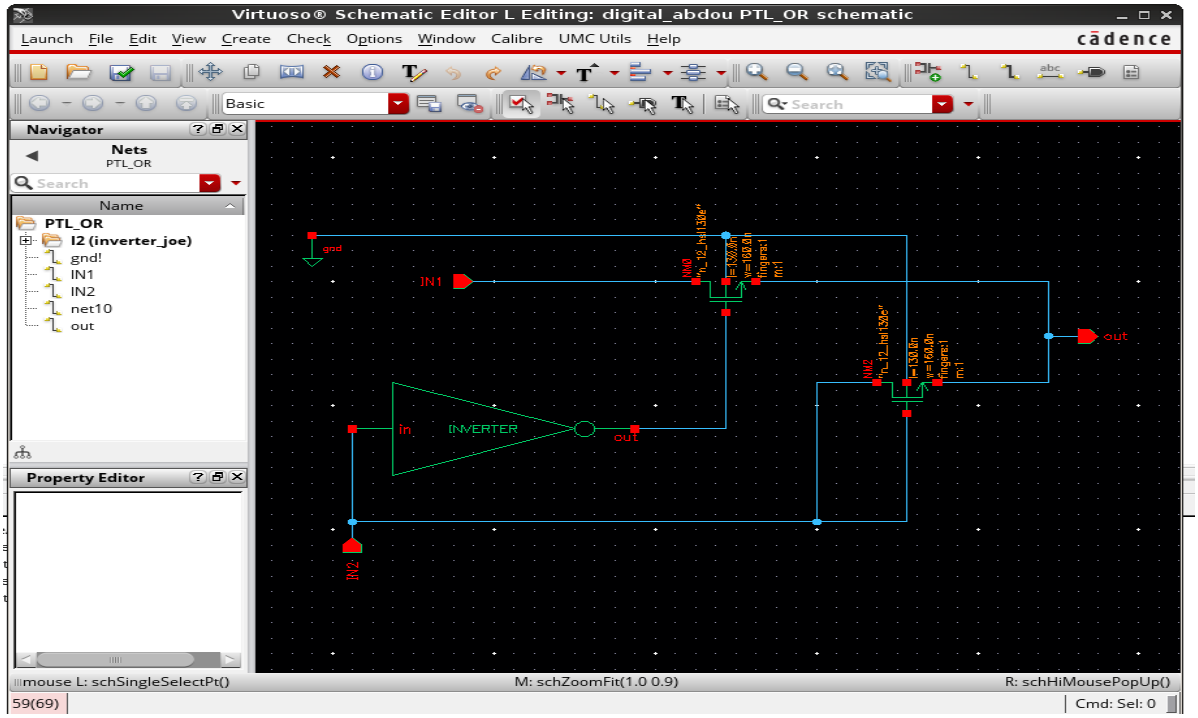


As shown a=6, b=-6, out→0

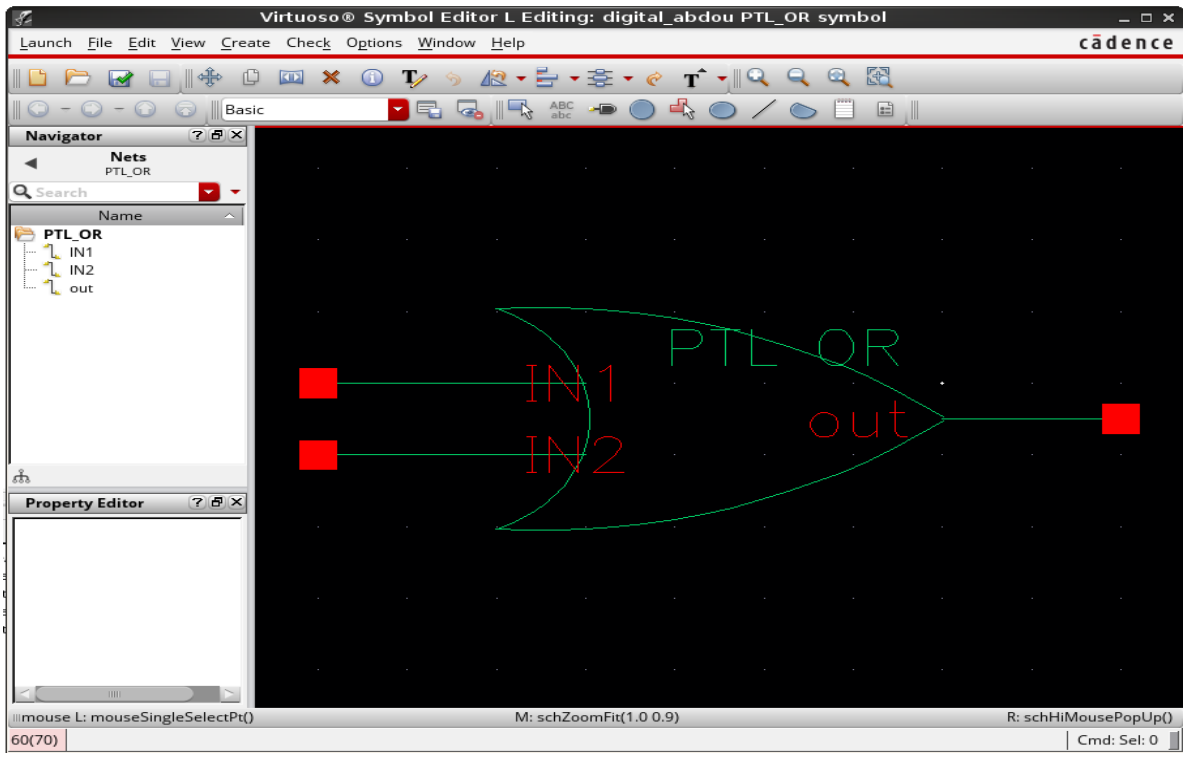


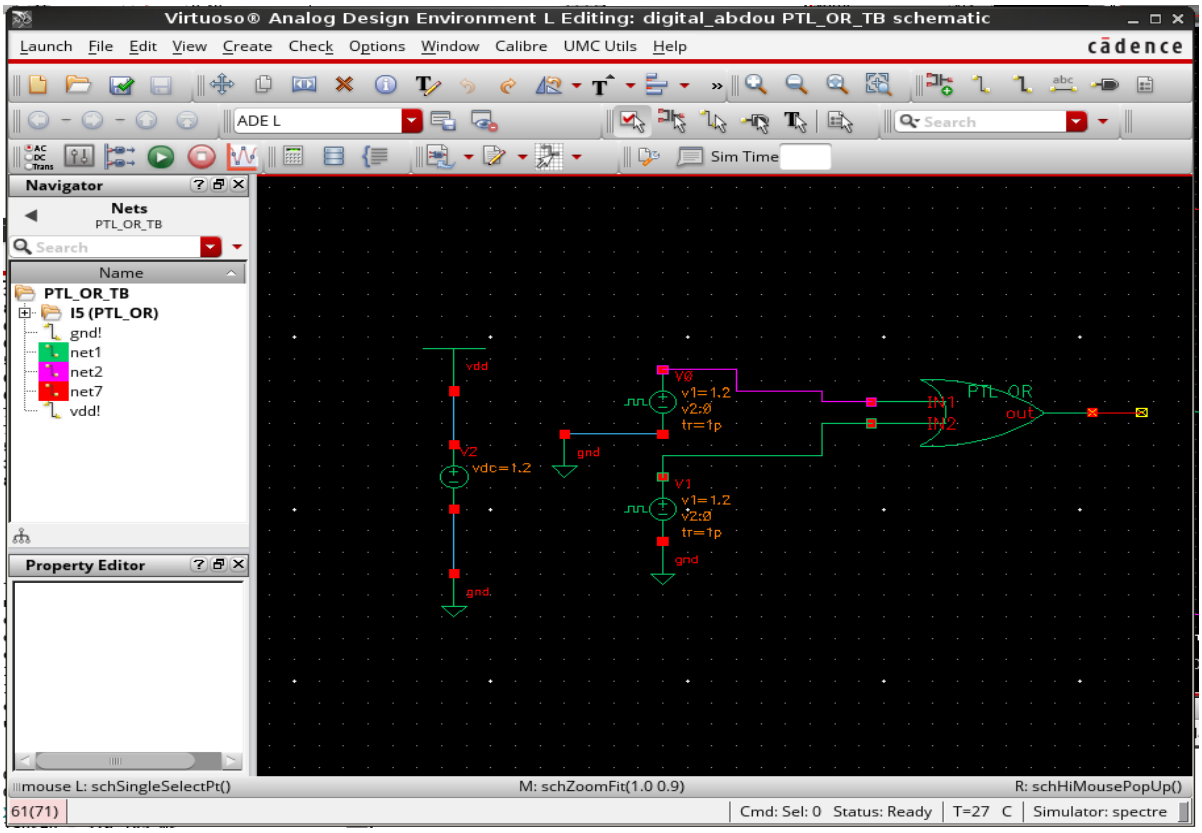
PTL OR

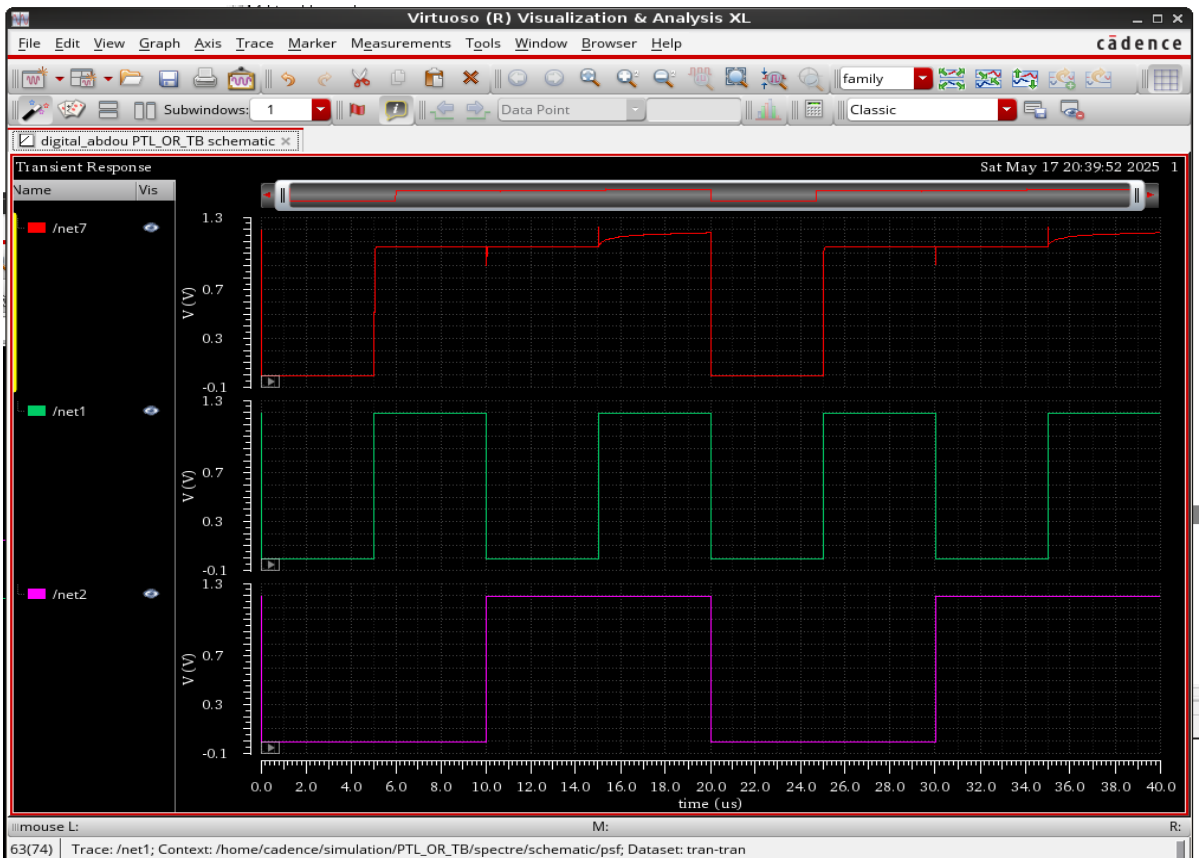
schematic



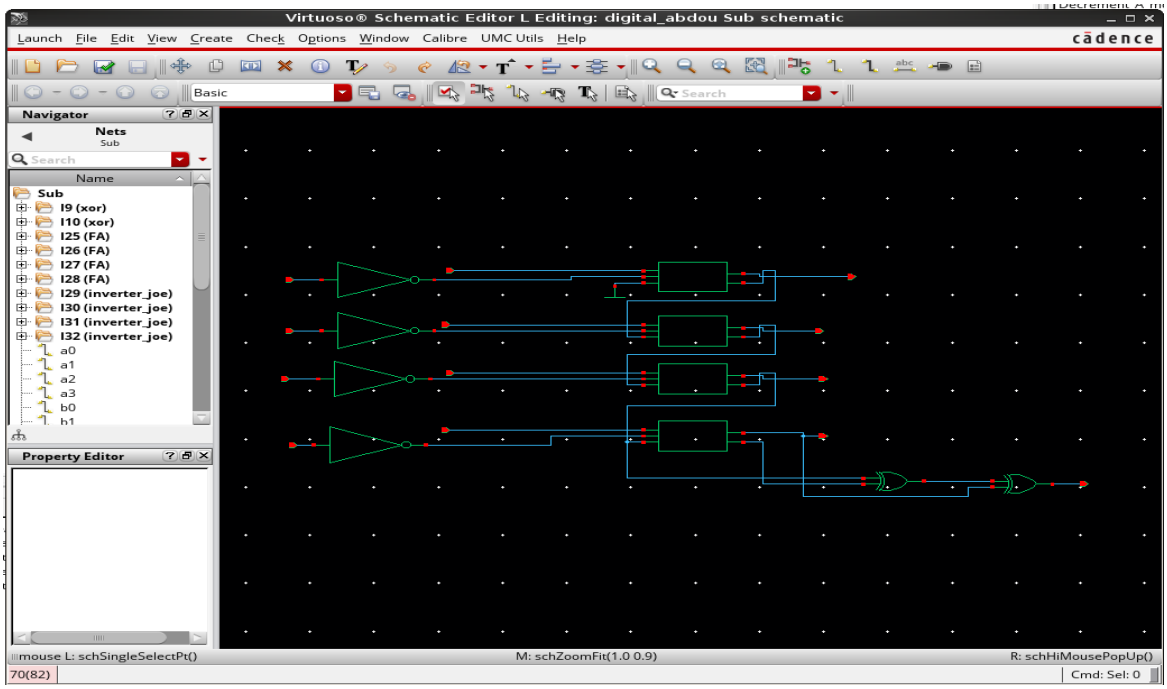
Symbol




Virtuoso® Analog Design Environment L Editing: digital_abdou PTL_OR_TB schematic
Launch File Edit View Create Check Options Window Calibre UMC Utils Help
cadence
ADE L
Search
Sim Time
Navigator
Nets
PTL_OR_TB
Search
Name
PTL_OR_TB
I5 (PTL_OR)
gnd!
net1
net2
net7
vdd!
Property Editor
vdd
V2
vdc=1.2
gnd
gnd.
V0
v1=1.2
v2:0
tr=1p
gnd
V1
v1=1.2
v2:0
tr=1p
gnd
PTL_OR
IN1
IN2
out
mouse L: schSingleSelectPt()
M: schZoomFit(1.0 0.9)
R: schHiMousePopUp()
61(71)
Cmd: Sel: 0 Status: Ready T=27 C Simulator: spectre

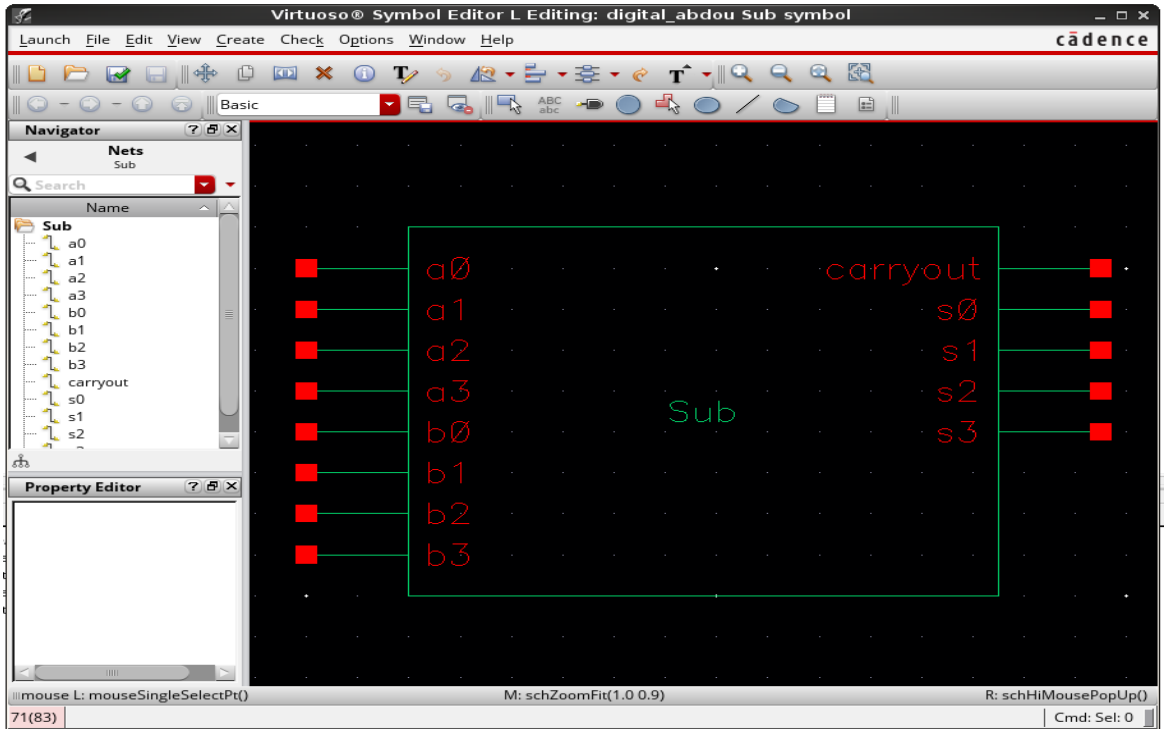

Virtuoso (R) Visualization & Analysis XL
File Edit View Graph Axis Trace Marker Measurements Tools Window Browser Help
cadence
family
Subwindows: 1
Data Point
Classic
digital_abdou PTL_OR_TB schematic
Transient Response
Sat May 17 20:39:52 2025 1
Name
Vis
/net7
/net1
/net2
V (V)
1.3
0.7
0.3
-0.1
0.0 2.0 4.0 6.0 8.0 10.0 12.0 14.0 16.0 18.0 20.0 22.0 24.0 26.0 28.0 30.0 32.0 34.0 36.0 38.0 40.0
time (us)
mouse L:
M:
R:
63(74)
Trace: /net1; Context: /home/cadence/simulation/PTL_OR_TB/spectre/schematic/psf; Dataset: tran-tran

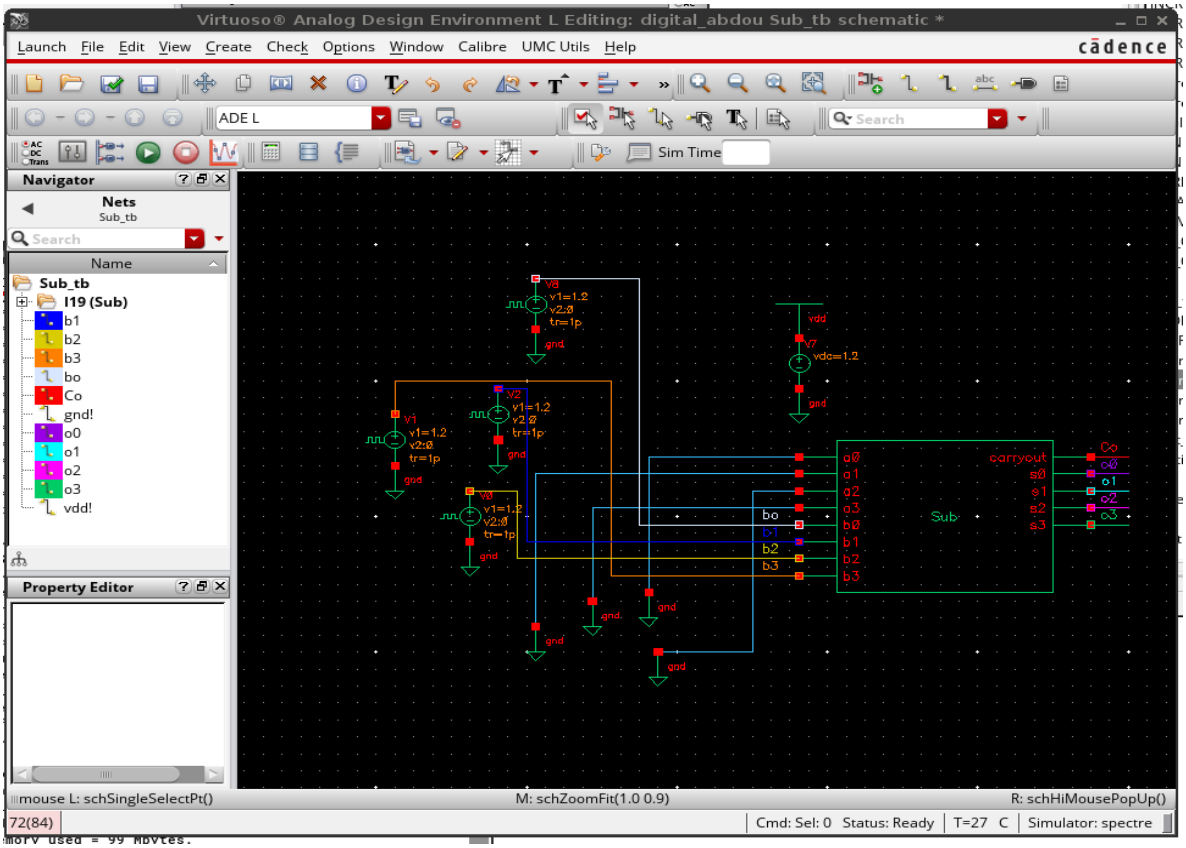
Sub

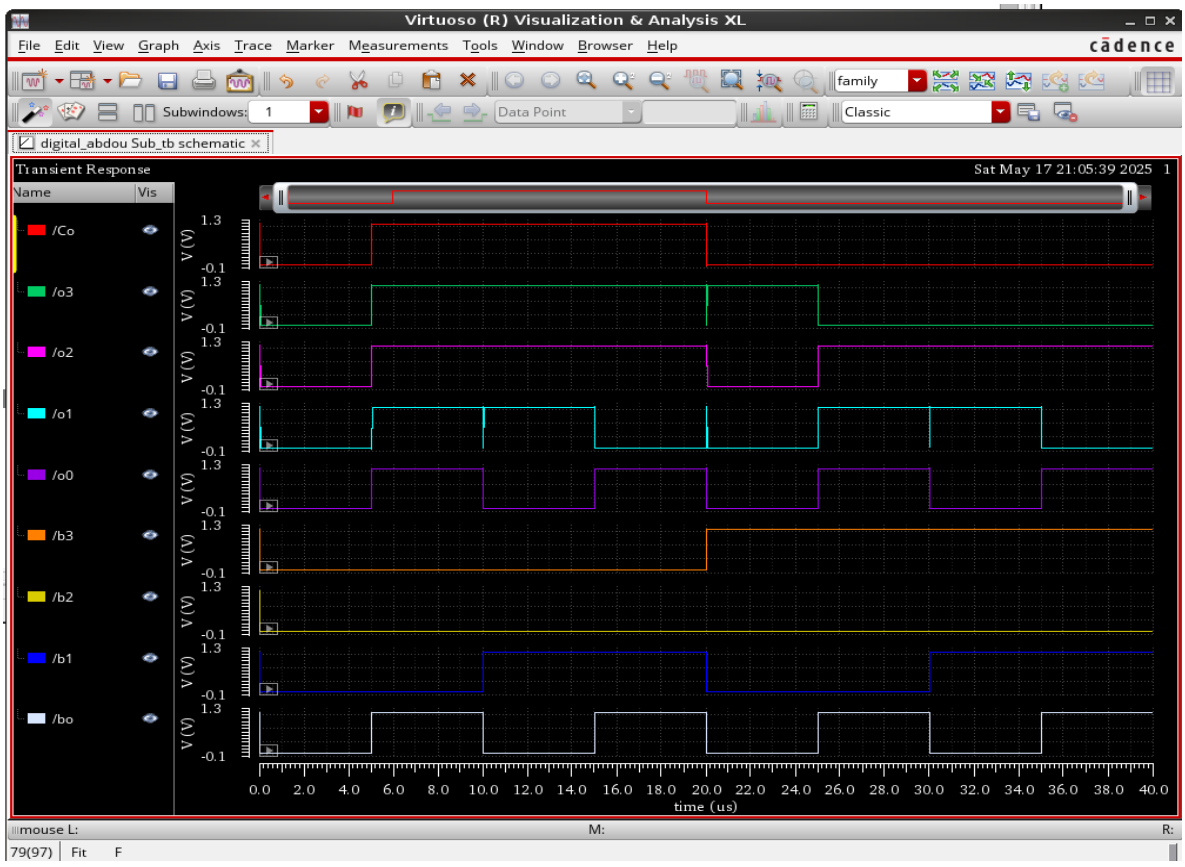


Same here, the overflow we put 2 XORs to solve the overflow and the extra here that we put B bar and put the first carry in equal 1 to generate the two's complement of B

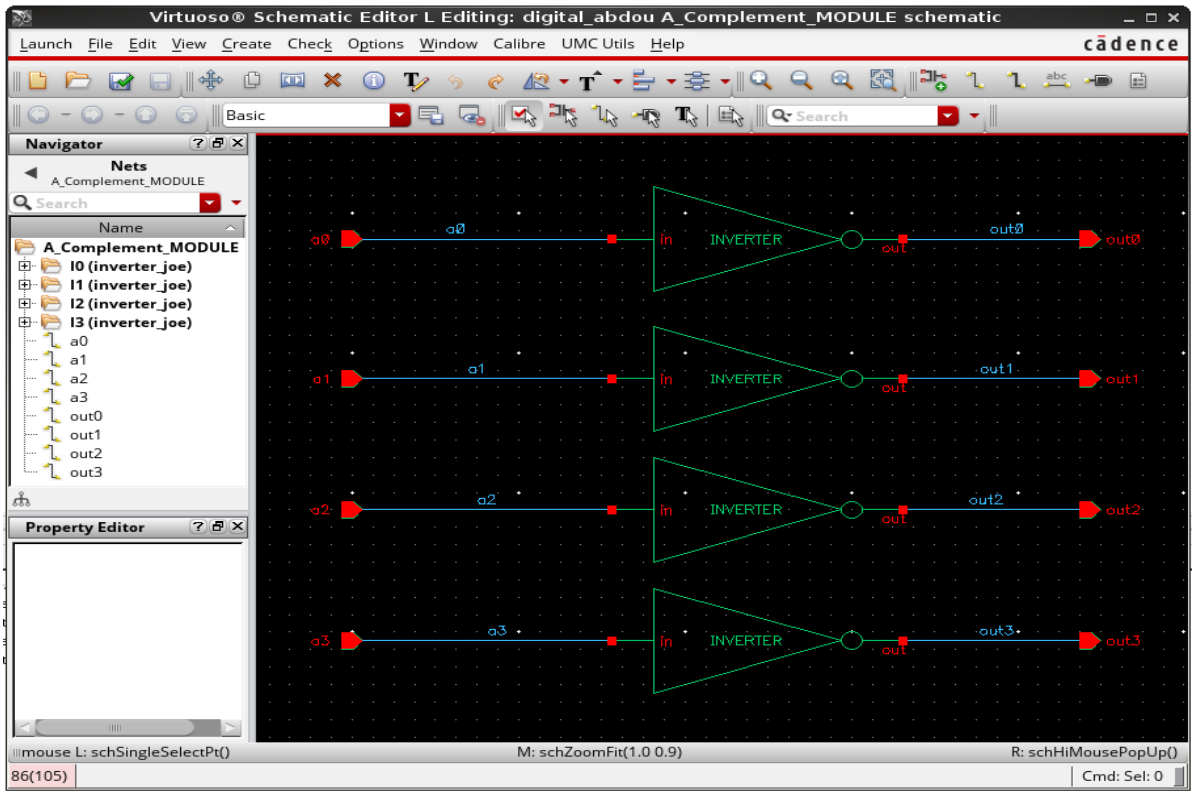
Symbol



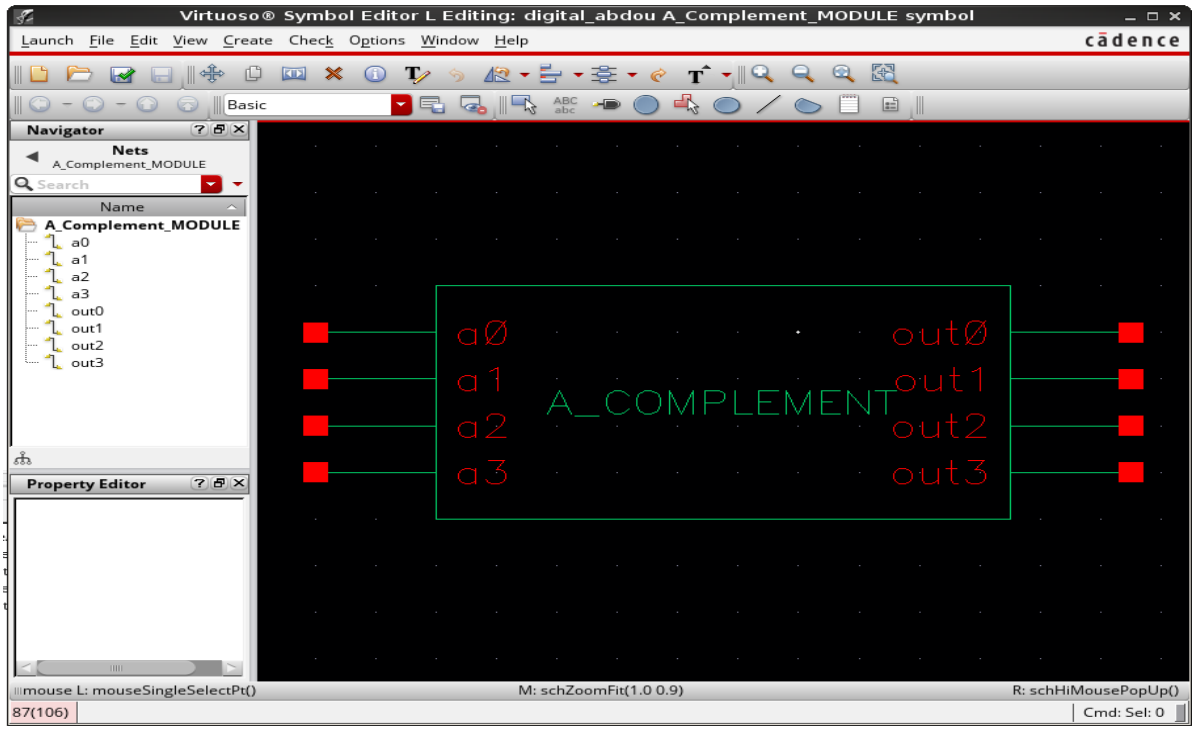

Virtuoso® Analog Design Environment L Editing: digital_abdou Sub_tb schematic *
Launch File Edit View Create Check Options Window Calibre UMC Utils Help
cadence
ADE L
Sim Time
Navigator
Nets
Sub_tb
Name
Sub_tb
I19 (Sub)
b1
b2
b3
bo
Co
gnd!
o0
o1
o2
o3
vdd!
Property Editor
Sub
carryout
mouse L: schSingleSelectPt()
M: schZoomFit(1.0 0.9)
R: schHiMousePopUp()
72(84)
Cmd: Sel: 0 Status: Ready
T=27 C
Simulator: spectre


Virtuoso (R) Visualization & Analysis XL
File Edit View Graph Axis Trace Marker Measurements Tools Window Browser Help
cadence
digital_abdou Sub_tb schematic
Transient Response
Sat May 17 21:05:39 2025
Name
Vis
/Co
/o3
/o2
/o1
/o0
/b3
/b2
/b1
/bo
V (V)
time (us)
mouse L:
M:
R:
79(97)
Fit
F

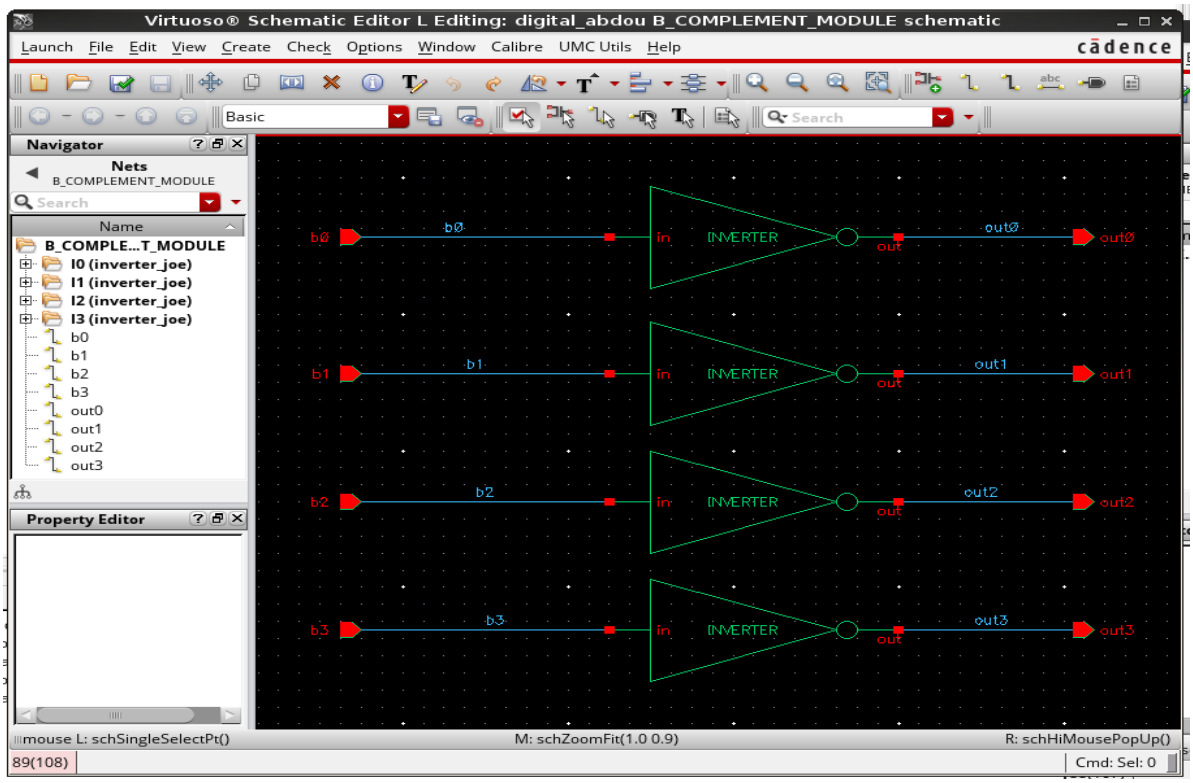
A Complement Module



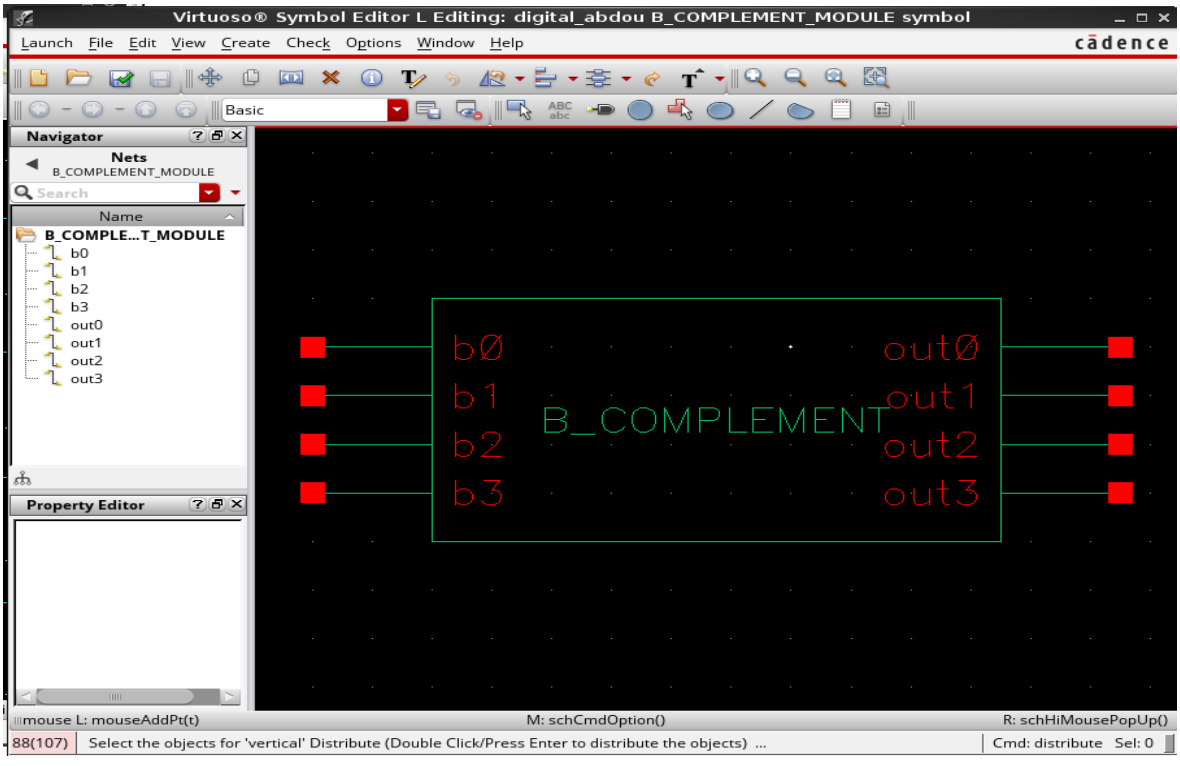
Symbol



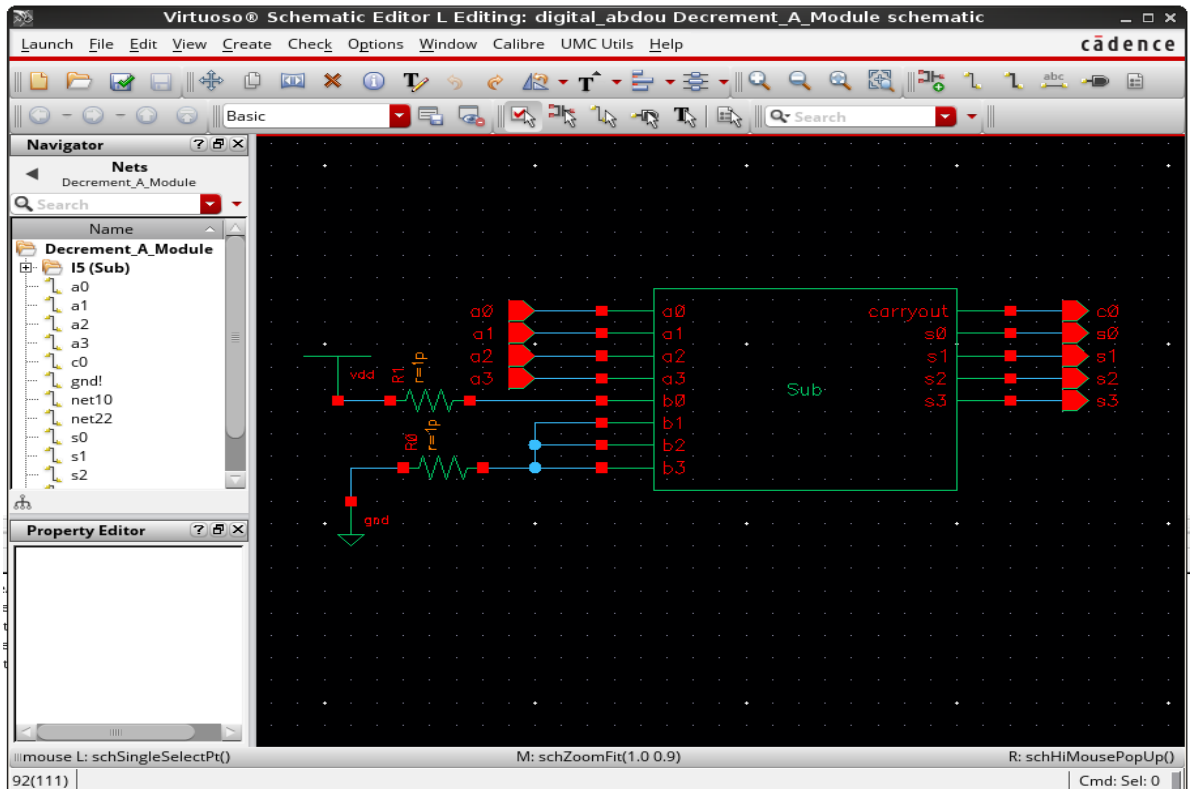
B Complement Module



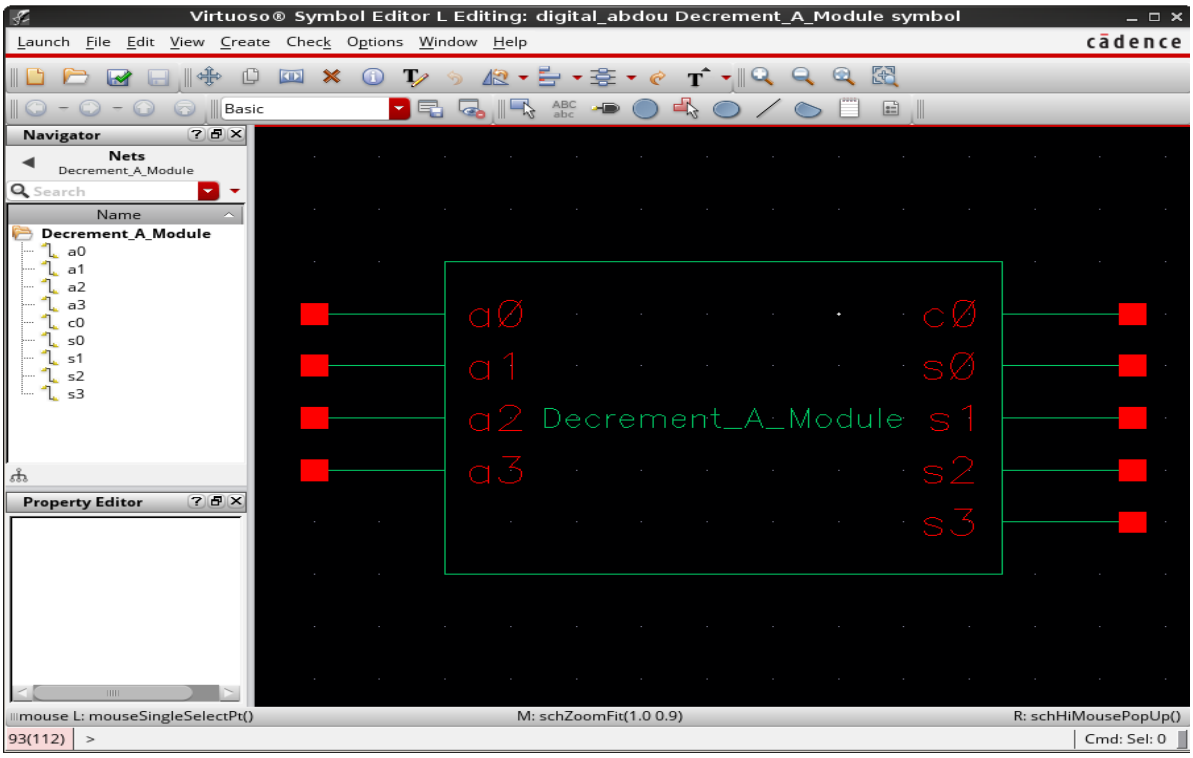
Symbol



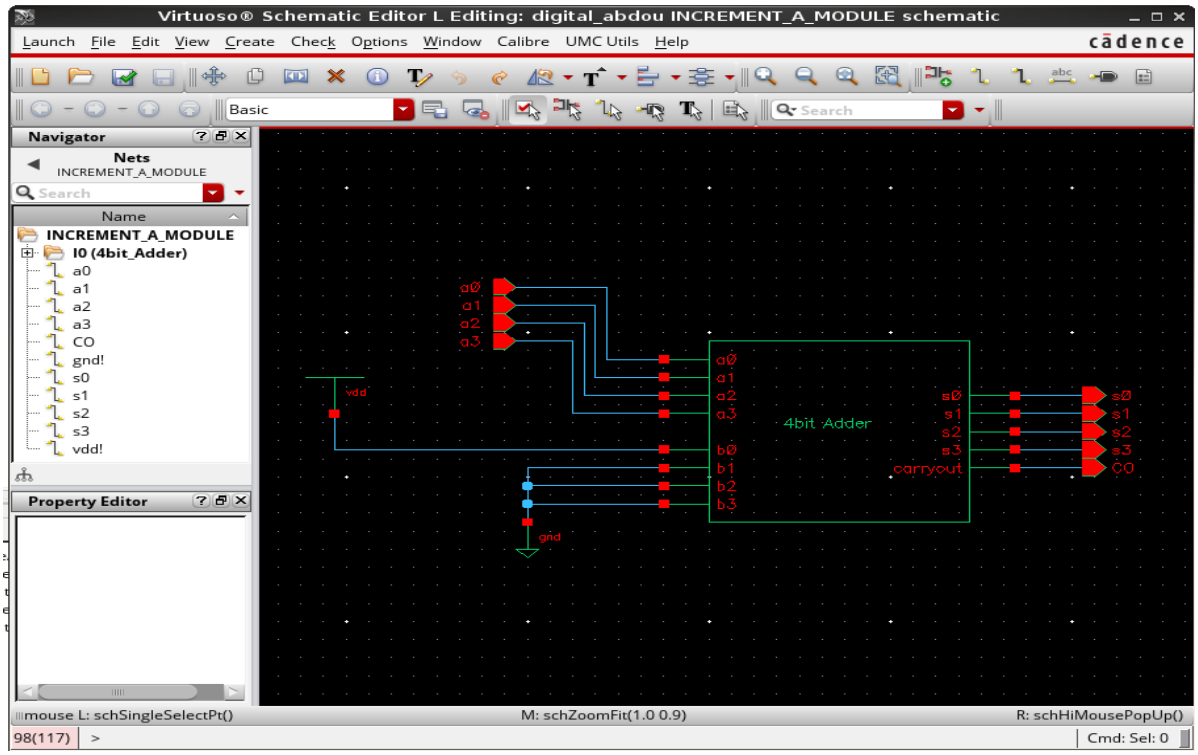
Decrement A-Module



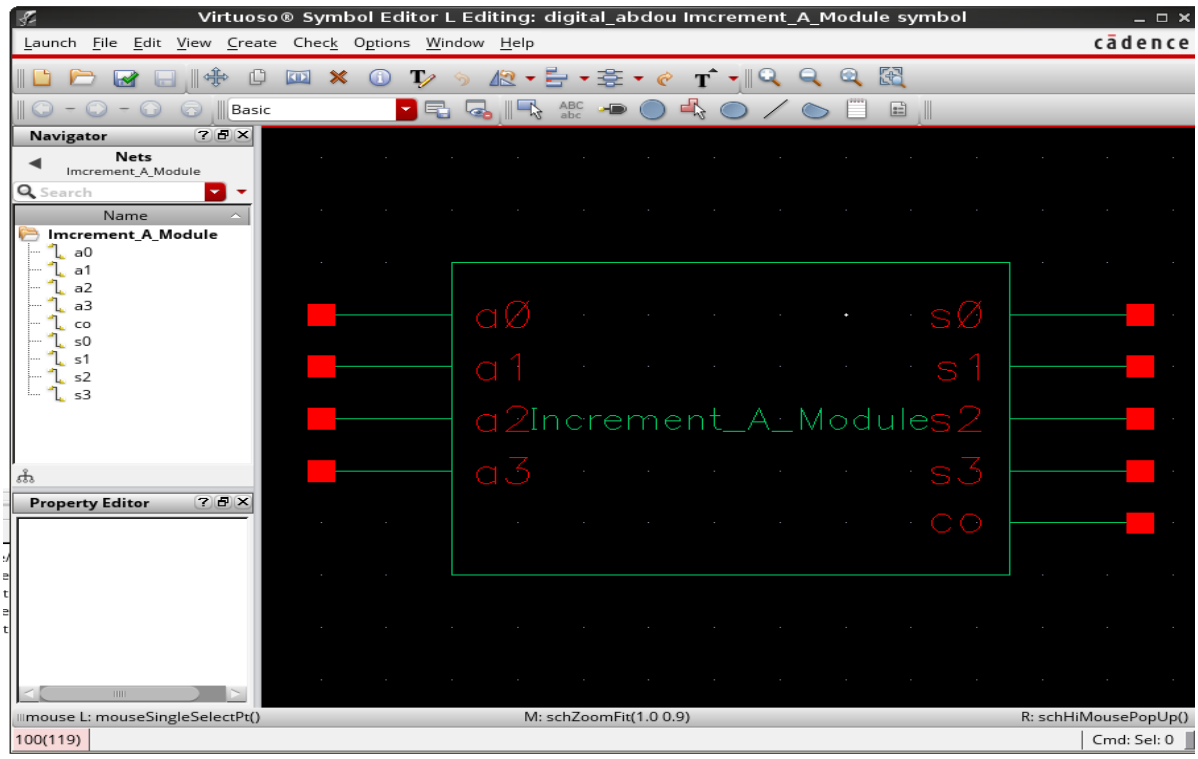
Symbol



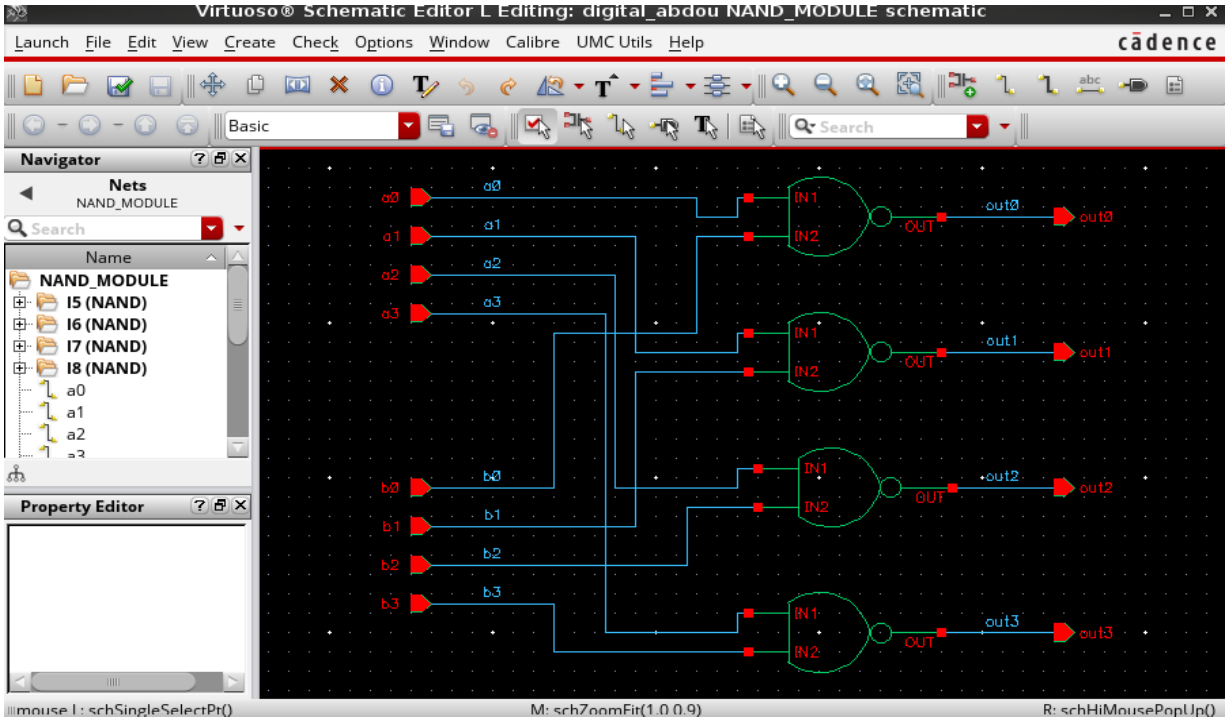
Increment A-Module



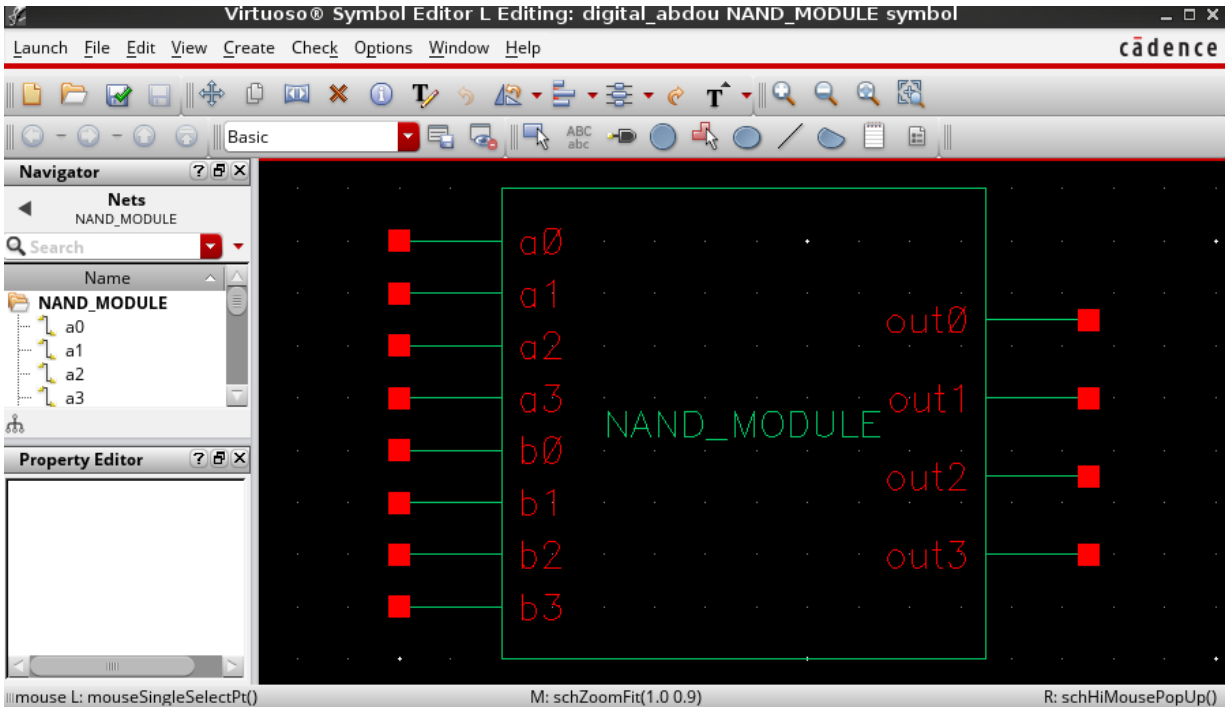
Symbol



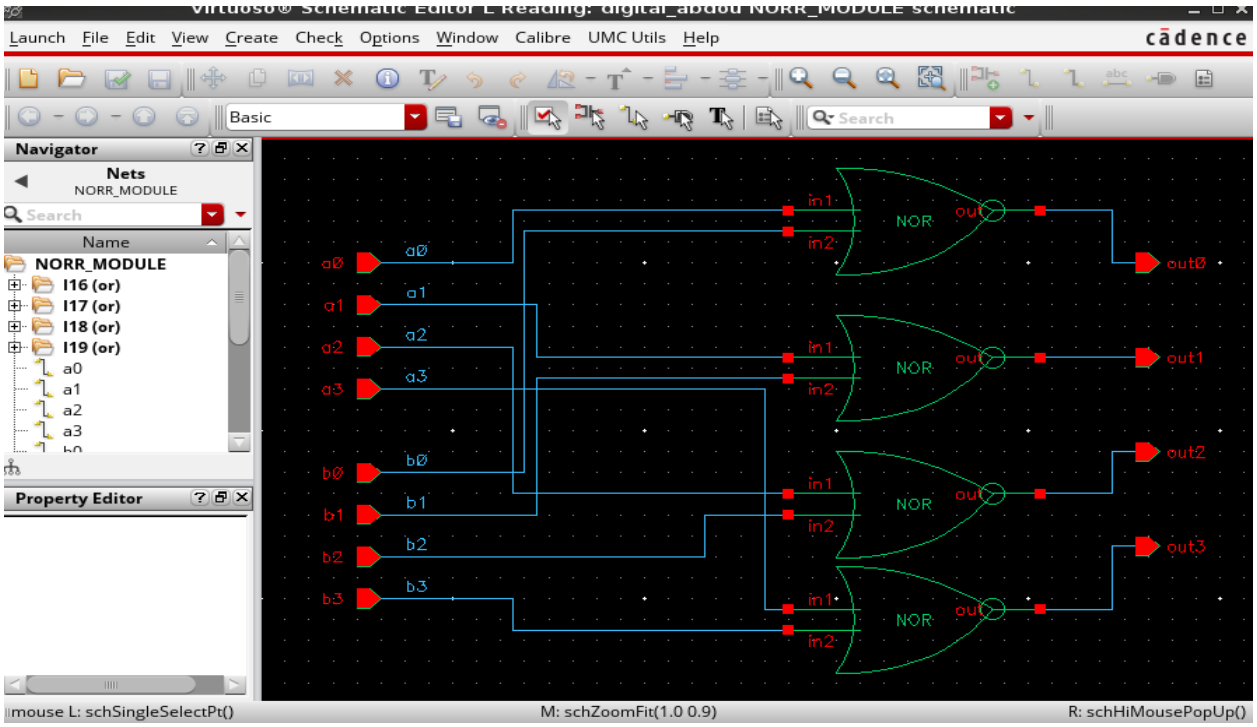
NAND Module



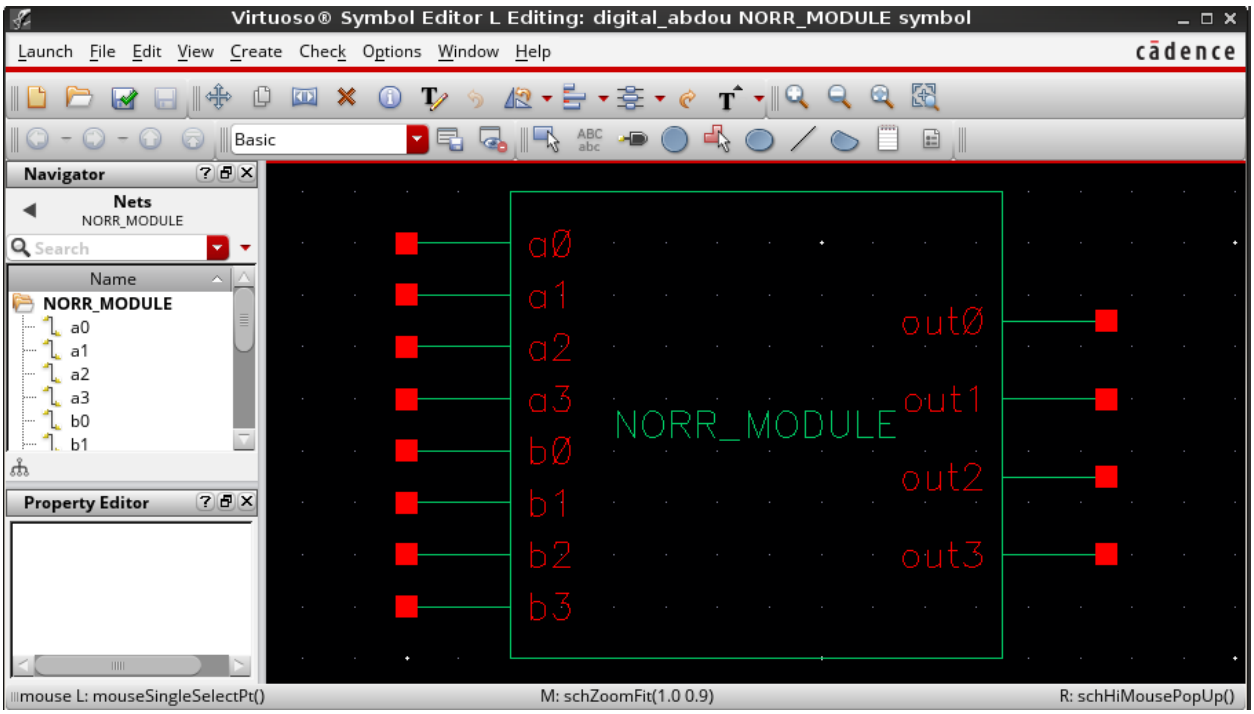
symbol



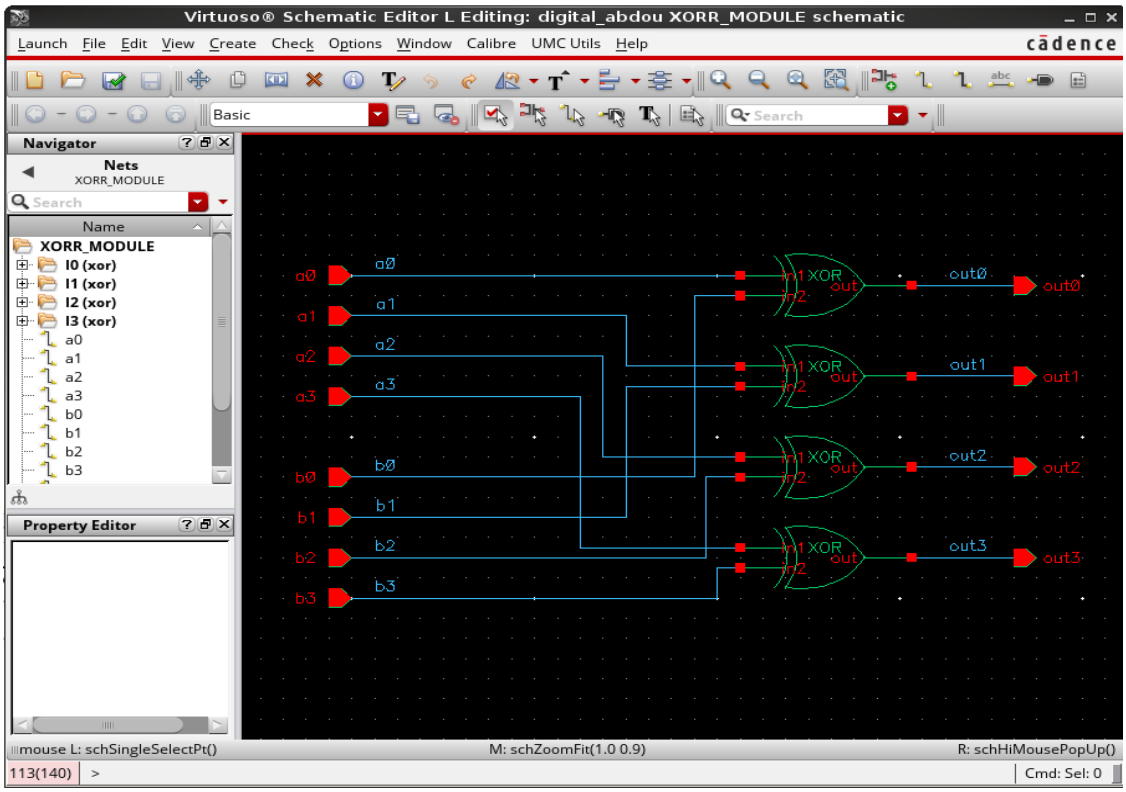
NOR Module



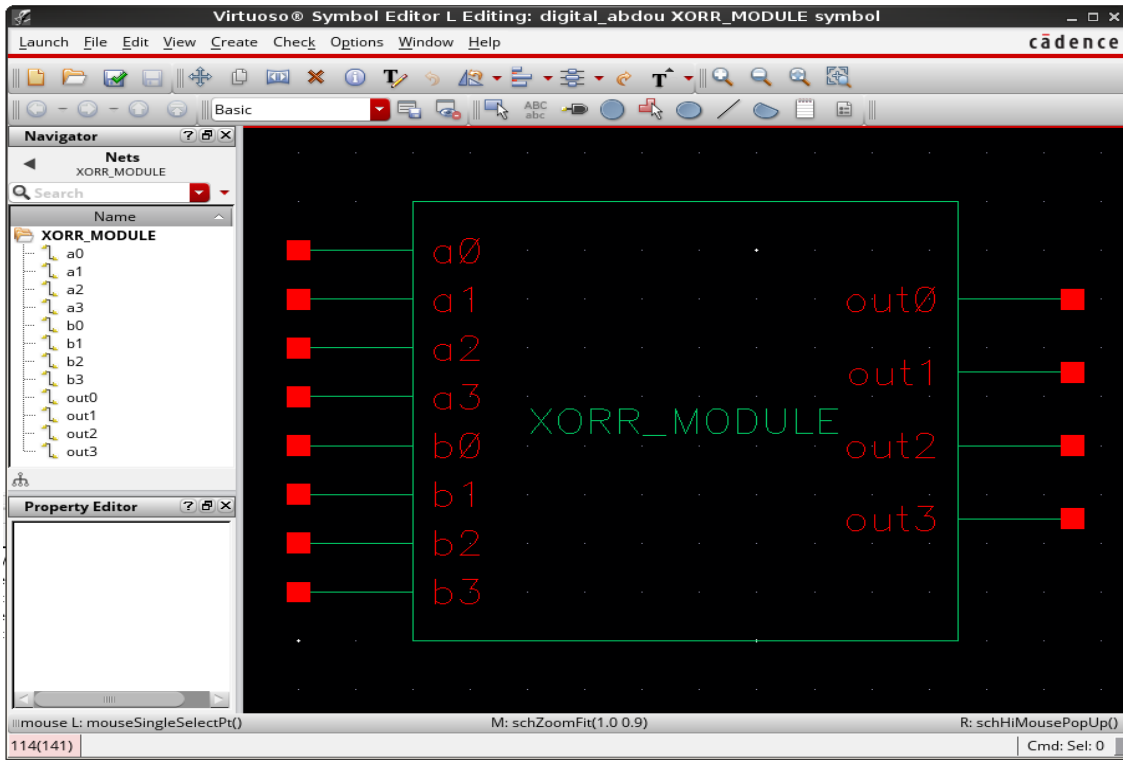
symbol



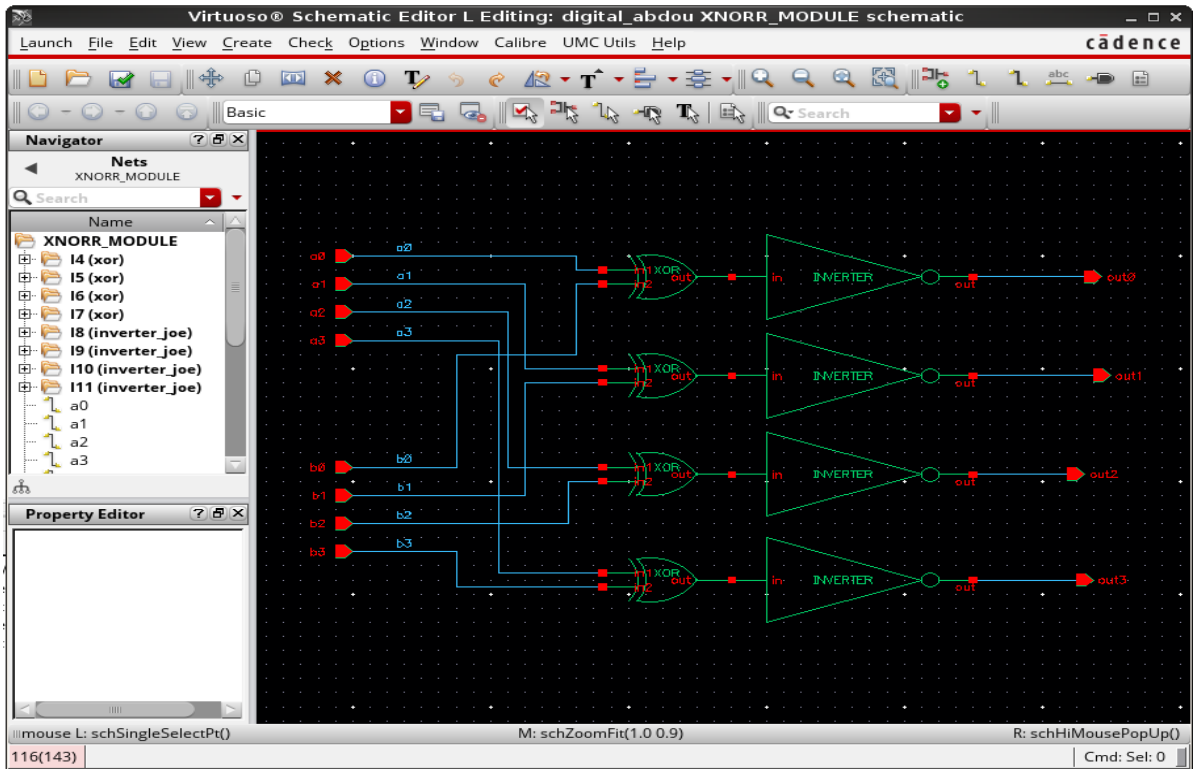
XOR Module



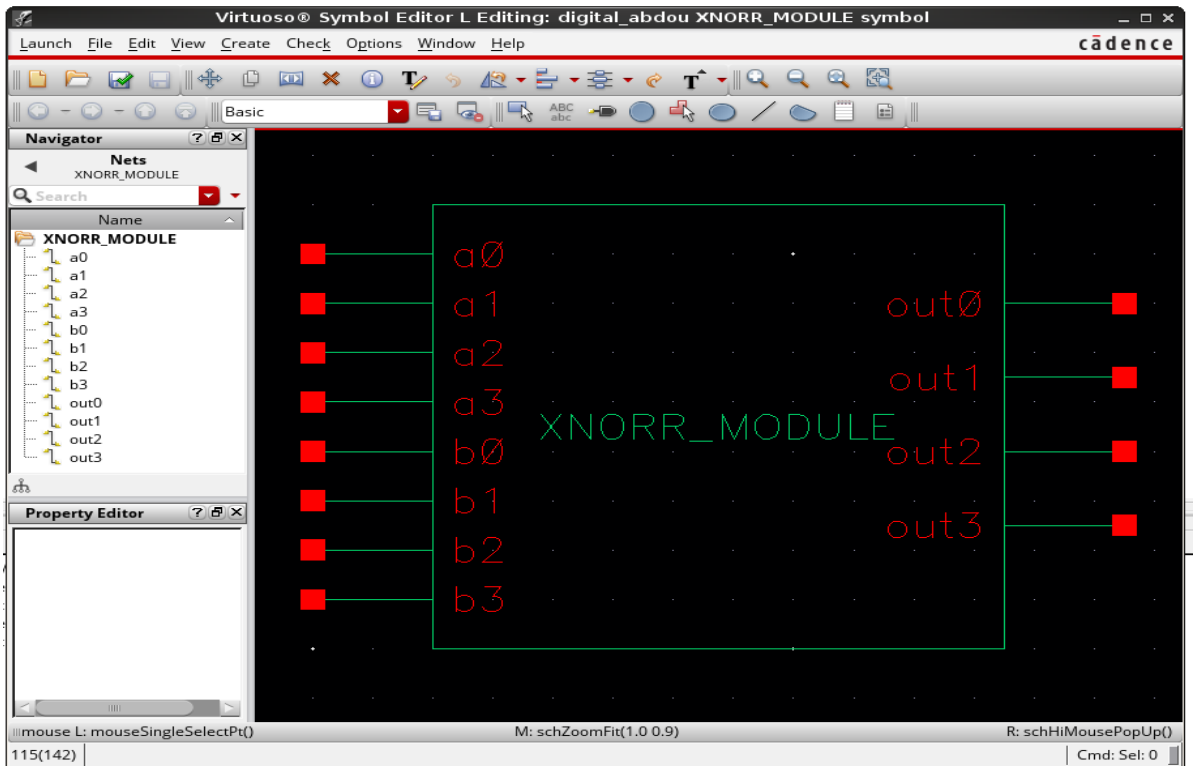
Symbol



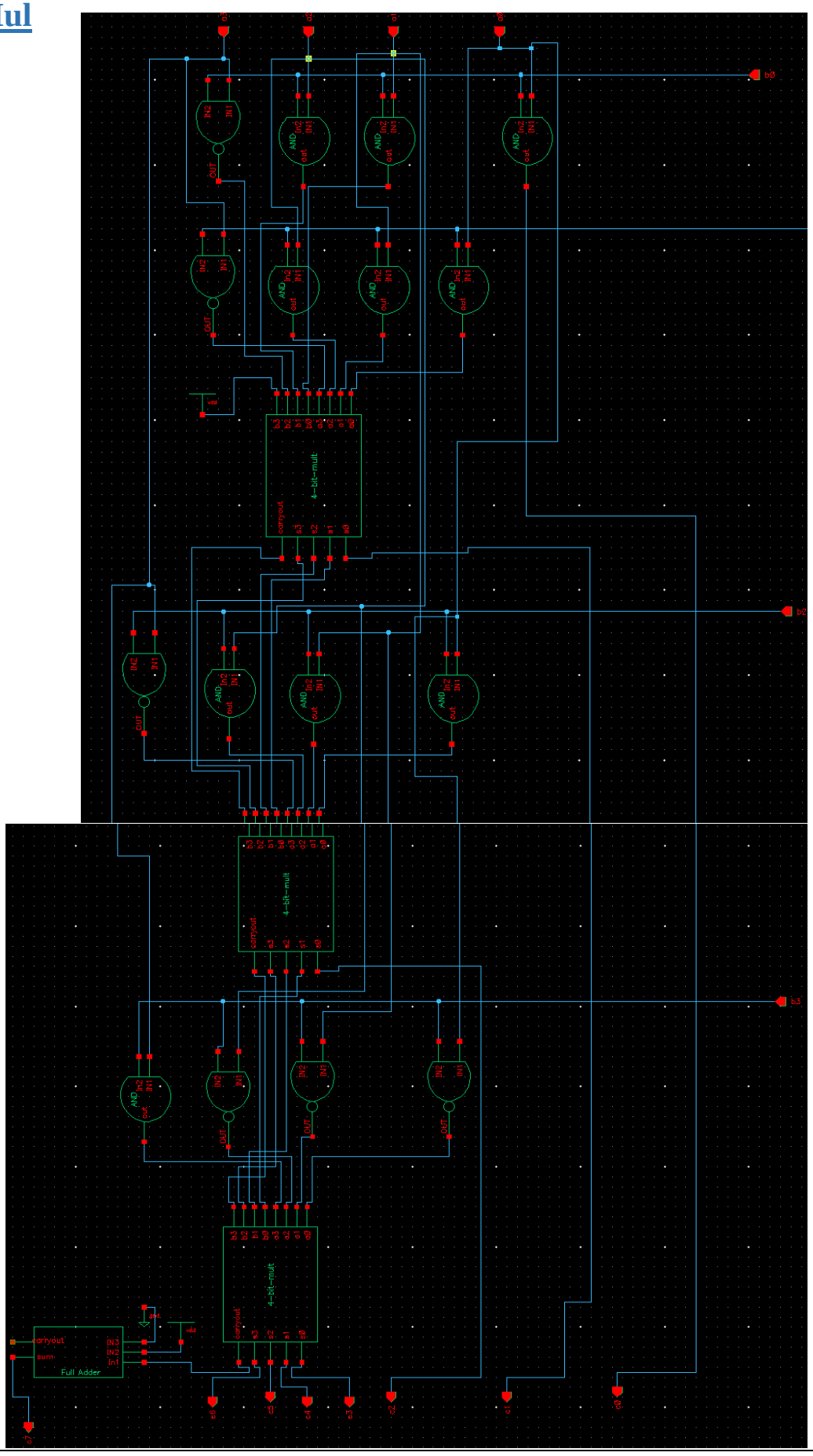
XNOR Module



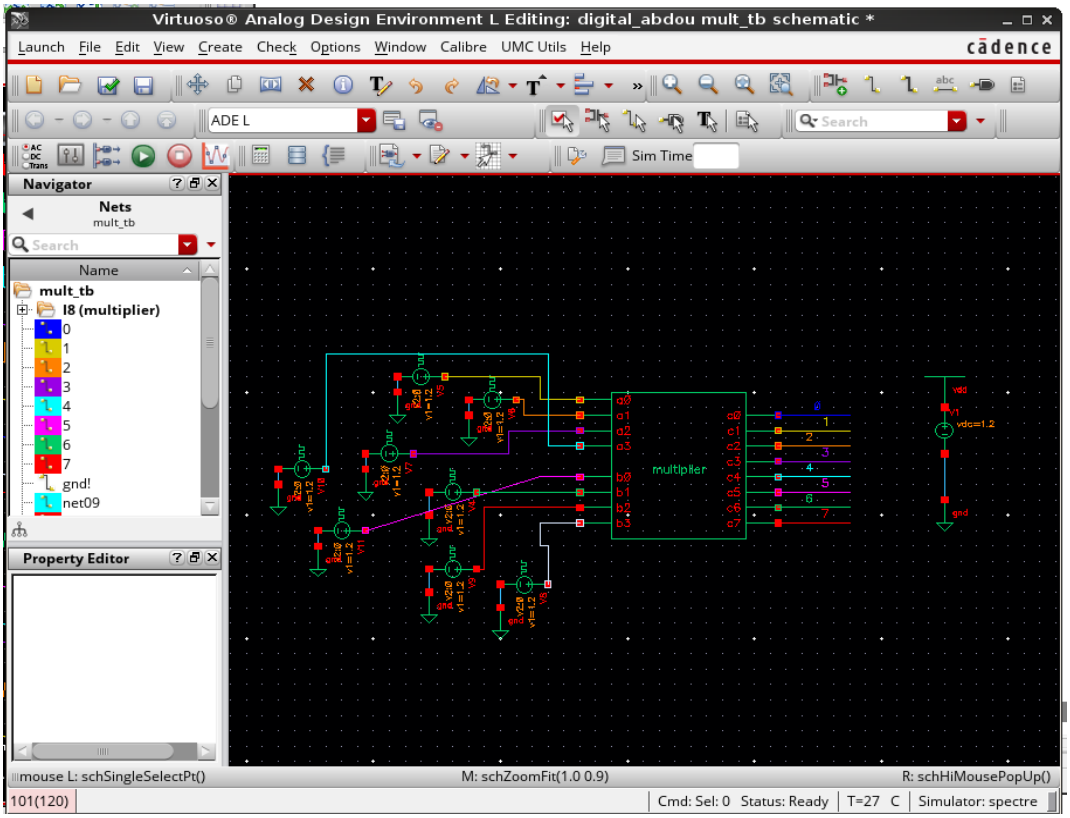
Symbol

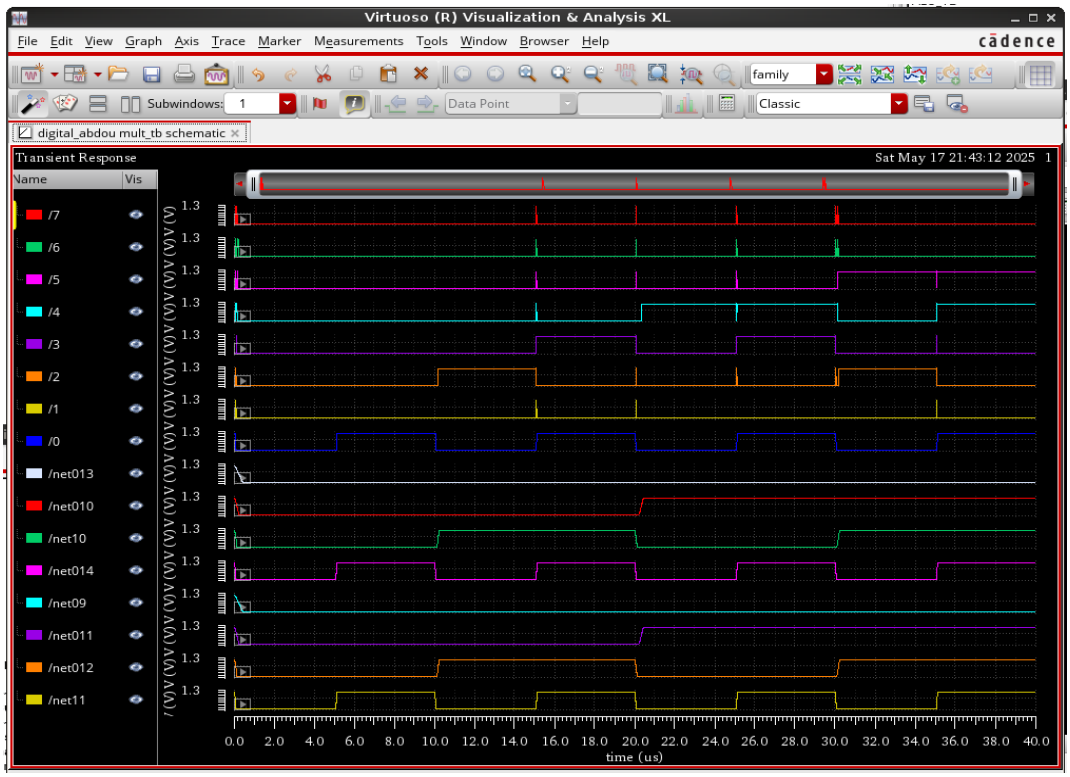


Mul

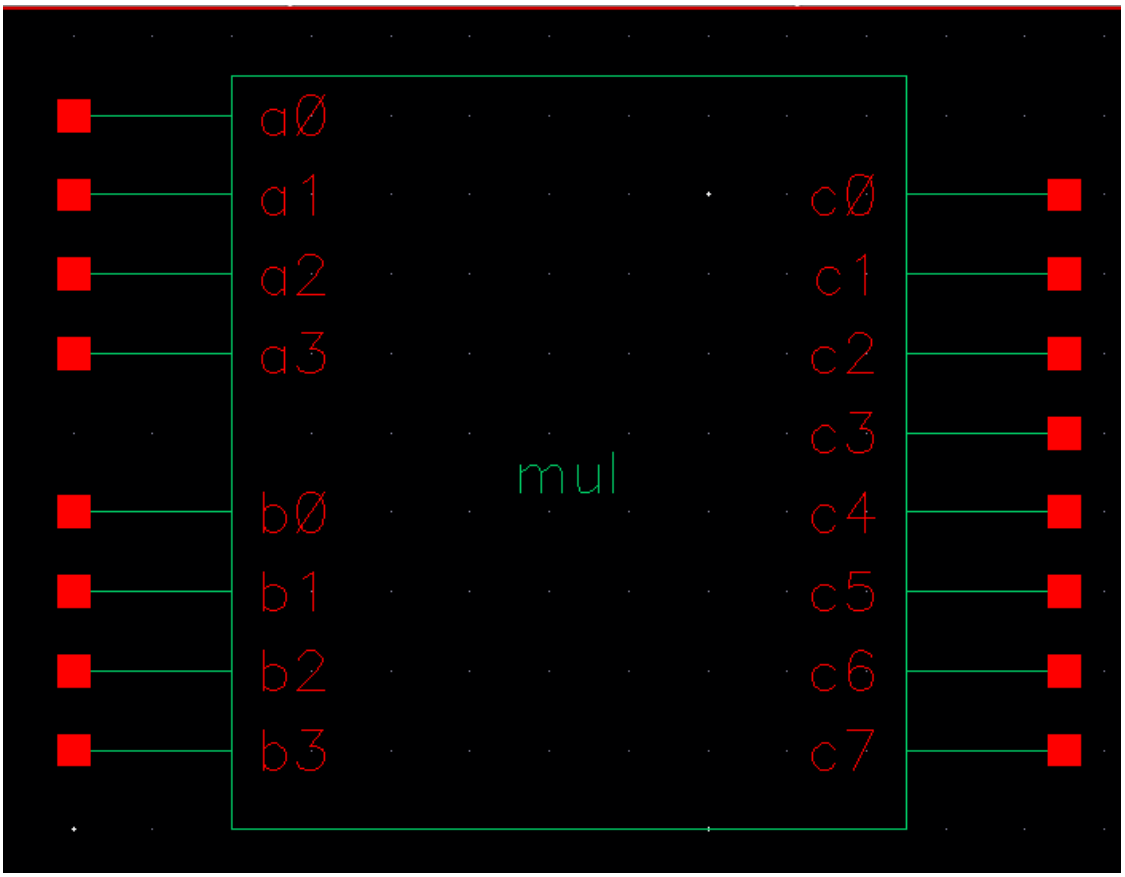


TEST-Bench

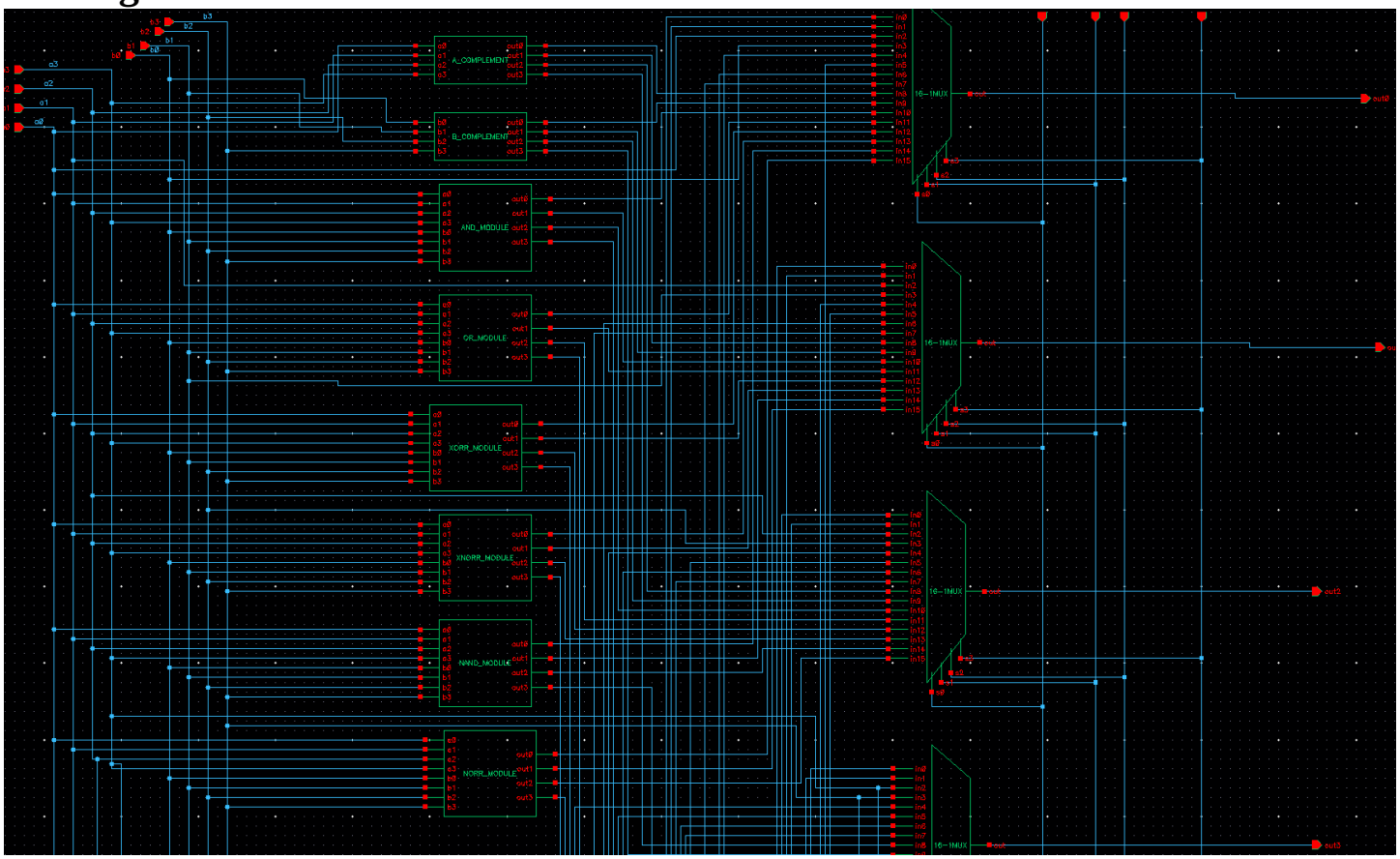

Virtuoso® Analog Design Environment L Editing: digital_abdou mult_tb schematic *
Launch File Edit View Create Check Options Window Calibre UMC Utils Help
cadence
ADE L
Sim Time
Navigator
Nets
mult_tb
Search
Name
mult_tb
I8 (multiplier)
0
1
2
3
4
5
6
7
gnd!
net09
Property Editor
multiplier
vdd
vdc=1.2
gnd
mouse L: schSingleSelectPt()
M: schZoomFit(1.0 0.9)
R: schHiMousePopUp()
101(120)
Cmd: Sel: 0 Status: Ready
T=27 C
Simulator: spectre


Virtuoso (R) Visualization & Analysis XL
File Edit View Graph Axis Trace Marker Measurements Tools Window Browser Help
cadence
family
Subwindows: 1
Data Point
Classic
digital_abdou mult_tb schematic
Transient Response
Sat May 17 21:43:12 2025
Name
Vis
/7
/6
/5
/4
/3
/2
/1
/0
/net013
/net010
/net10
/net014
/net09
/net011
/net012
/net11
1.3
0.0 2.0 4.0 6.0 8.0 10.0 12.0 14.0 16.0 18.0 20.0 22.0 24.0 26.0 28.0 30.0 32.0 34.0 36.0 38.0 40.0
time (us)

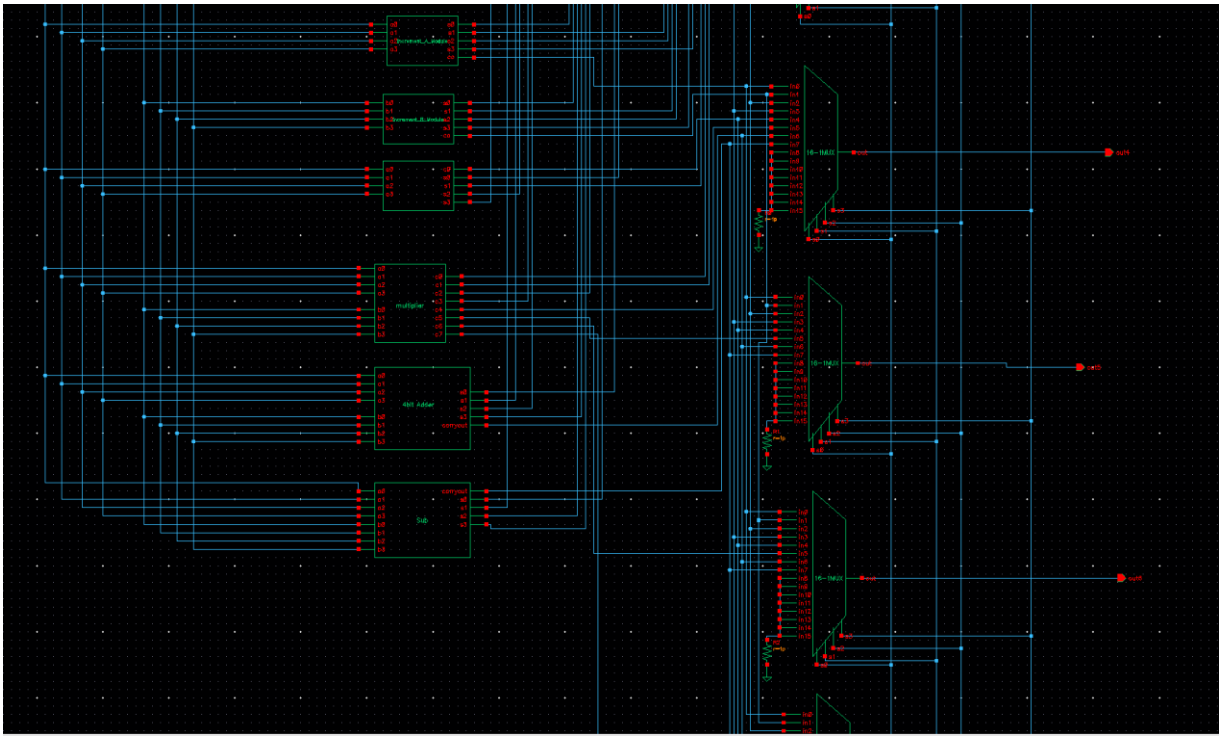
Mul Symbol



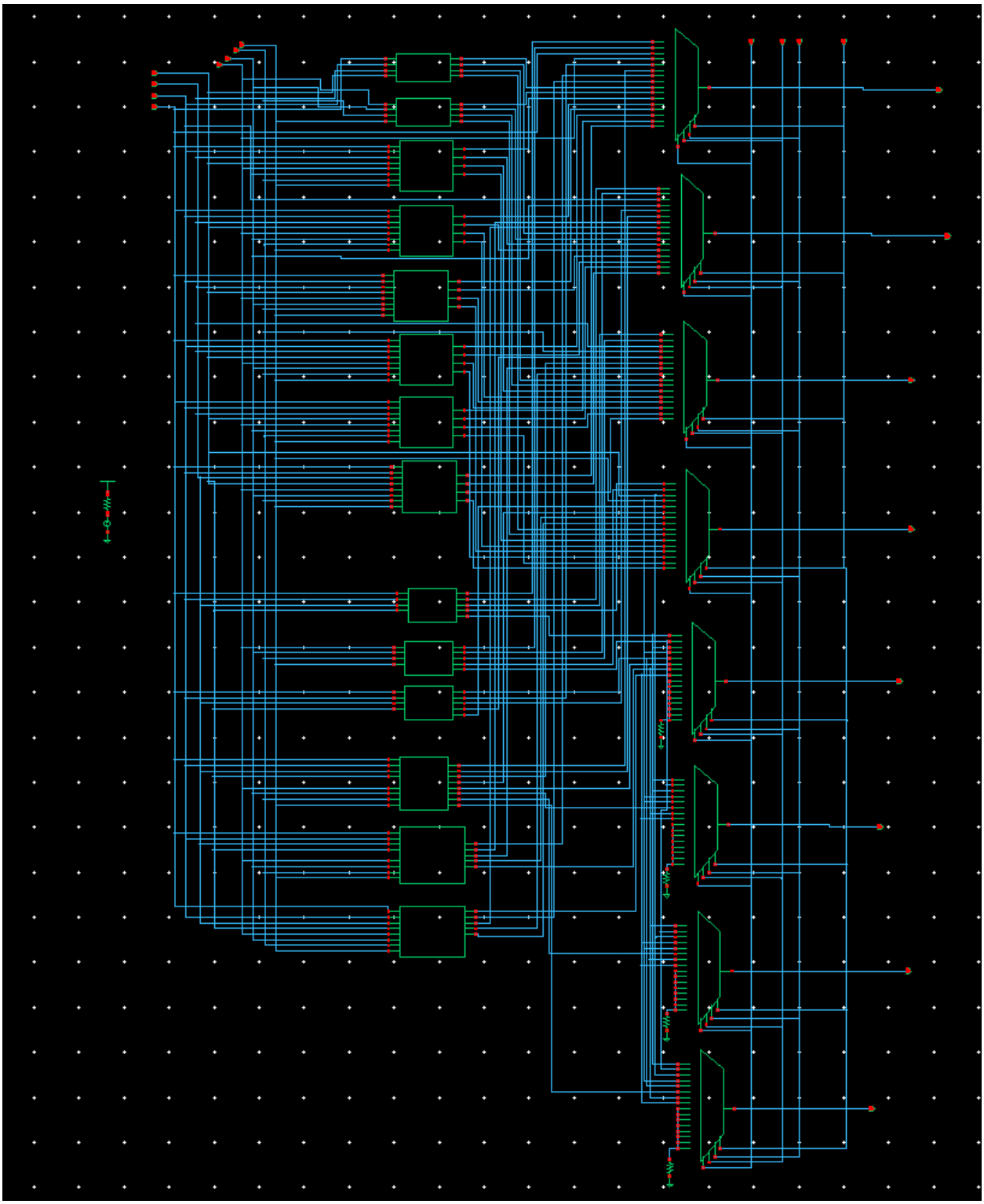
Logical unit of the ALU

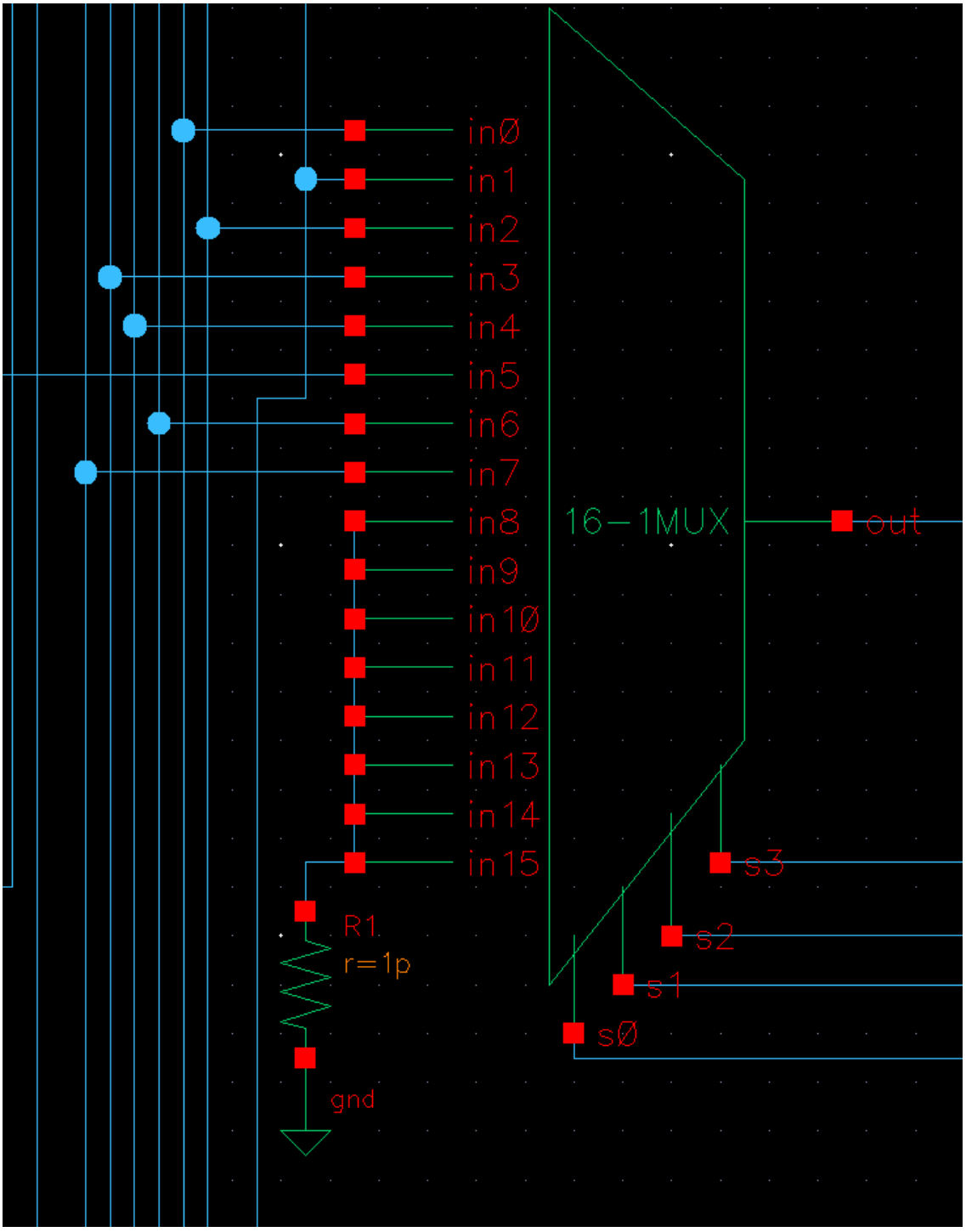


Arethmetic unit of the ALU



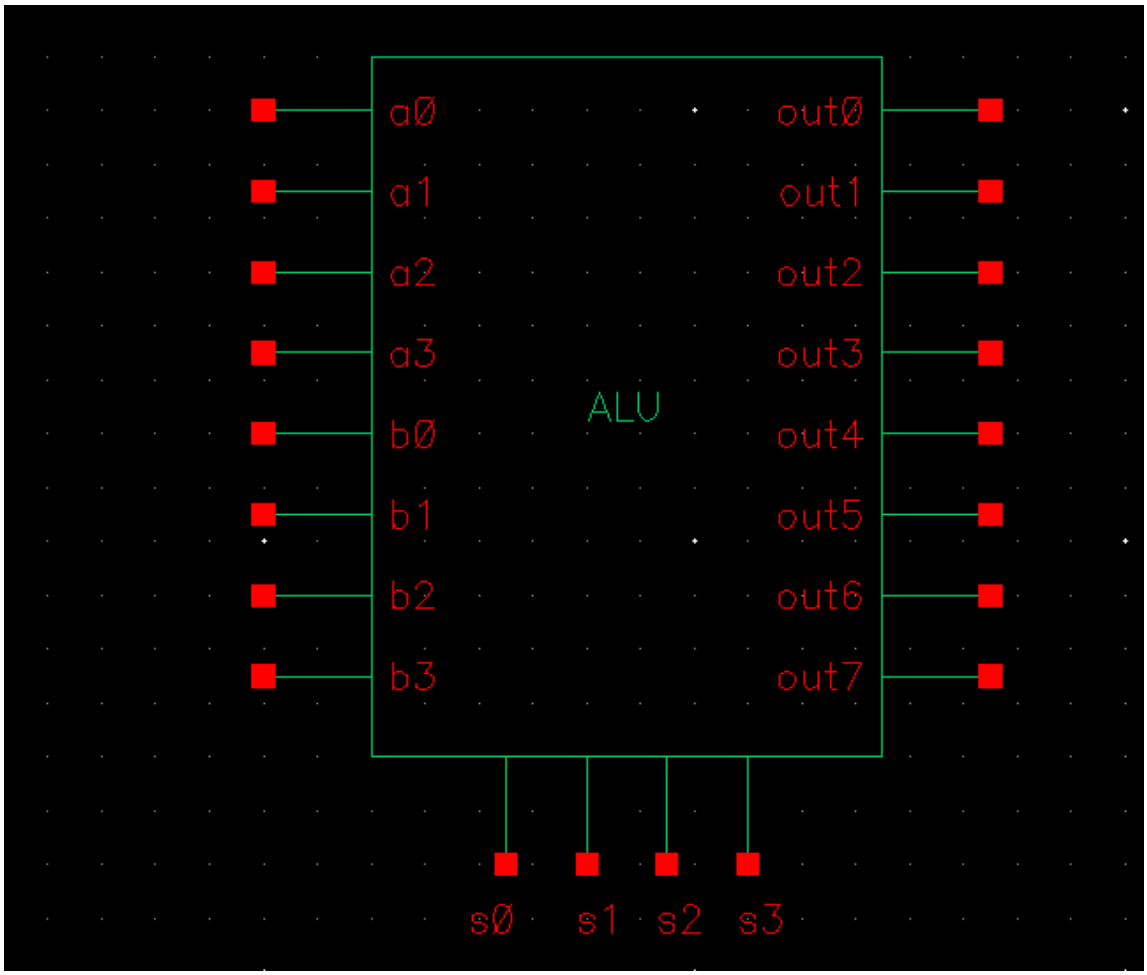
ALU





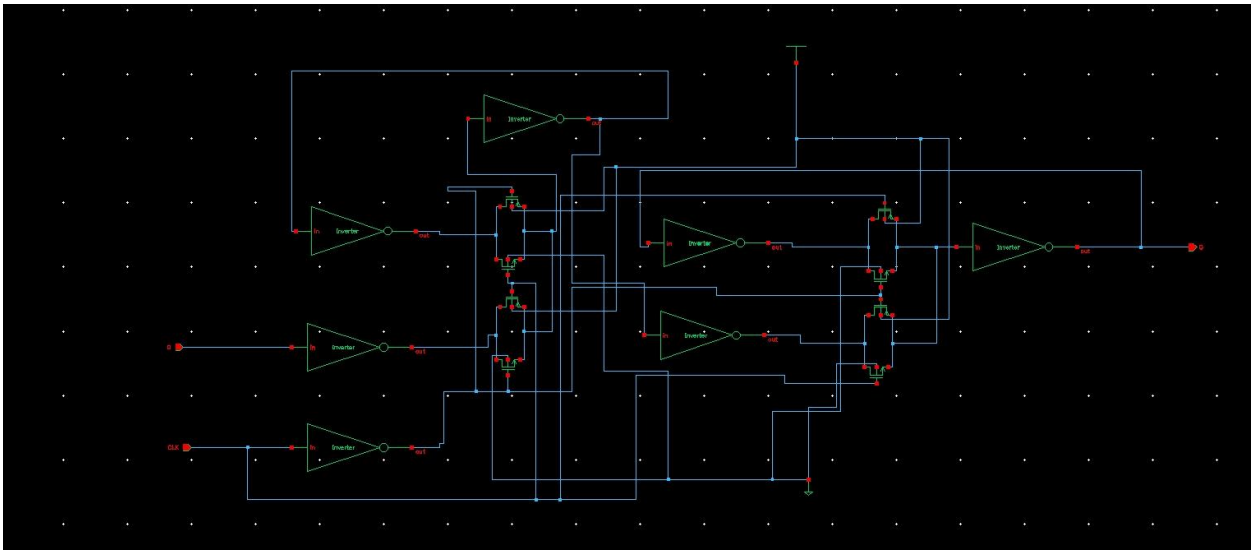
The inputs on the first four muxes were all filled as from input 8-15 it receives the bits of the logical operation from the logical unit but we needed extra for muxes as the output from the multiplication needed 8 bits so we normalized all outputs to be 8-bits and we made logical extension to the logical output and sign extended the last 3 bits from the addition and subtraction output

Alu Symbol


a0
a1
a2
a3
b0
b1
b2
b3
ALU
out0
out1
out2
out3
out4
out5
out6
out7
s0
s1
s2
s3

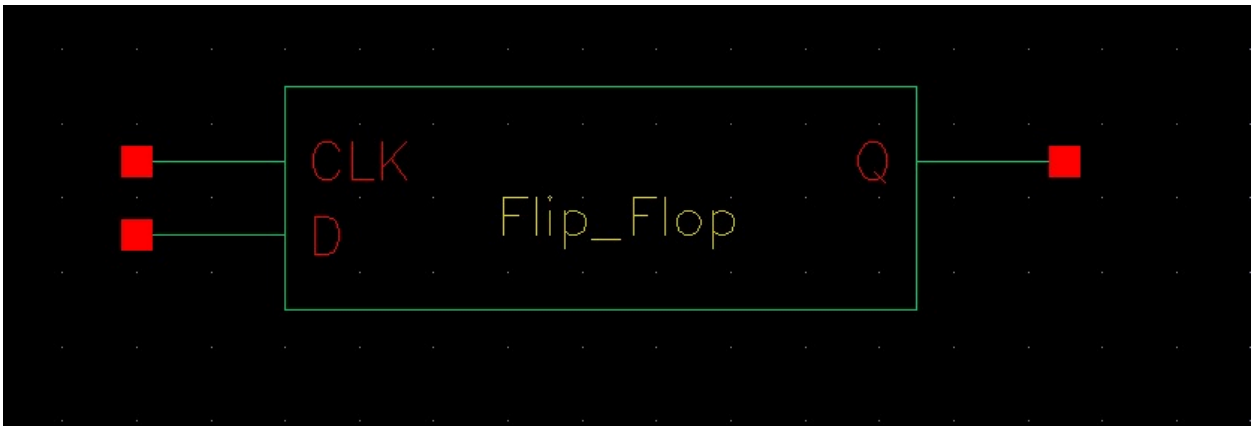
D-Q flip flop

schematic

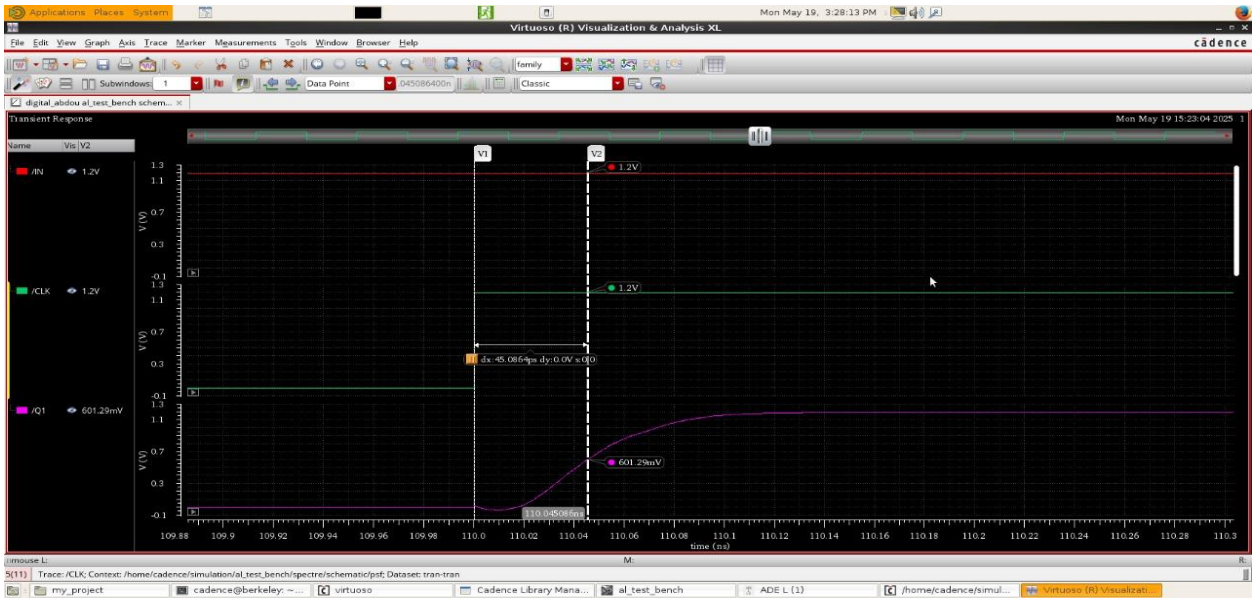


We used mux based positive edge trigger(master-slave)

Symbol



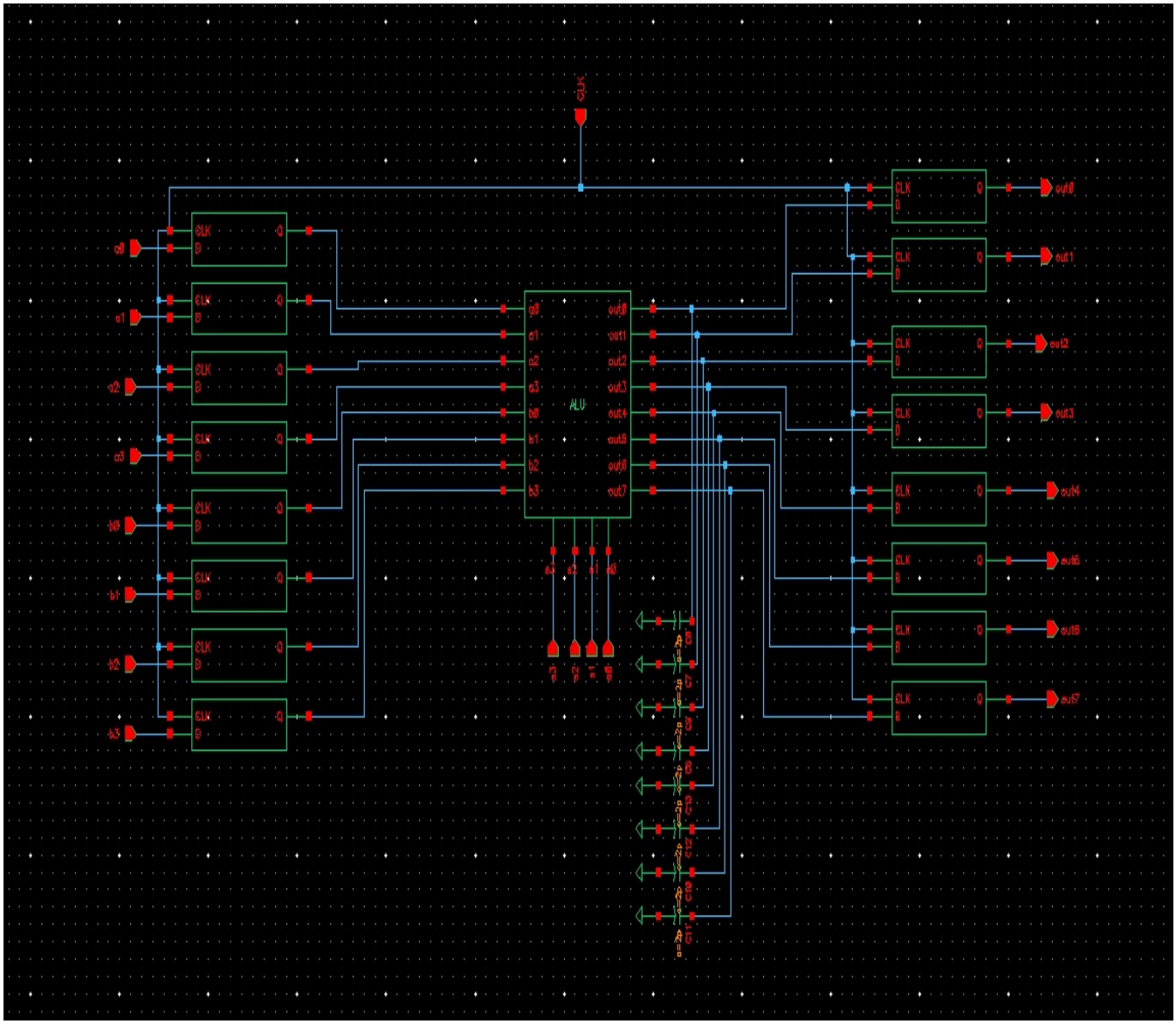
D-Q flip flop-TB



When the input (D)=1 and the positive clock edge comes the (Q) turns High

When the input (D)=0 and the positive clock edge comes the (Q) turns low

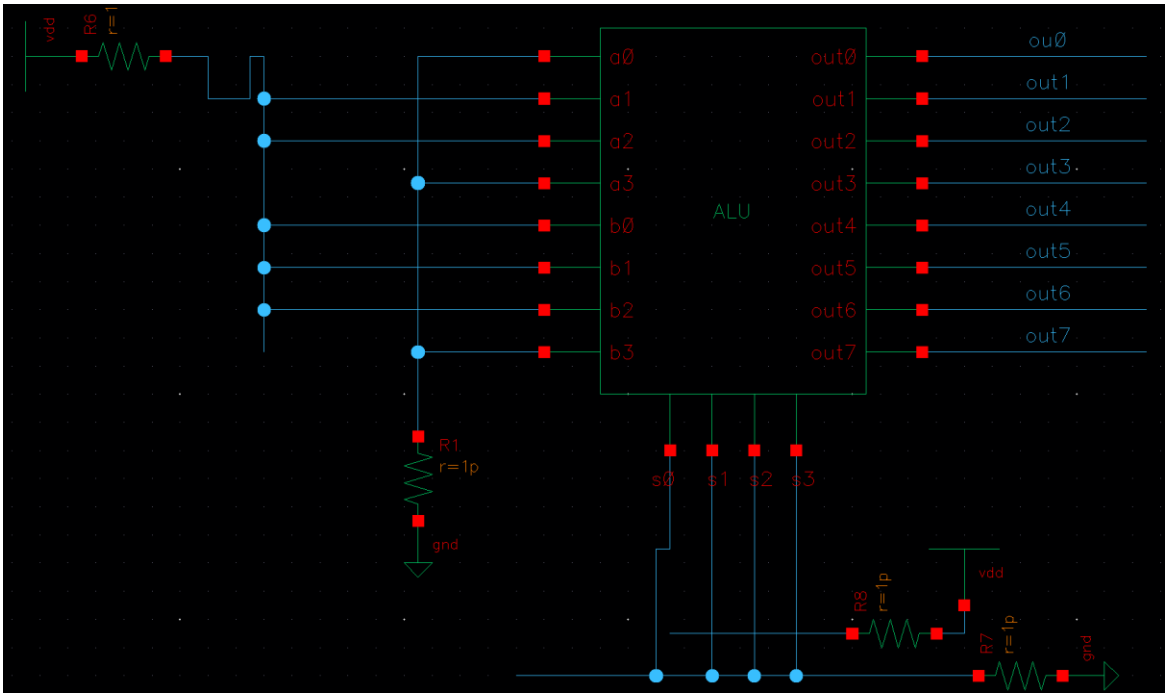
Final Alu with load capacitance and flipflop

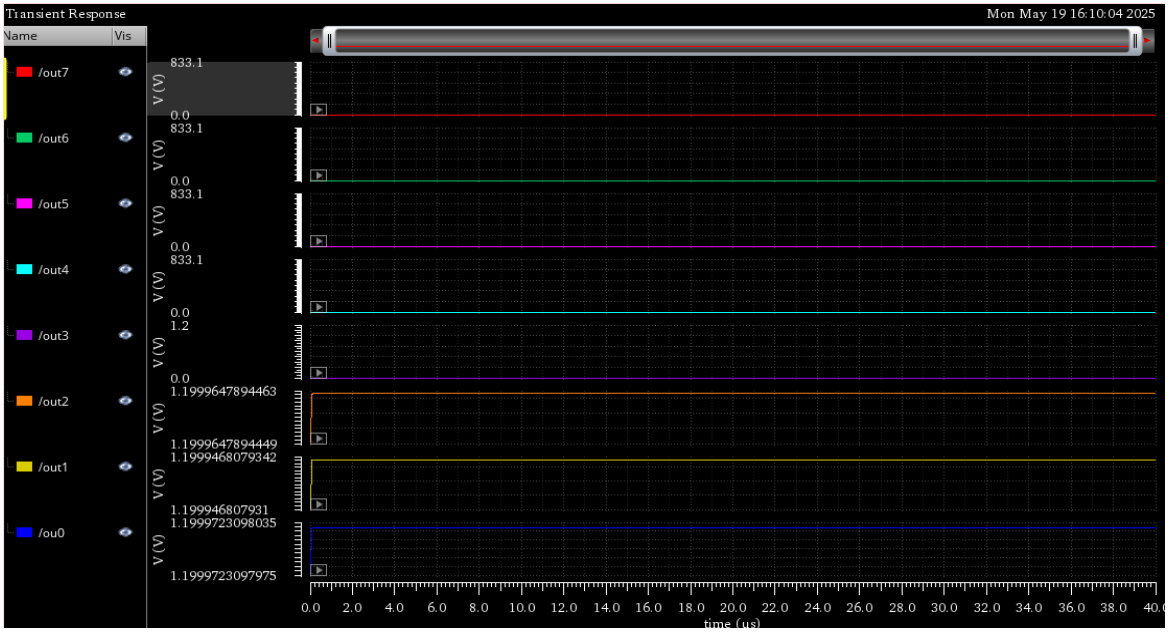


Alu operations

1-Increment A

Selection lines→0000 A→0110=6

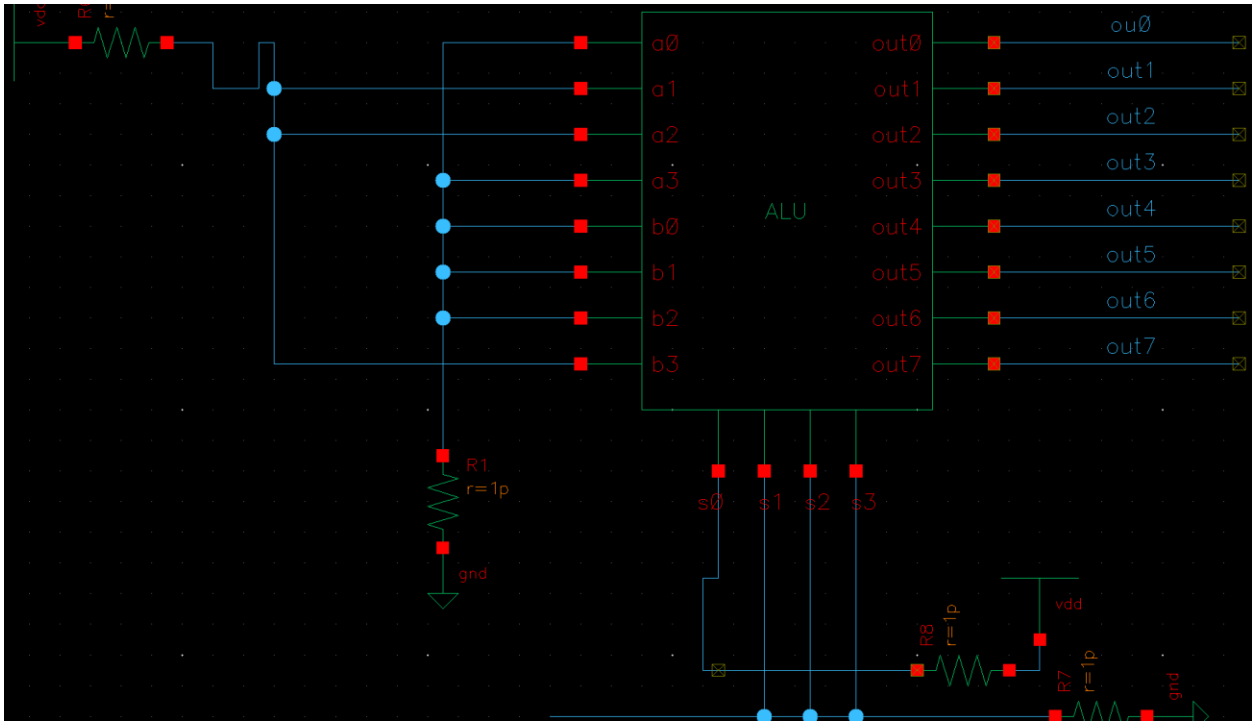


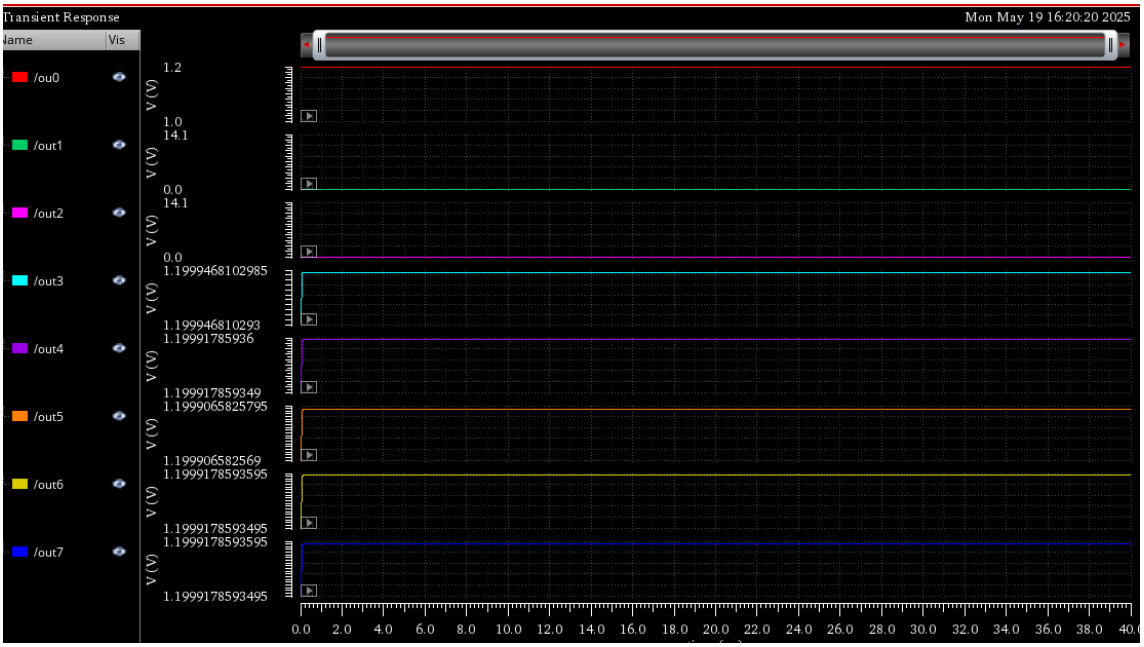


the output as shown→0000 0111=7

2-Increment B

Selection lines→0001 B→1000=-8

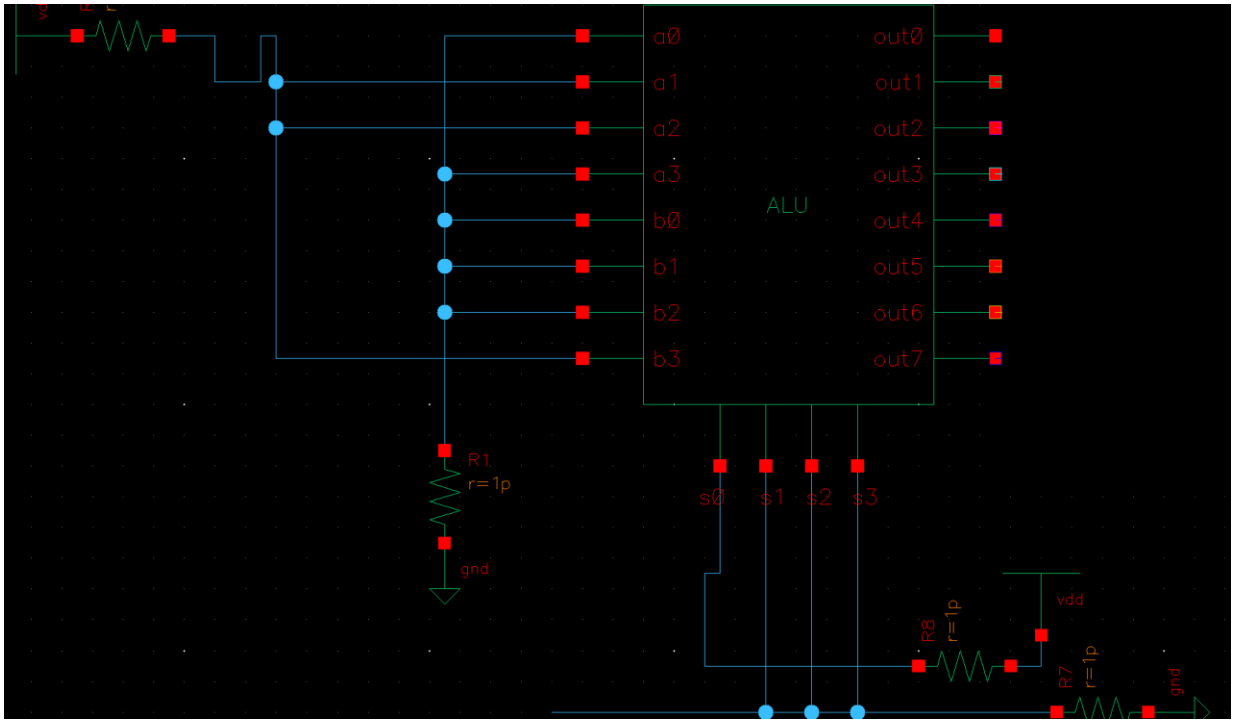


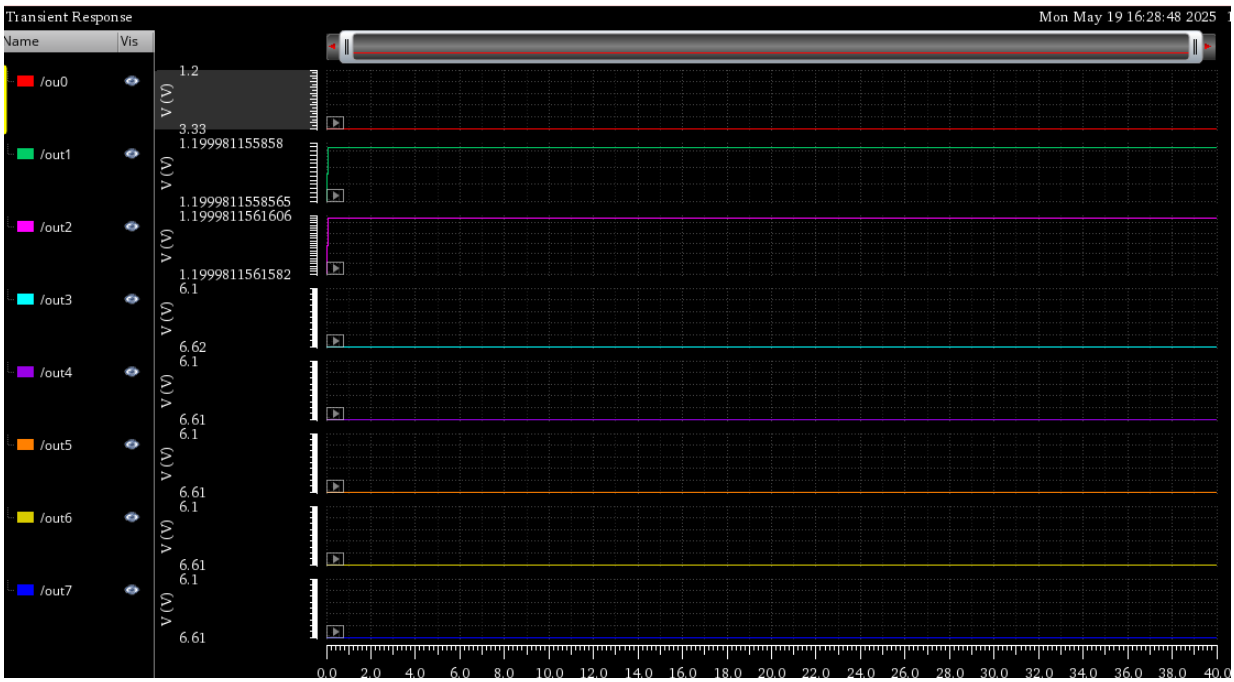


the output as shown→1111 1001=-7

3-Transfer A

Selection lines→0010 A→0110

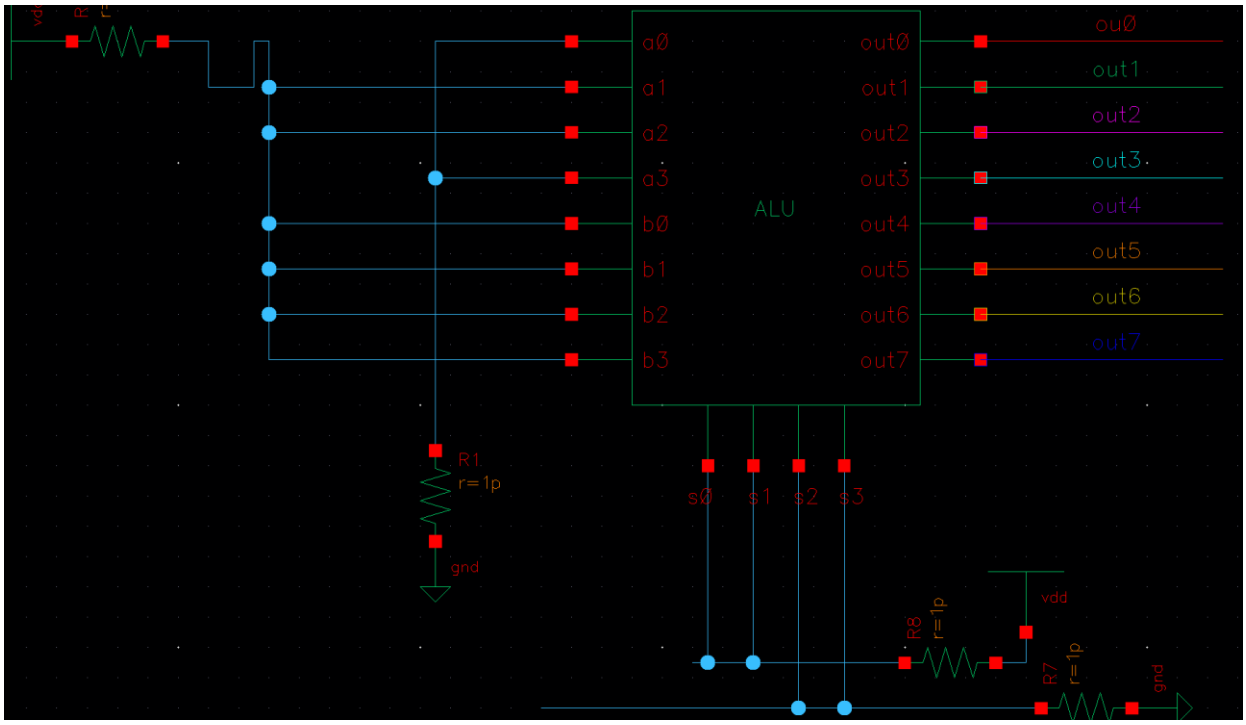


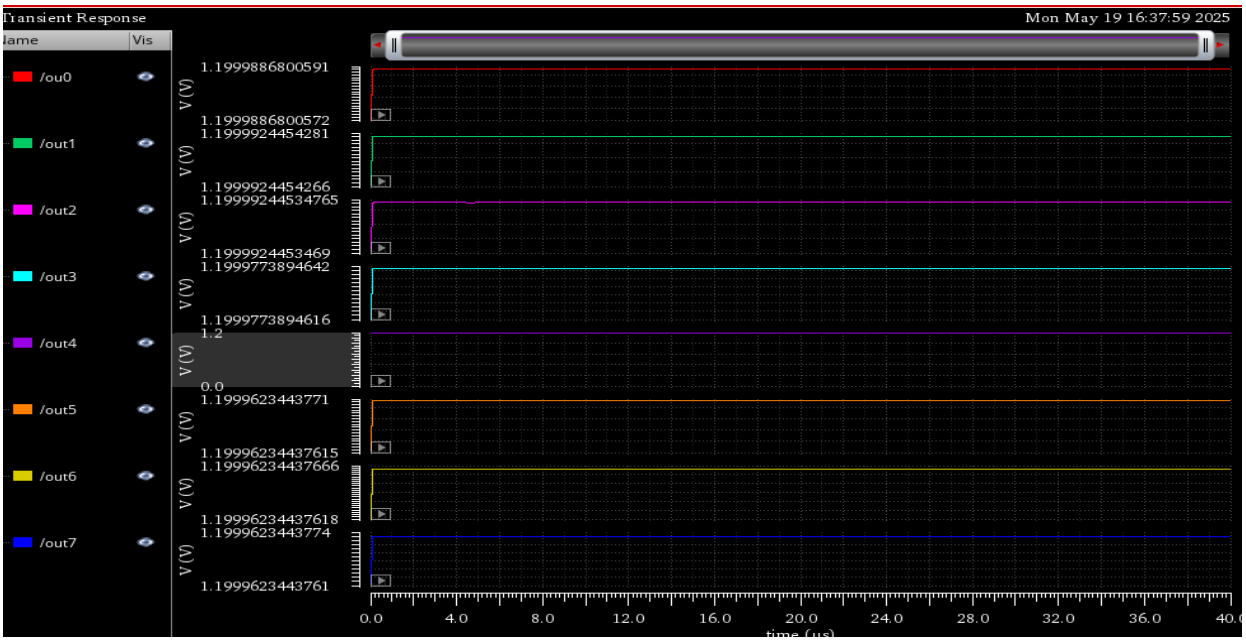


the output as shown→0000 0110=6

4-Transfer B

Selection lines→0011 B→1111=-1

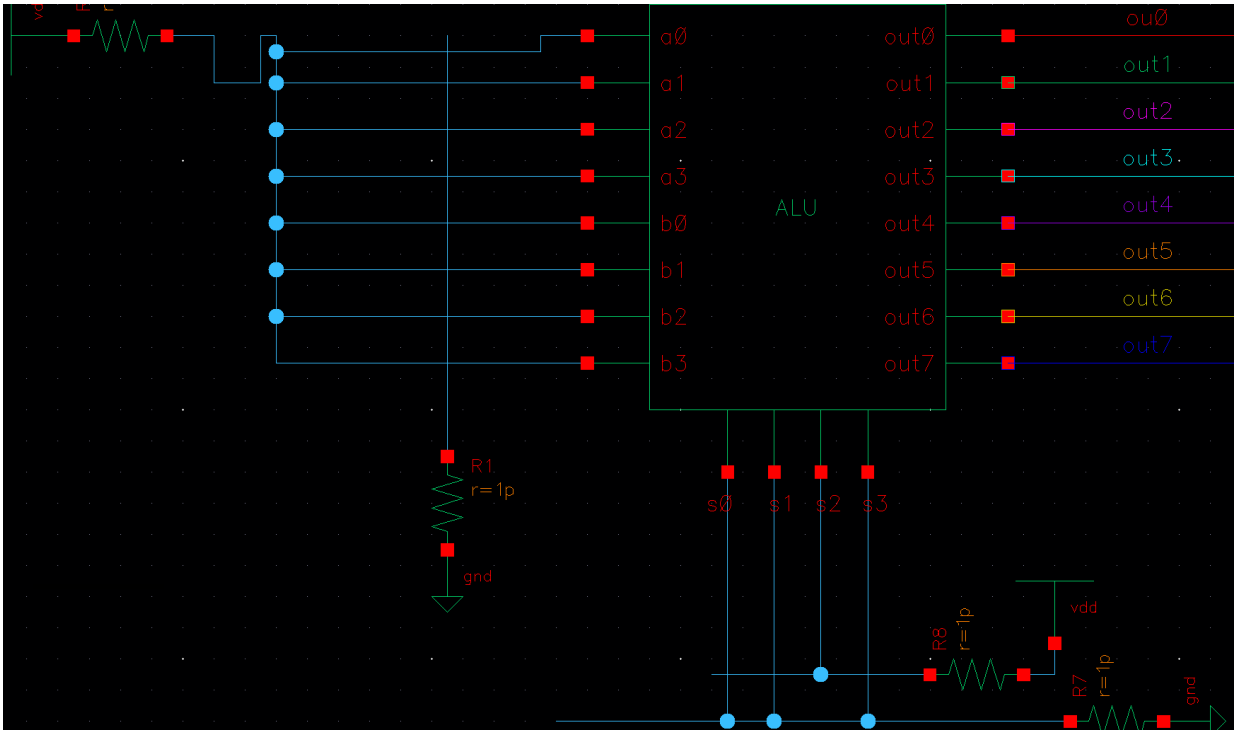


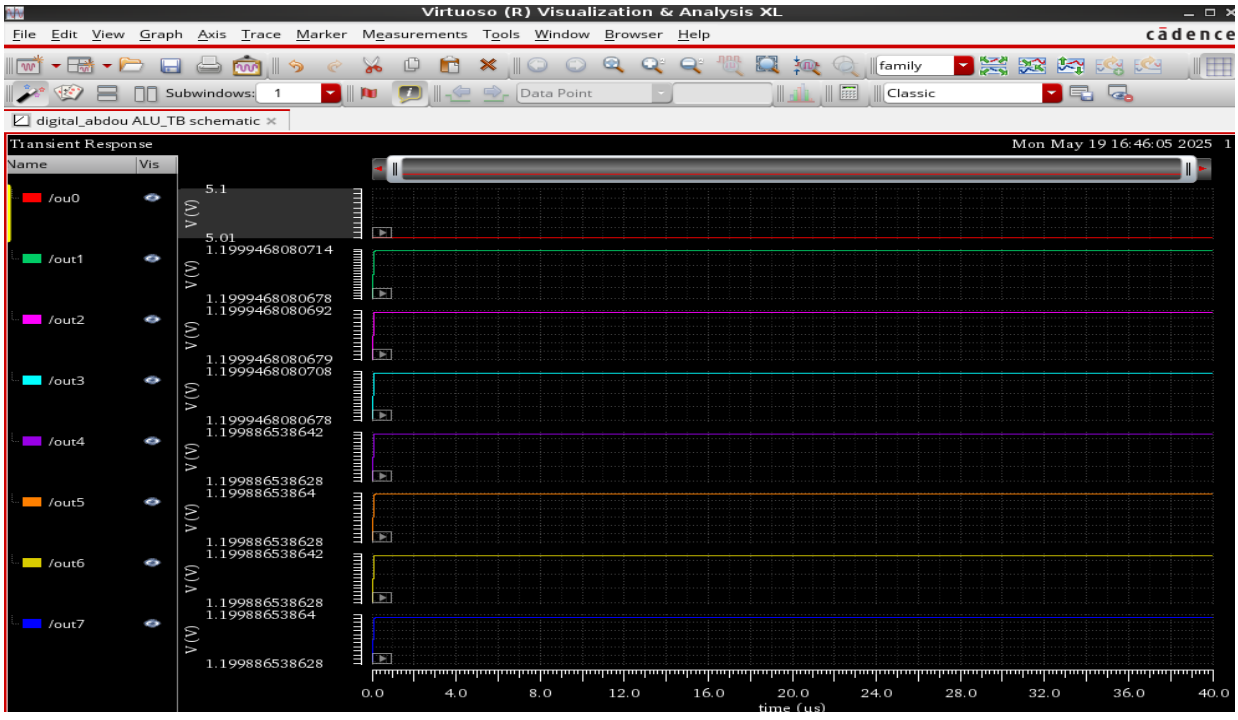


the output as shown→1111 1111=-1

5-Decrement A

Selection lines→0100 A→1111=-1

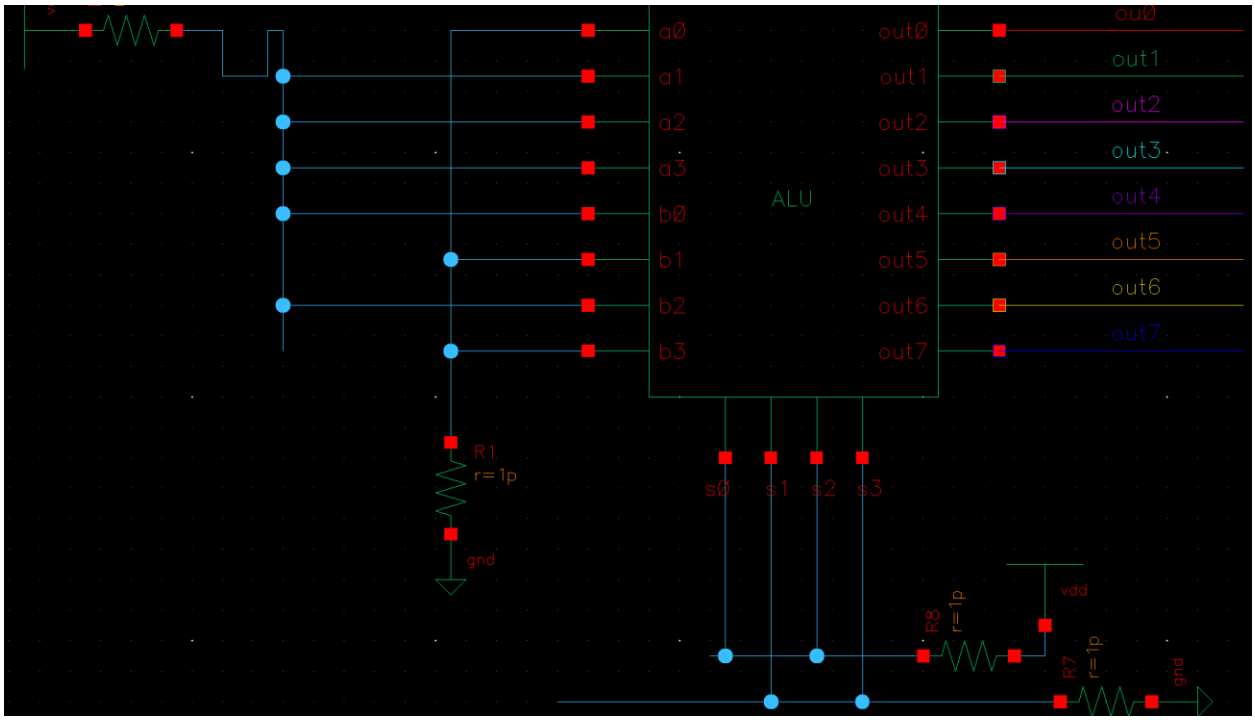


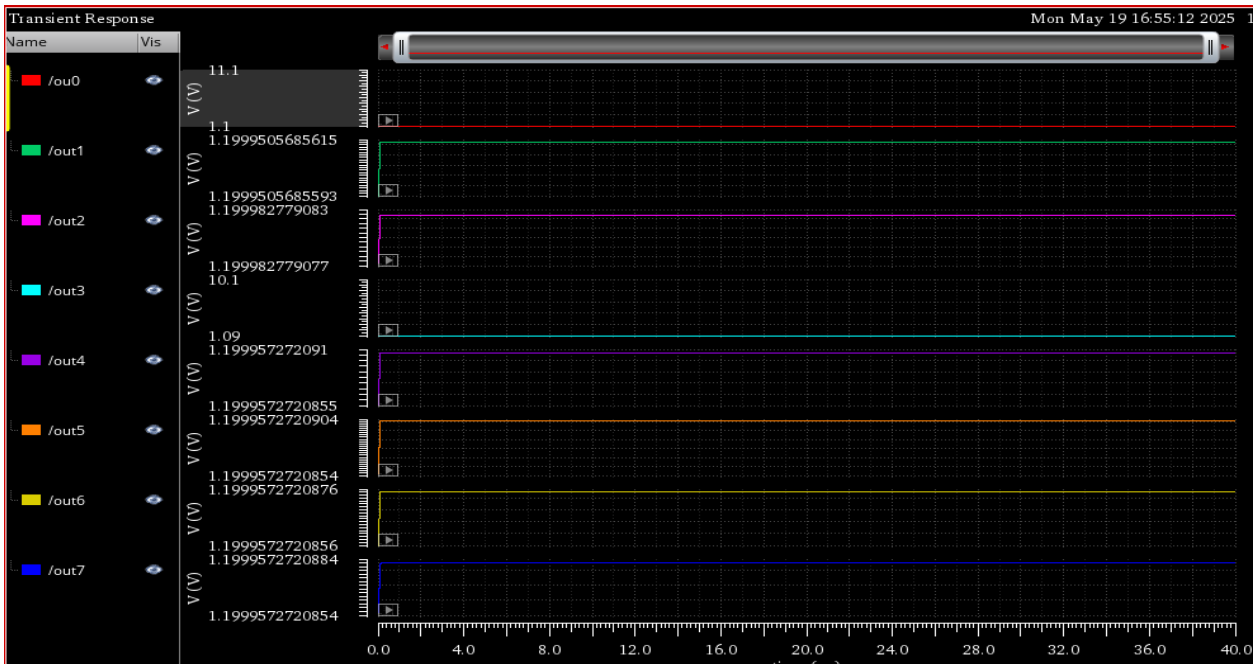


the output as shown→1111 1110=-2

6-Multiply

Selection lines→0101 A→1110=-2 B→0101=5

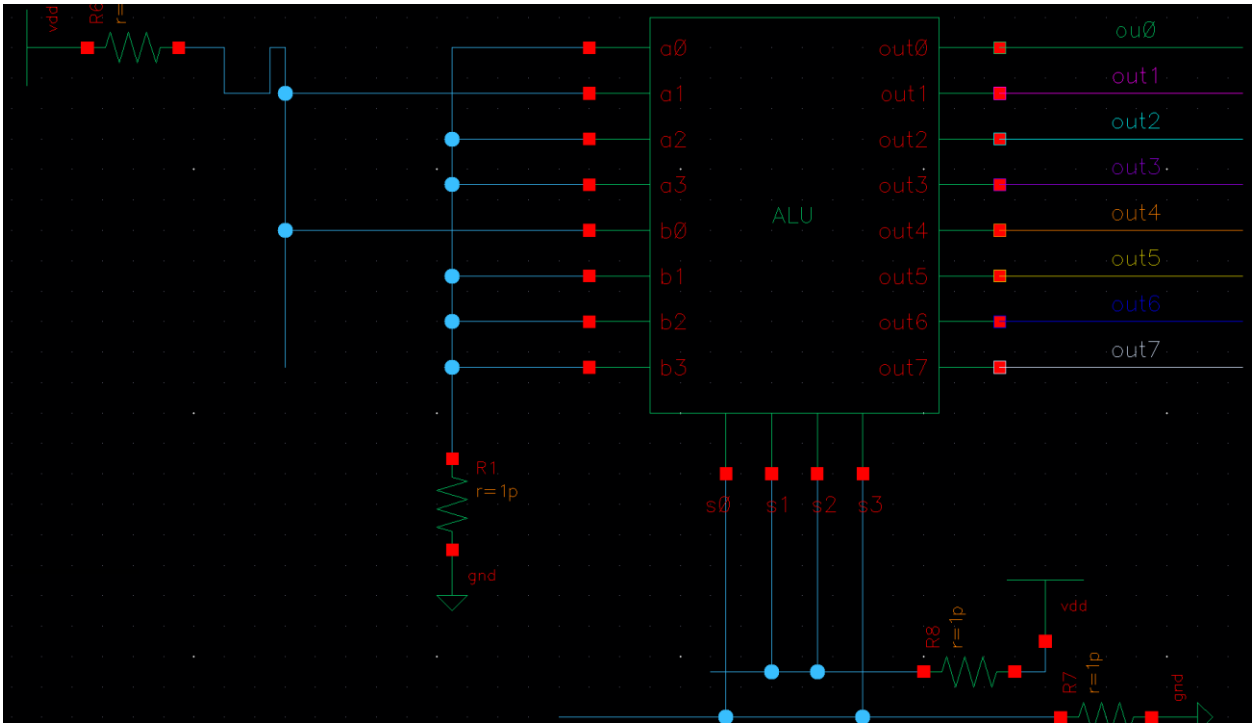


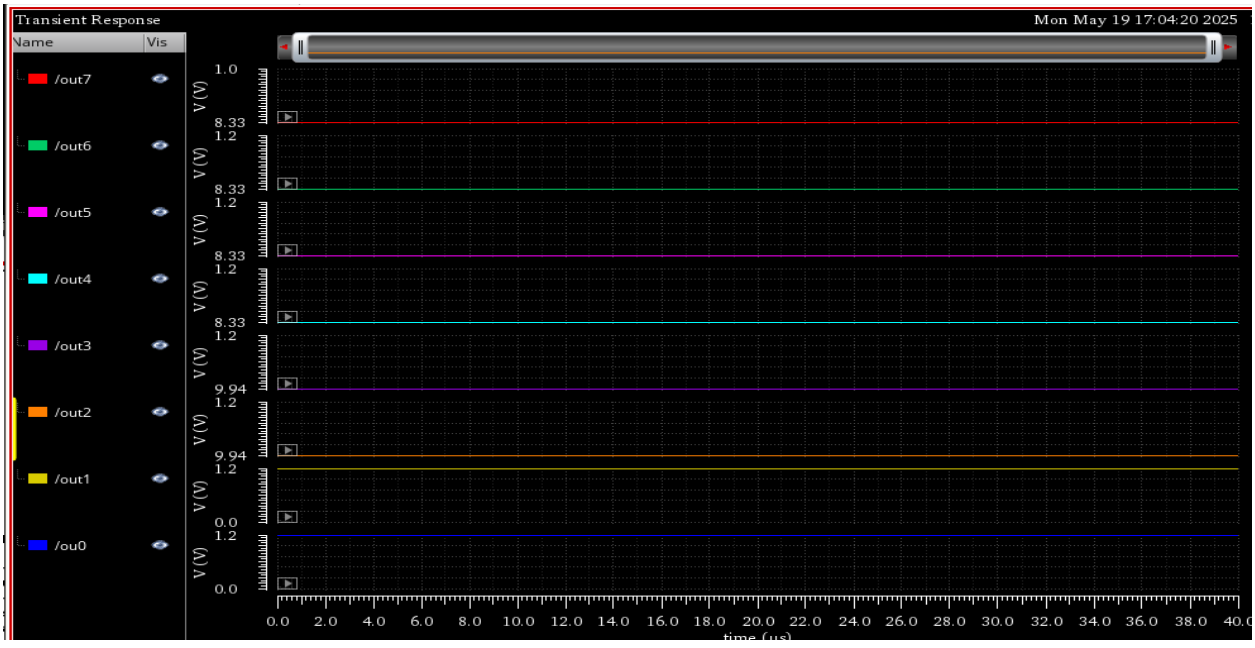


the output as shown→1111 0110=-10

7-Add

Selection lines→0110 A→0010=2 B→0001=1

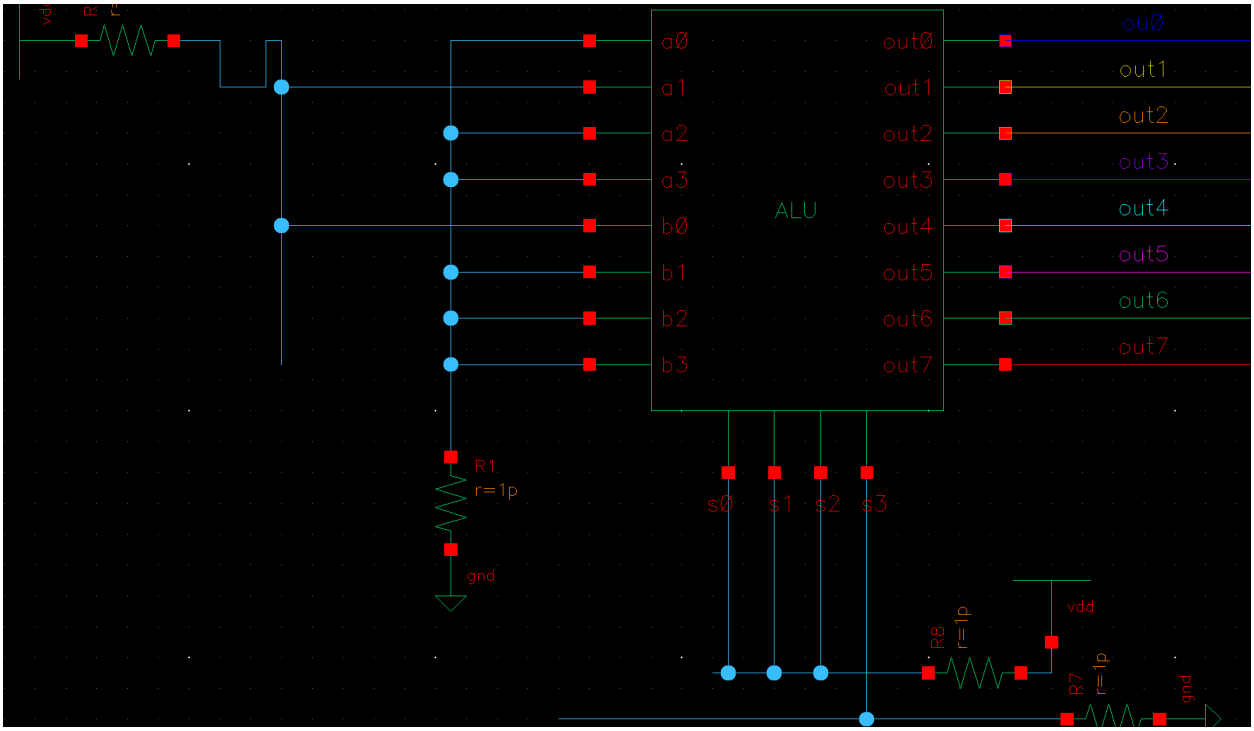


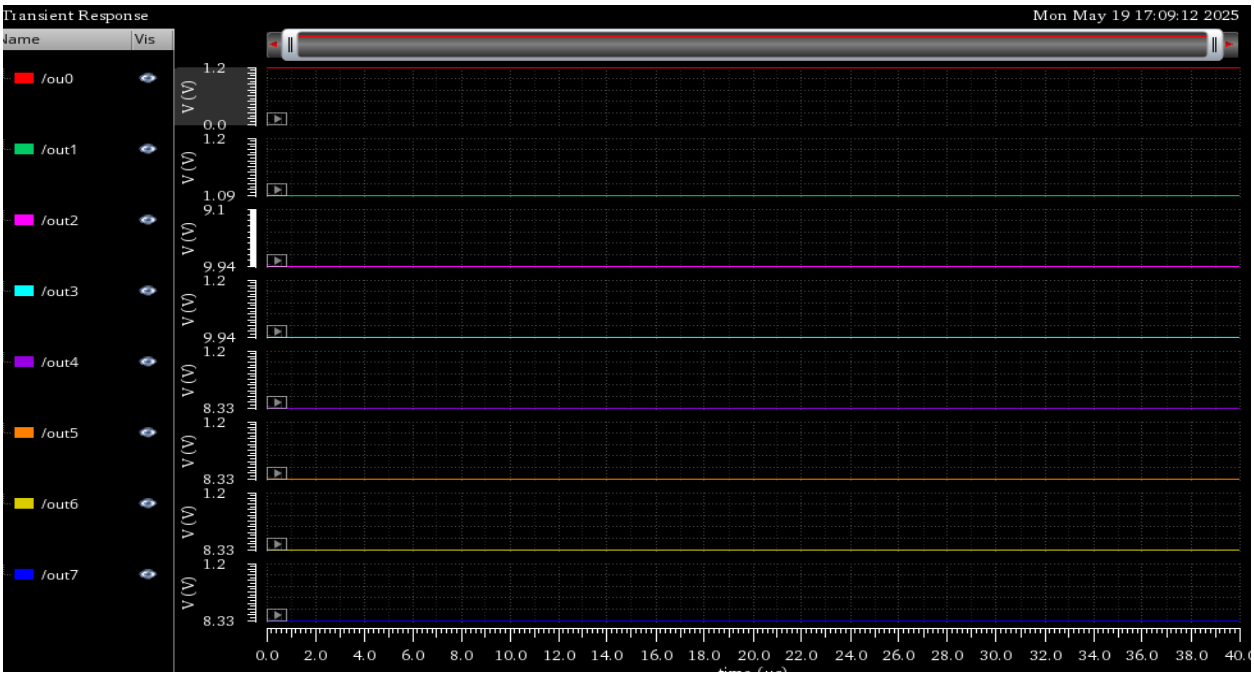


the output as shown→0000 0011=3

8-sub

Selection lines→0111 A→0010=2 B→0001=1

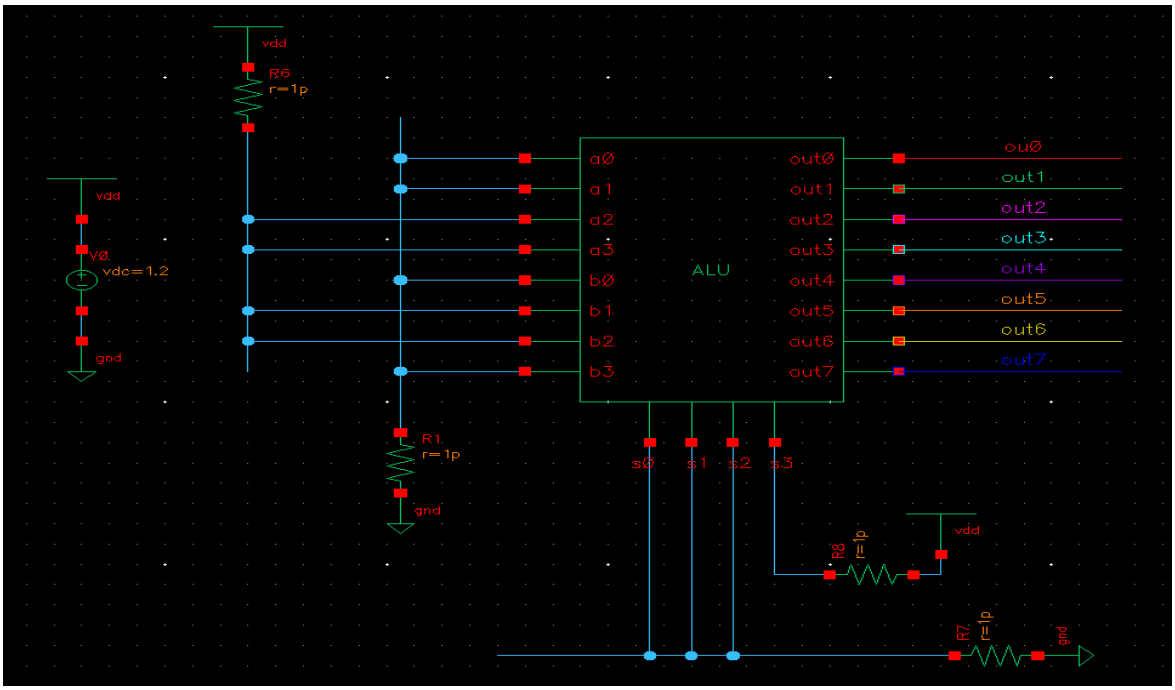




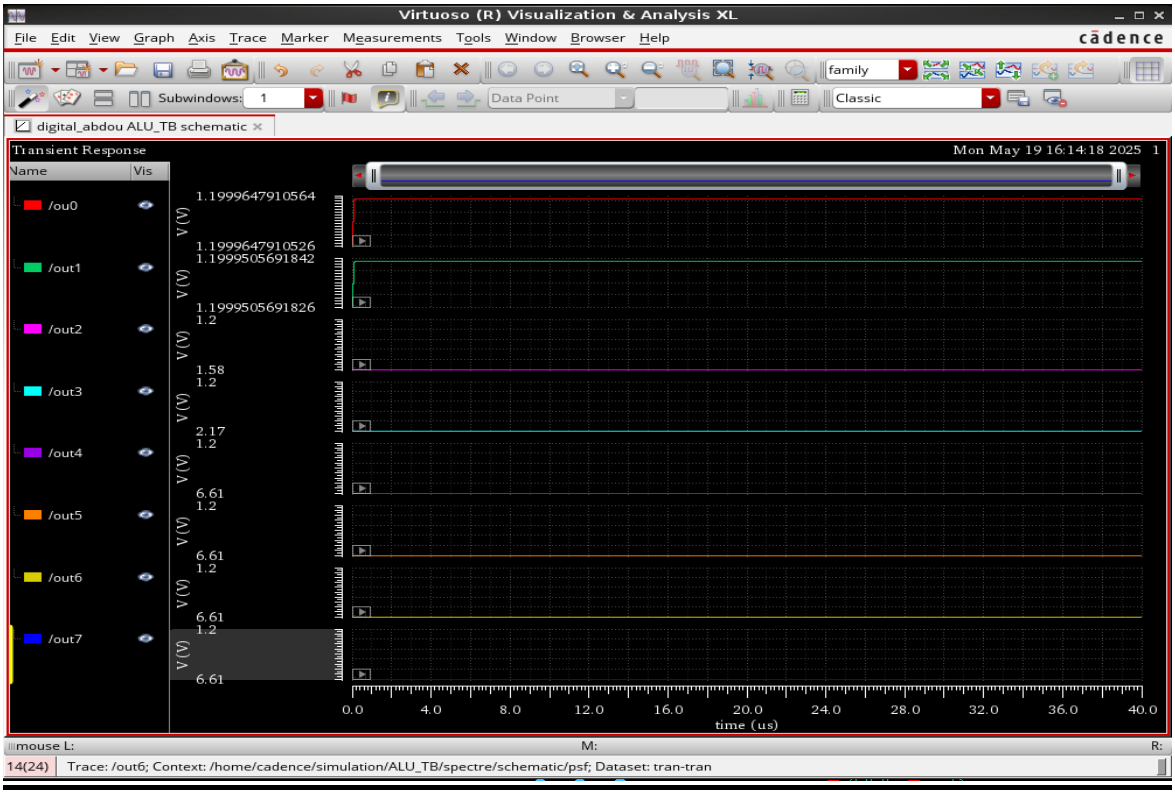
the output as shown→0000 0001=1

For all logical tests below inputs: A → 1100, B → 0110

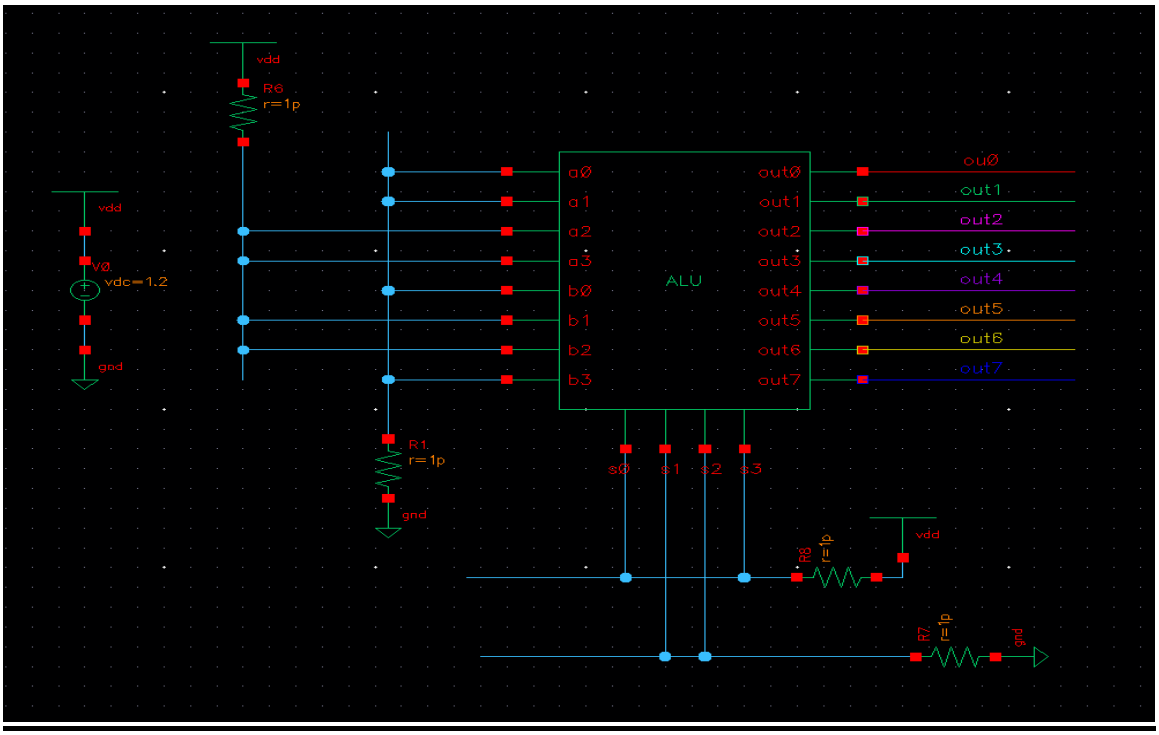
Complement A (selection line 1000)



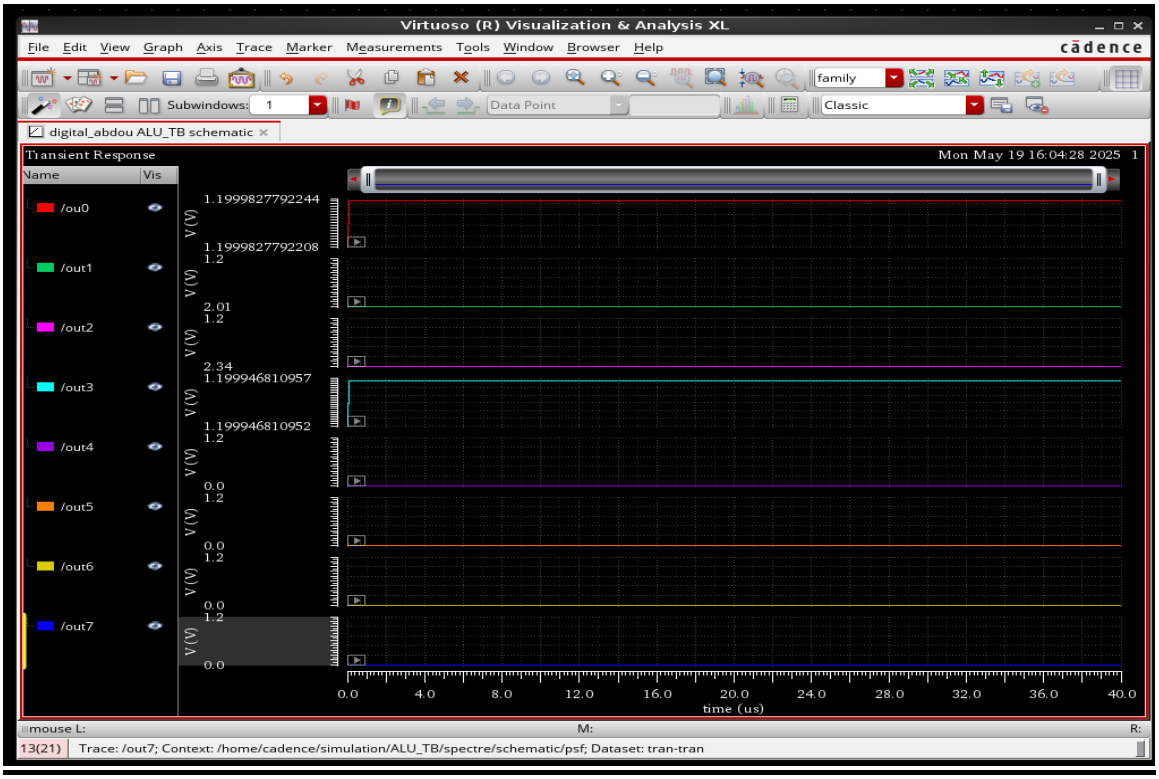
WaveForm



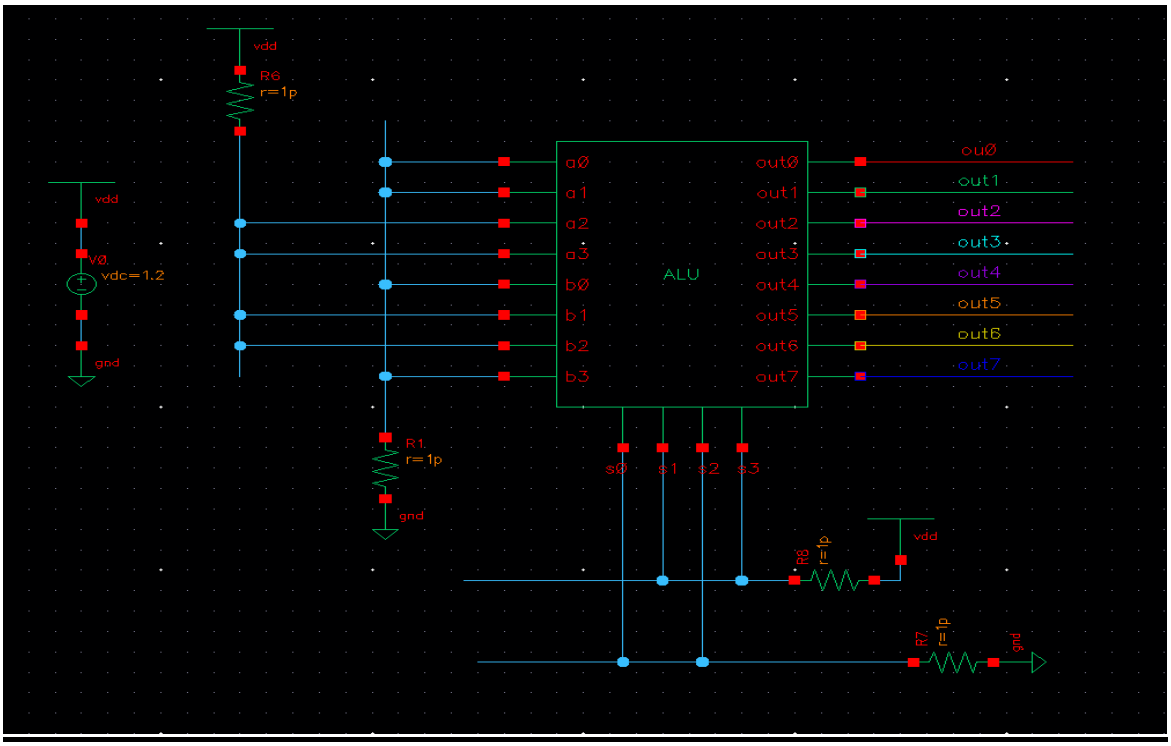
Complement B (selection line 1001)



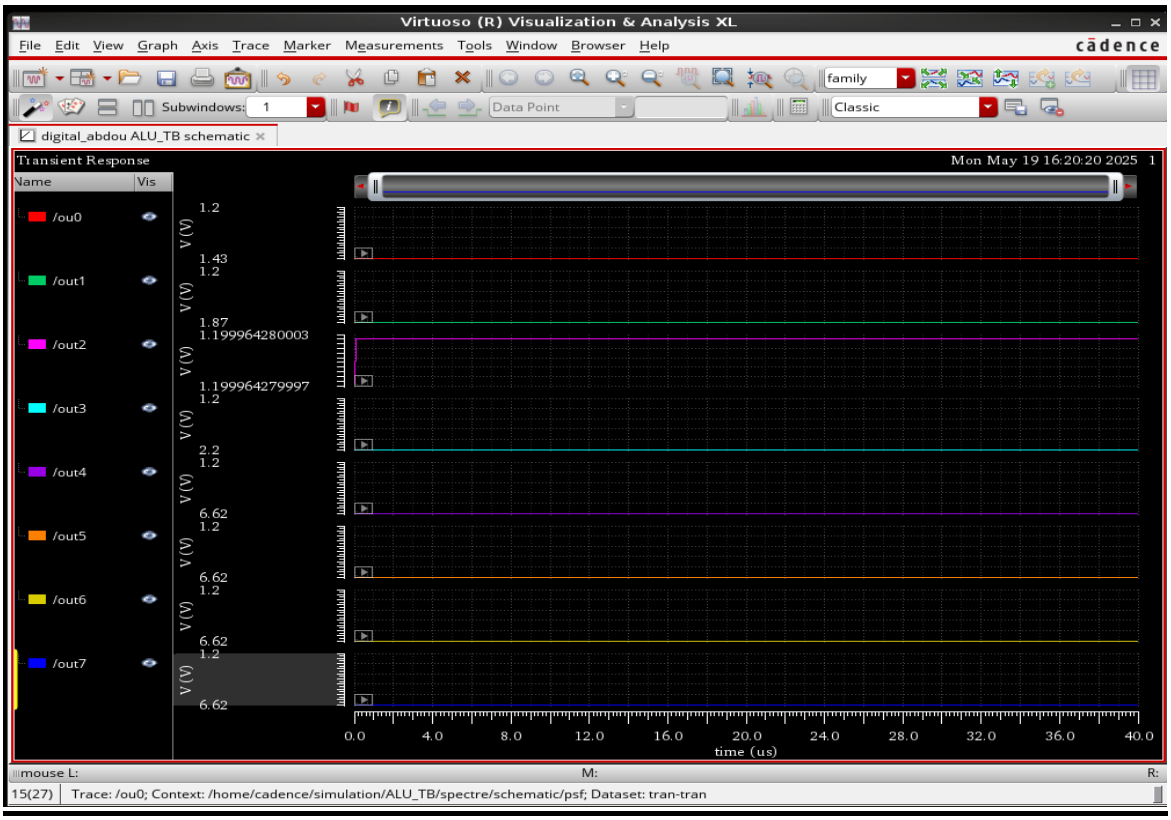
WaveForm



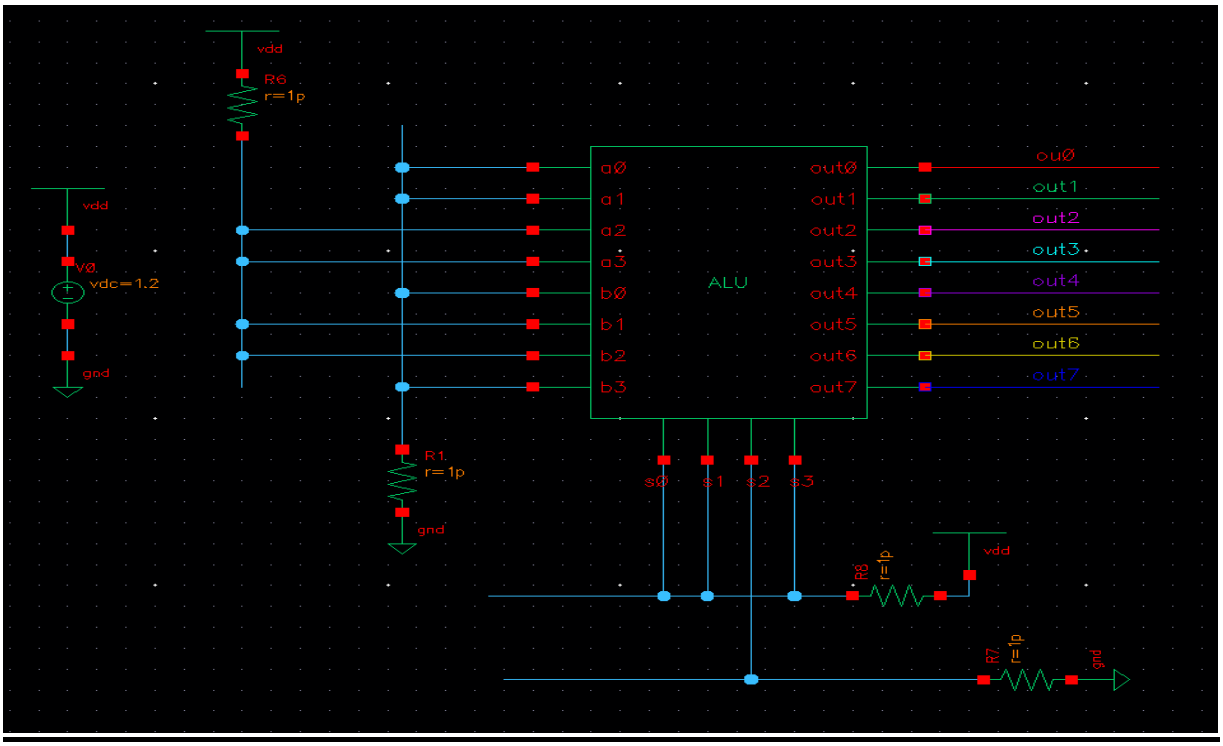
AND (selection line 1010)



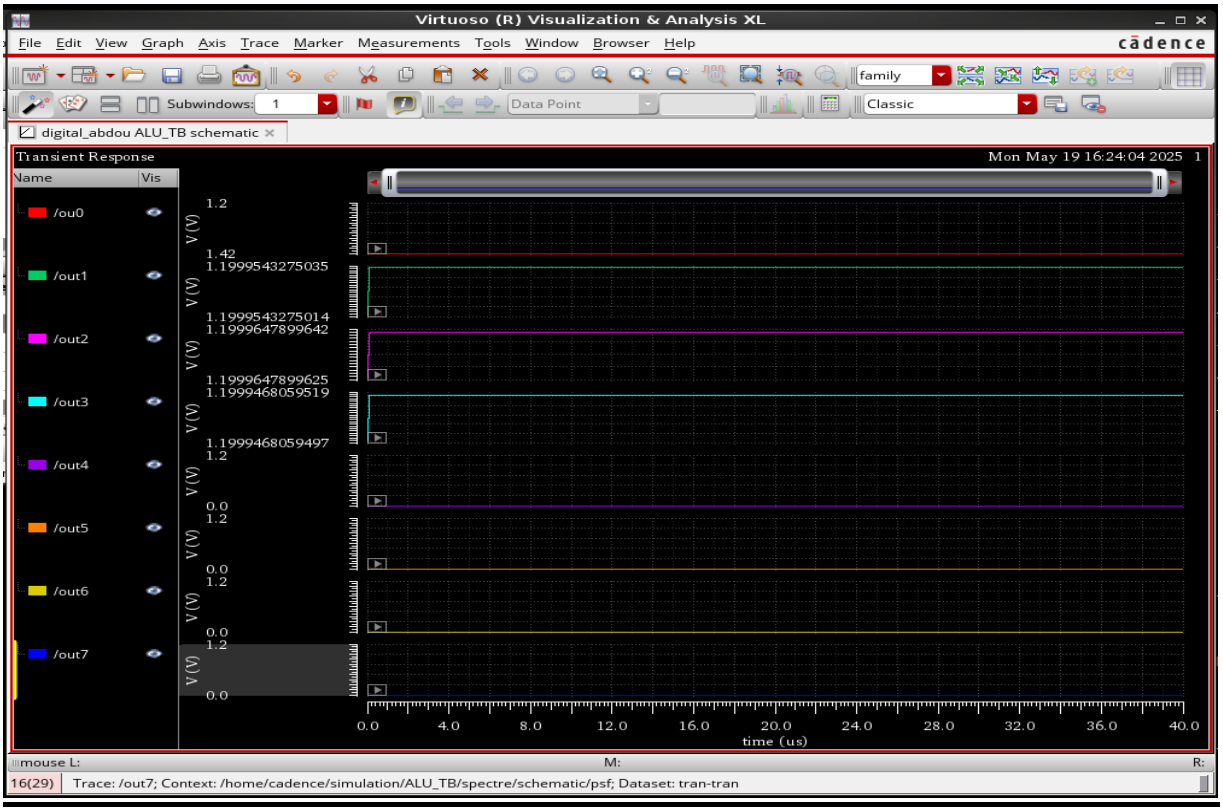
WaveForm



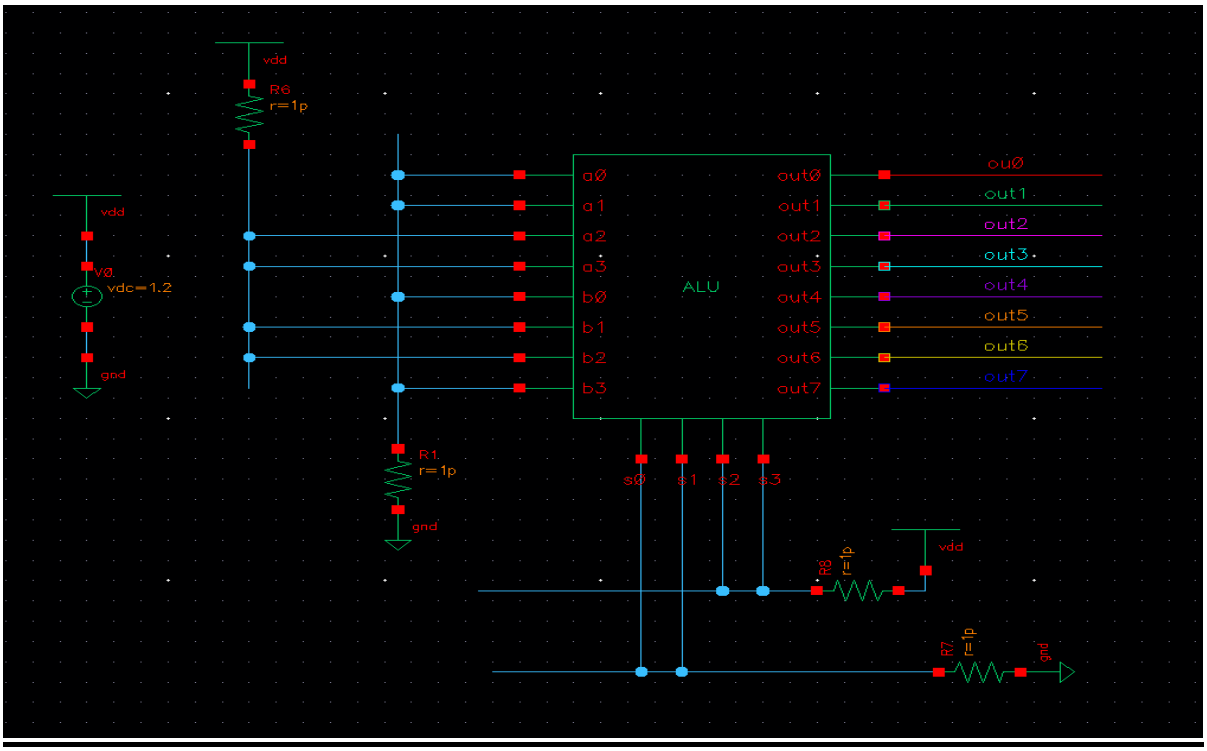
OR (selection line 1011)



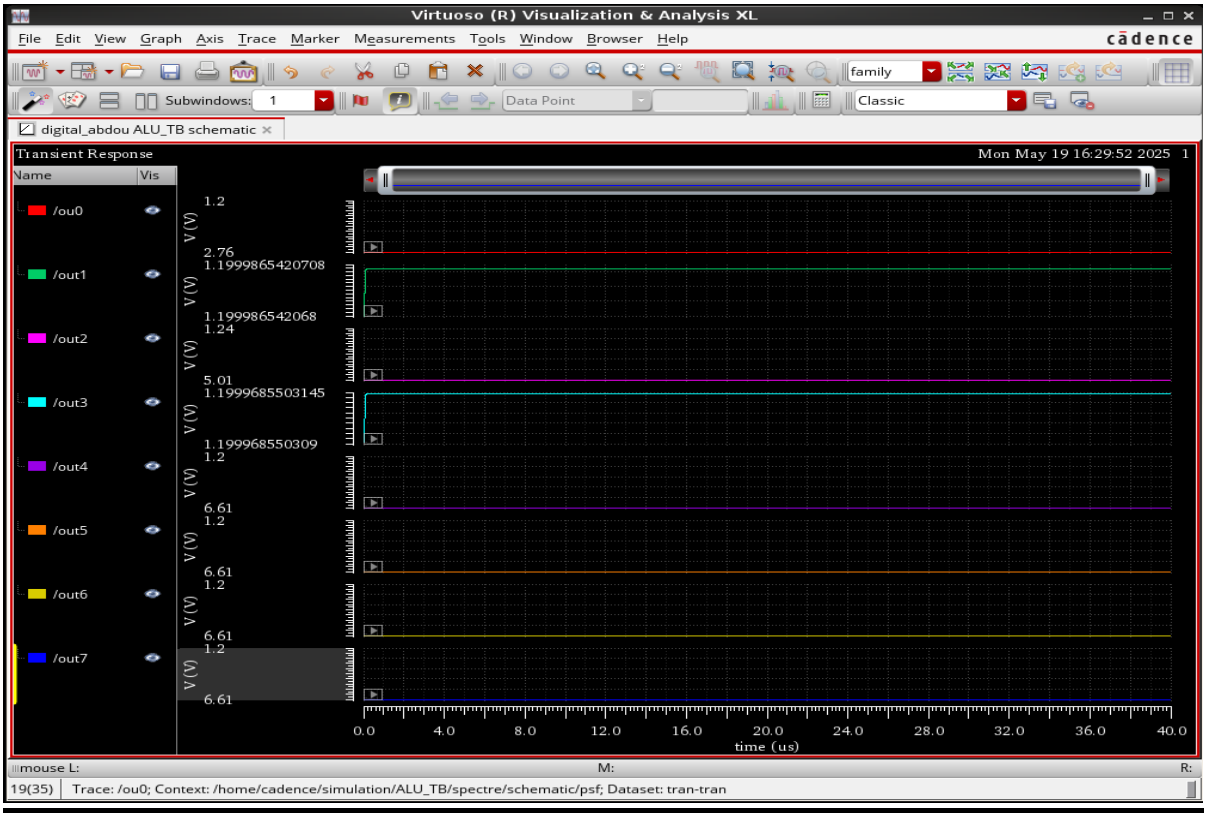
WaveForm



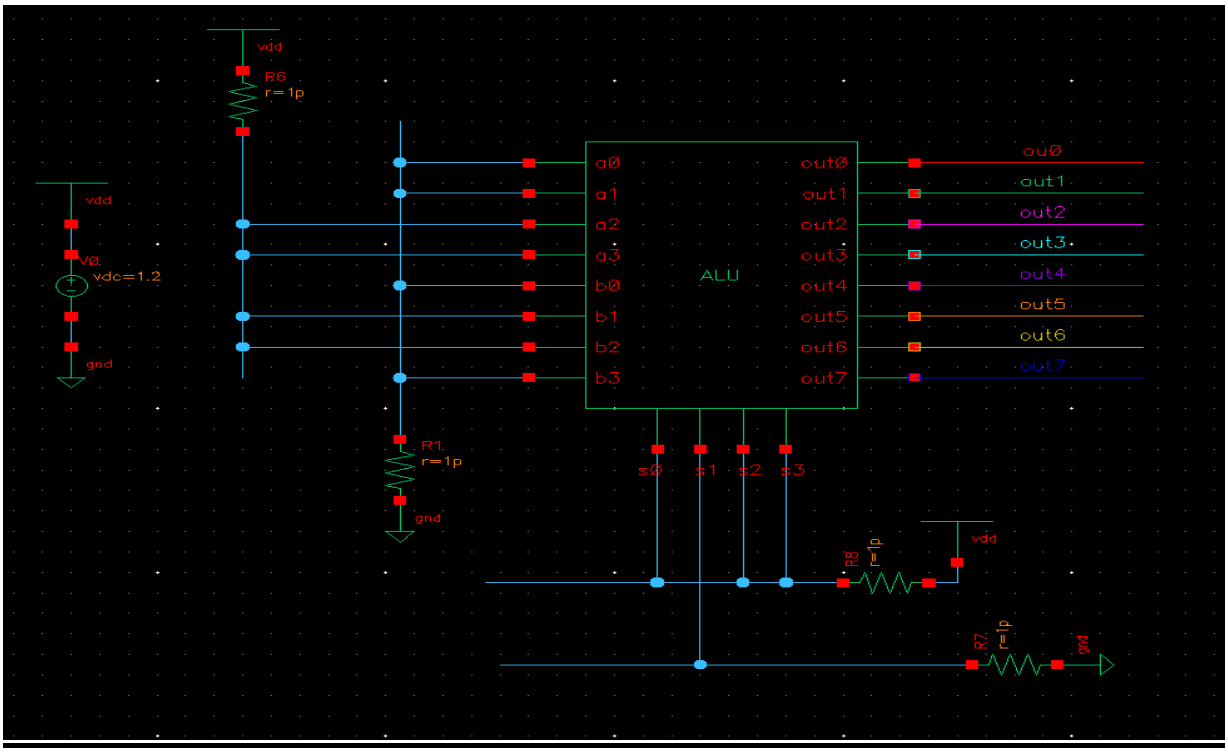
XOR (selection line 1100)



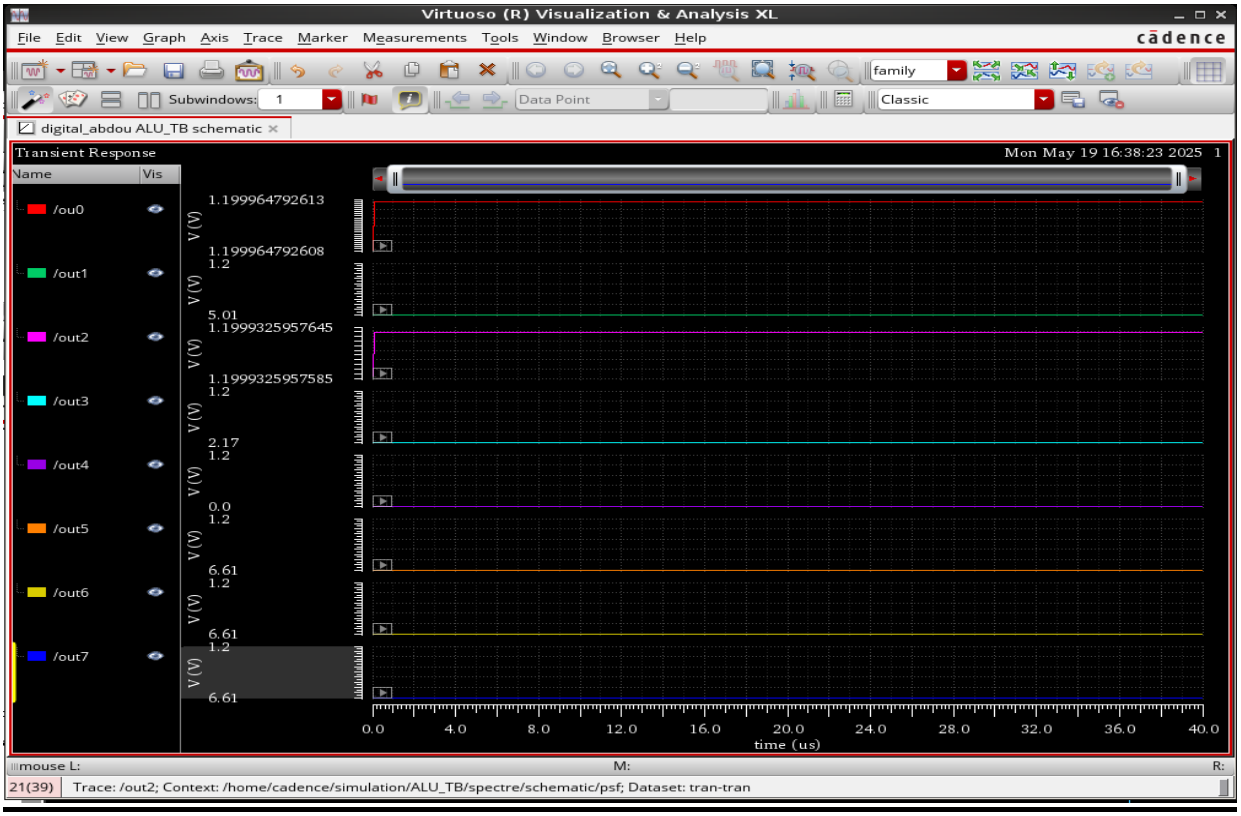
WaveForm



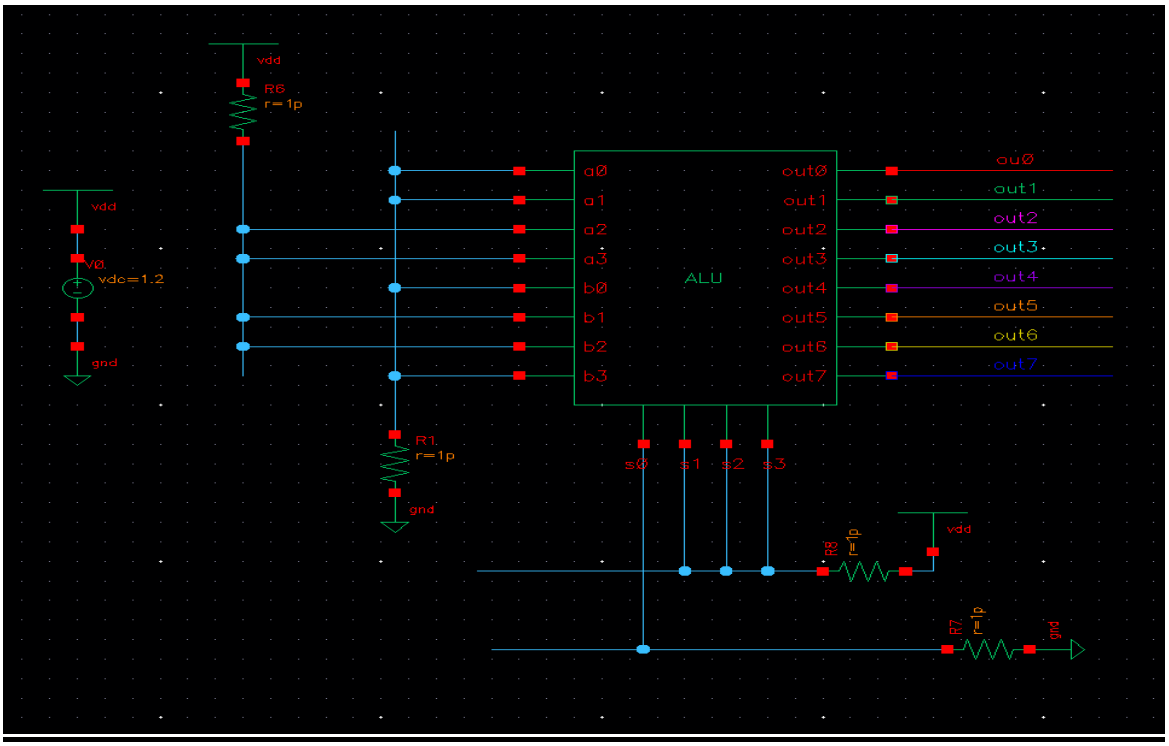
XNOR (selection line 1101)



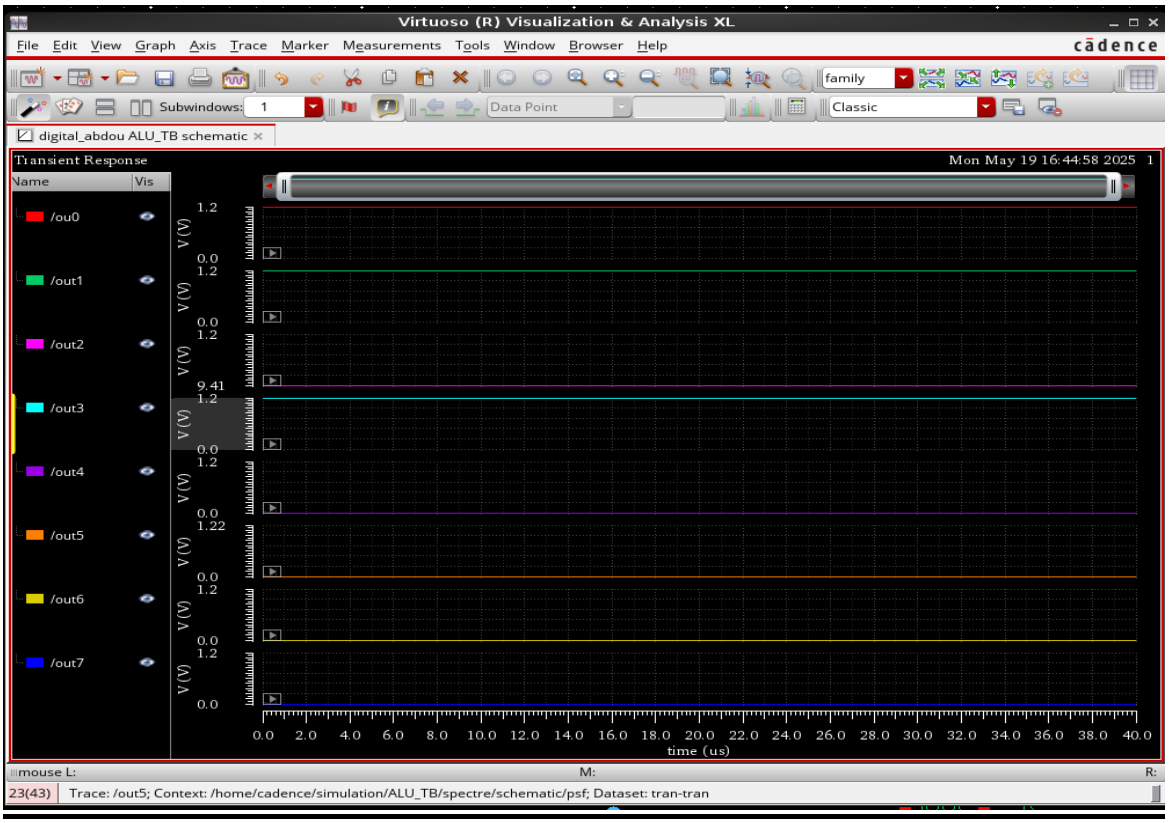
WaveForm



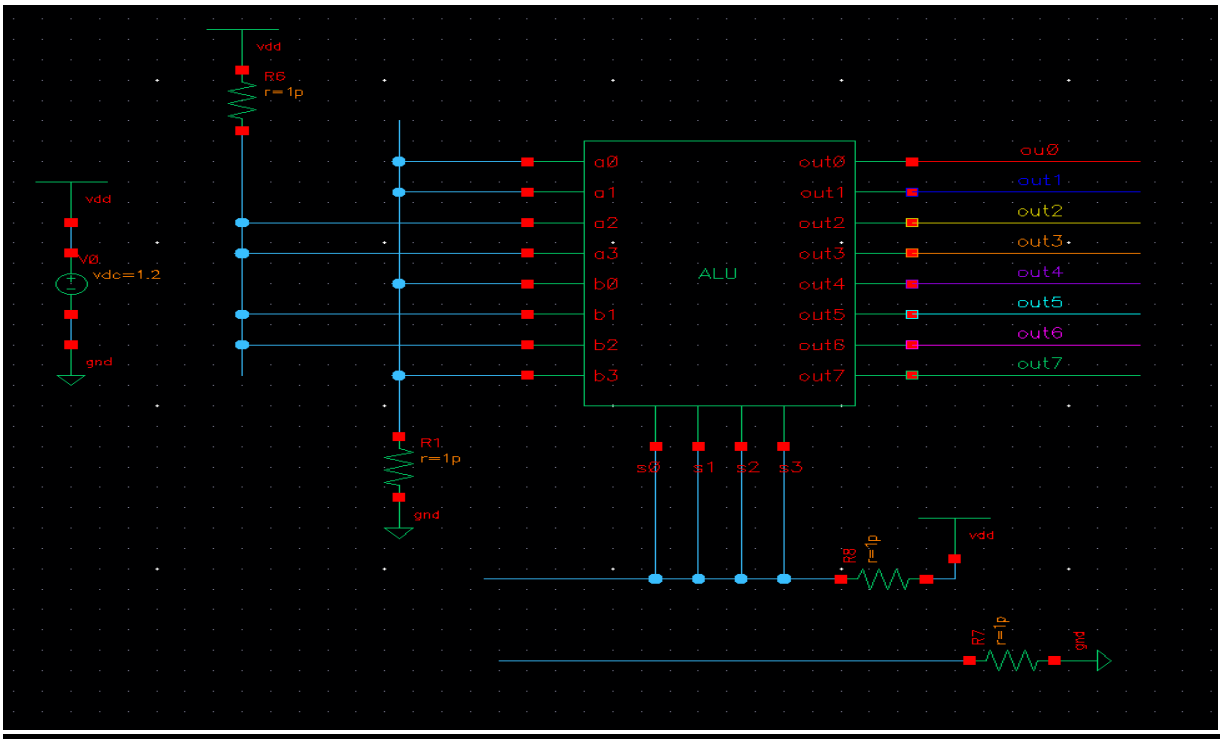
NAND (selection line 1110)



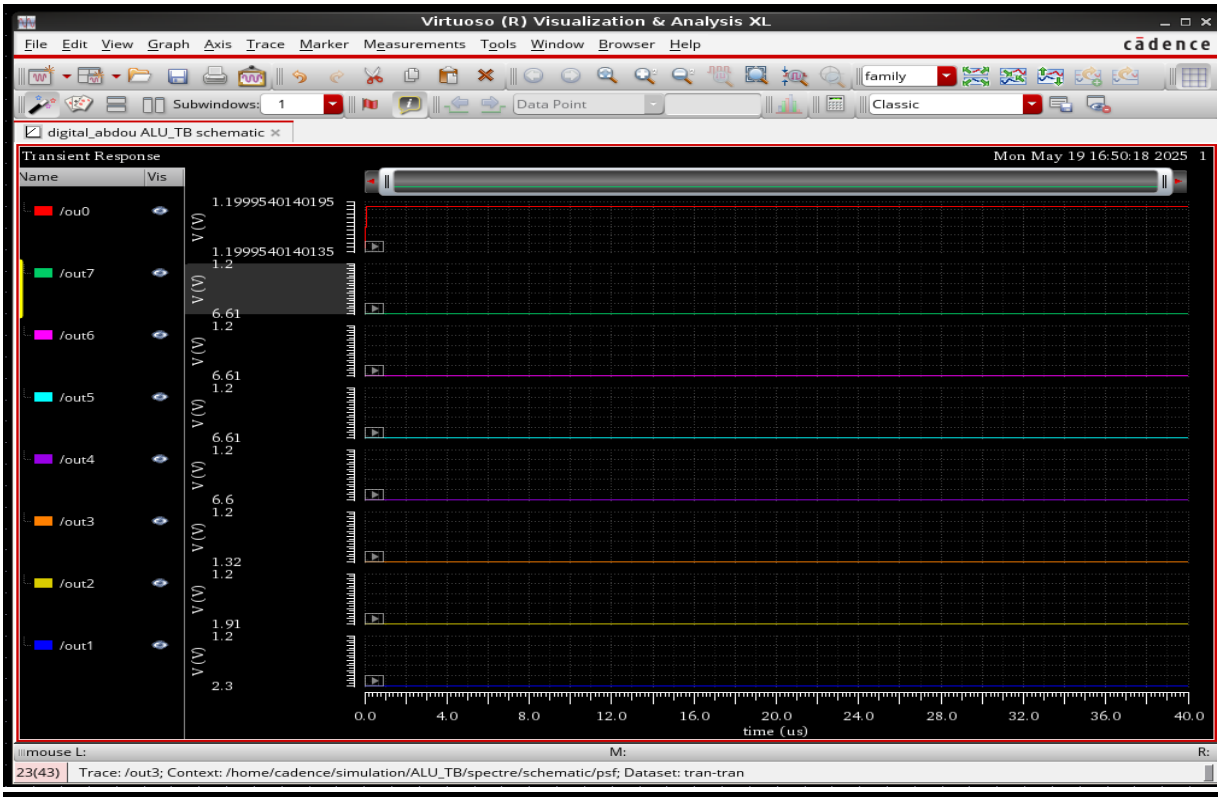
WaveForm



NOR (selection line 1111)



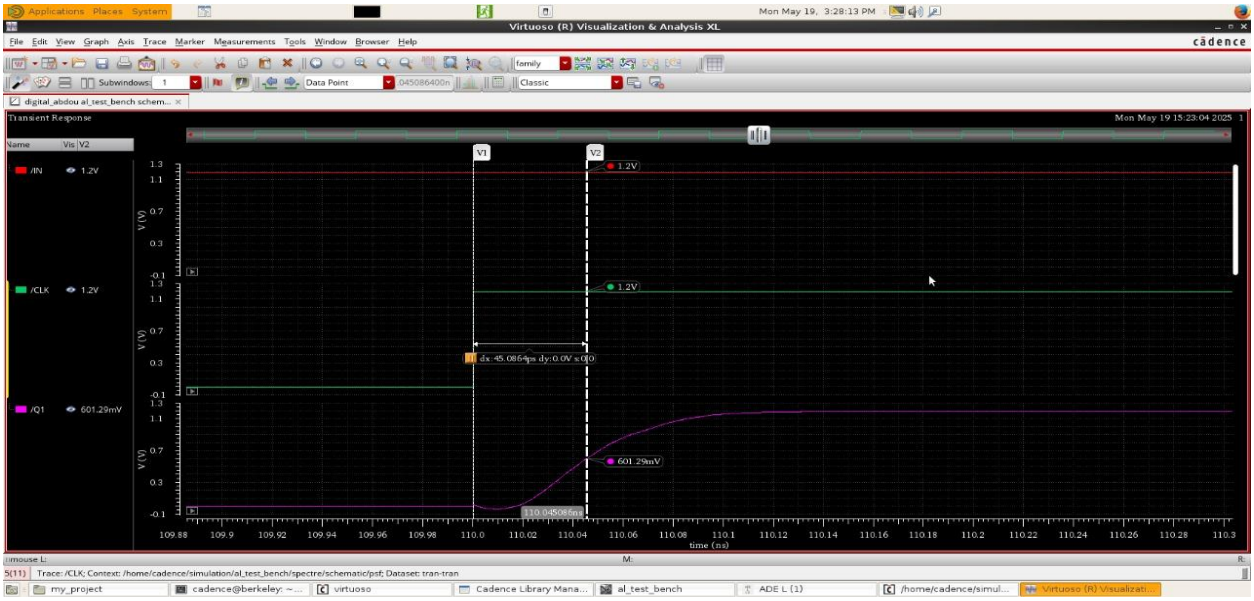
WaveForm



Delay calculations:

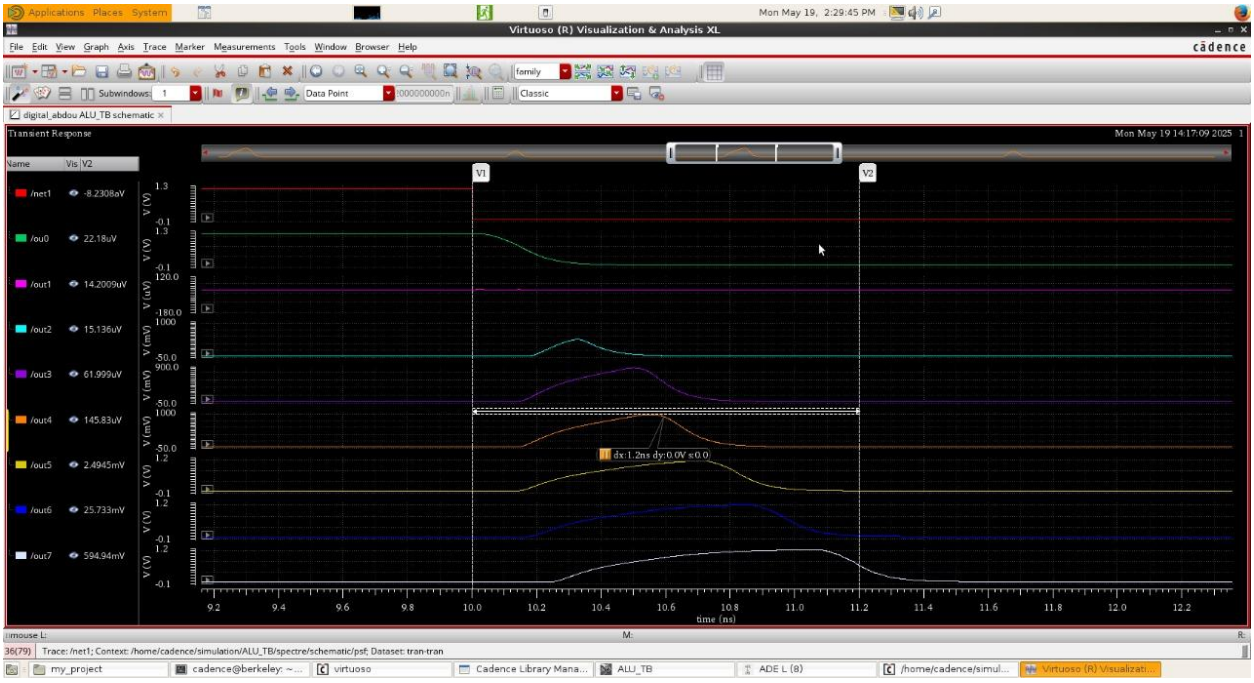
First, we calculate tcq which is the delay from clk to output

once from high to low and others from low to high and taking the worst.



Tcq = 48.51 ps

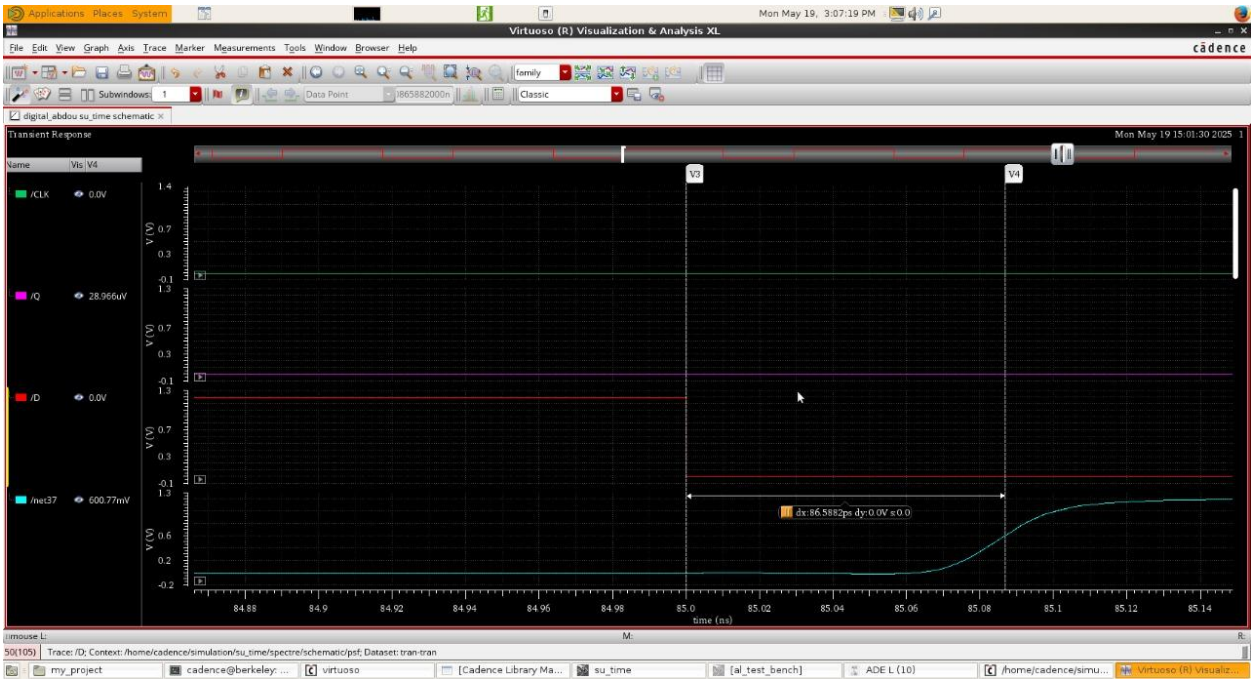
Second, we calculate worst case propagation delay



Tp = 1.35647 ns

Third, we calculate tsu:

(we take the worst case)



Tsu = 86.5882 Ps

Then,

Tclk (min) = Tcq +Tlogic +Tsu

= 48.51+1356.47+86.5882 = 1491.568 ps =1.491 ns

To have an error tolerance or in other words a bit of extra slack to avoid errors, we rounded up the clock period to 2ns

F(max) = 1/Tclk = 500 MHz

We then proceeded with the stated clock period in the following test-benches of the whole system that include:

1. **Inputs Flip Flops**
2. **Arithmetic Unit**
3. **Logical Unit**
4. **MUX Control Unit**
5. **Output Flip Flops**

Transient Analysis

Test Cases of the final ALU to show the correct sampling of the flipflops

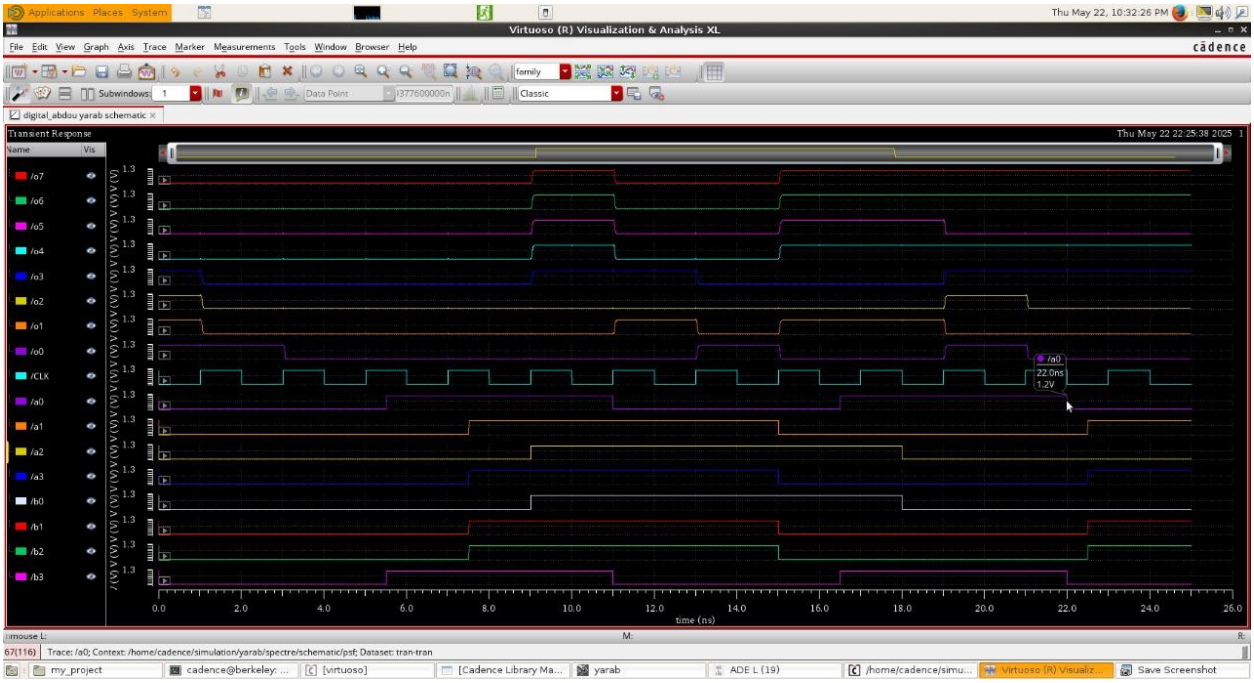
1-SUB: All Inputs of each A and B are bundled to two different voltage input lines for simplicity in this case

2-XOR: Inputs A01 & B23 are in on a different input voltage line from inputs A23 & B01

For the next two Cases, each input has its voltage line

3-OR:

4-Multiply:



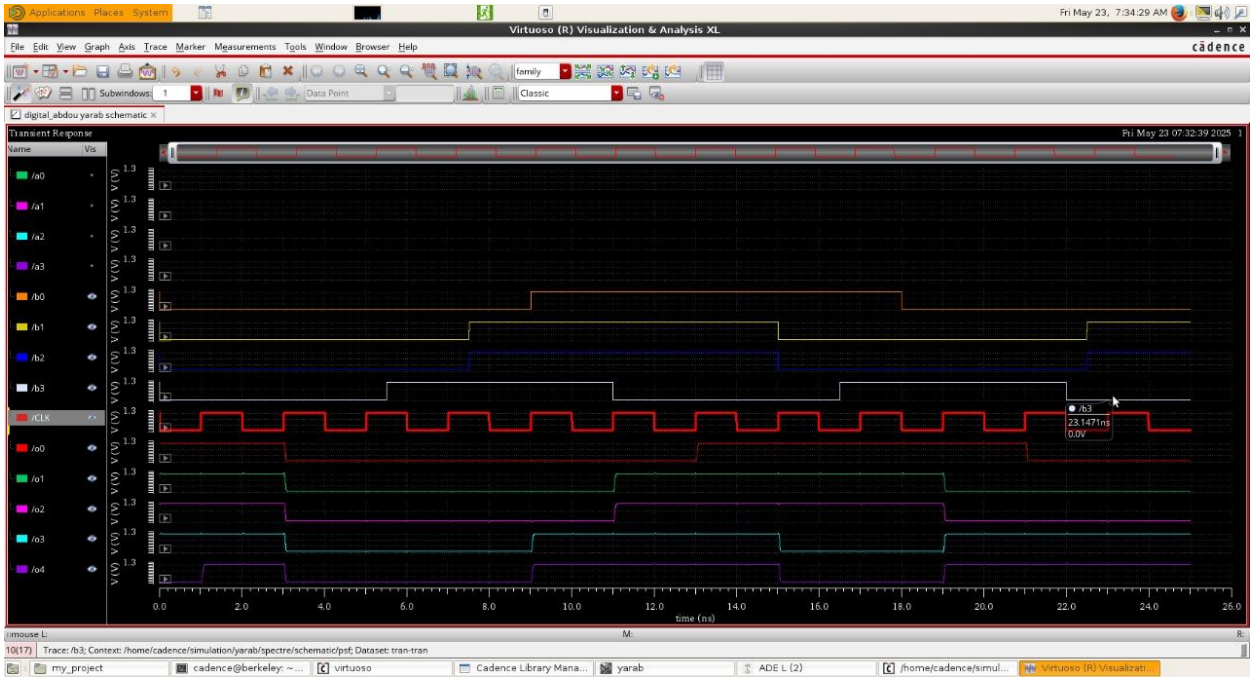
5-Increment A

6-Increment B

7-Transfer A

8-Decrement A

9-Transfer B

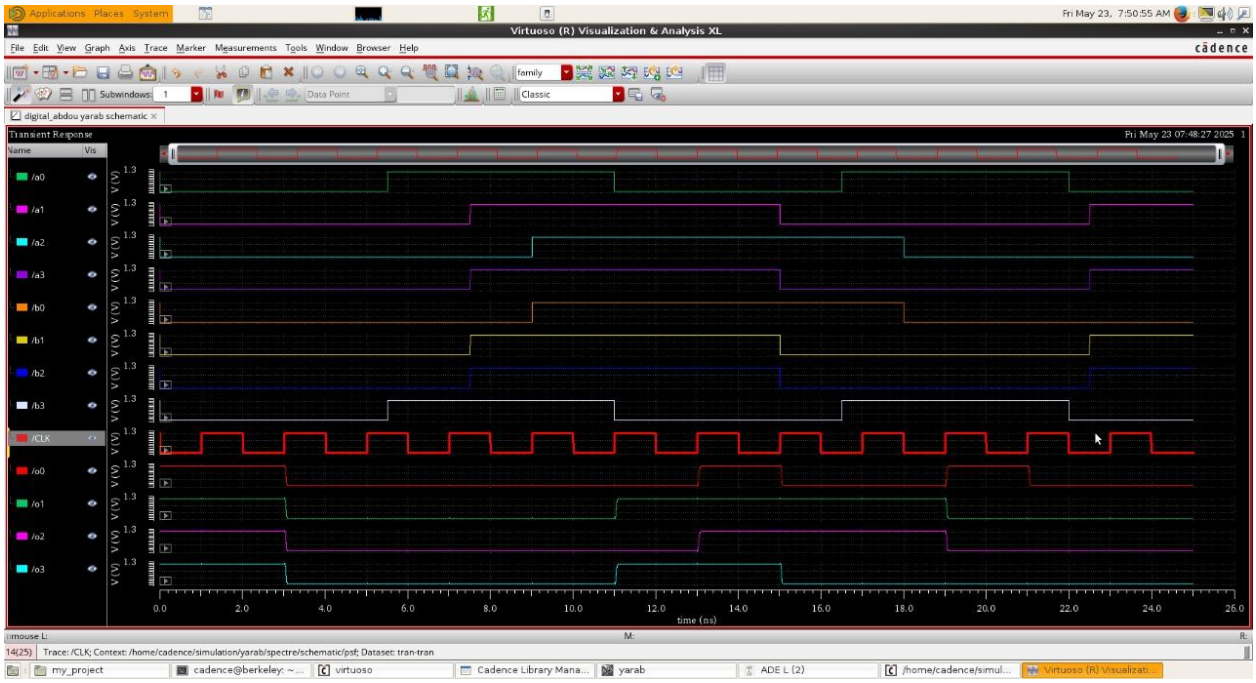


10-Add

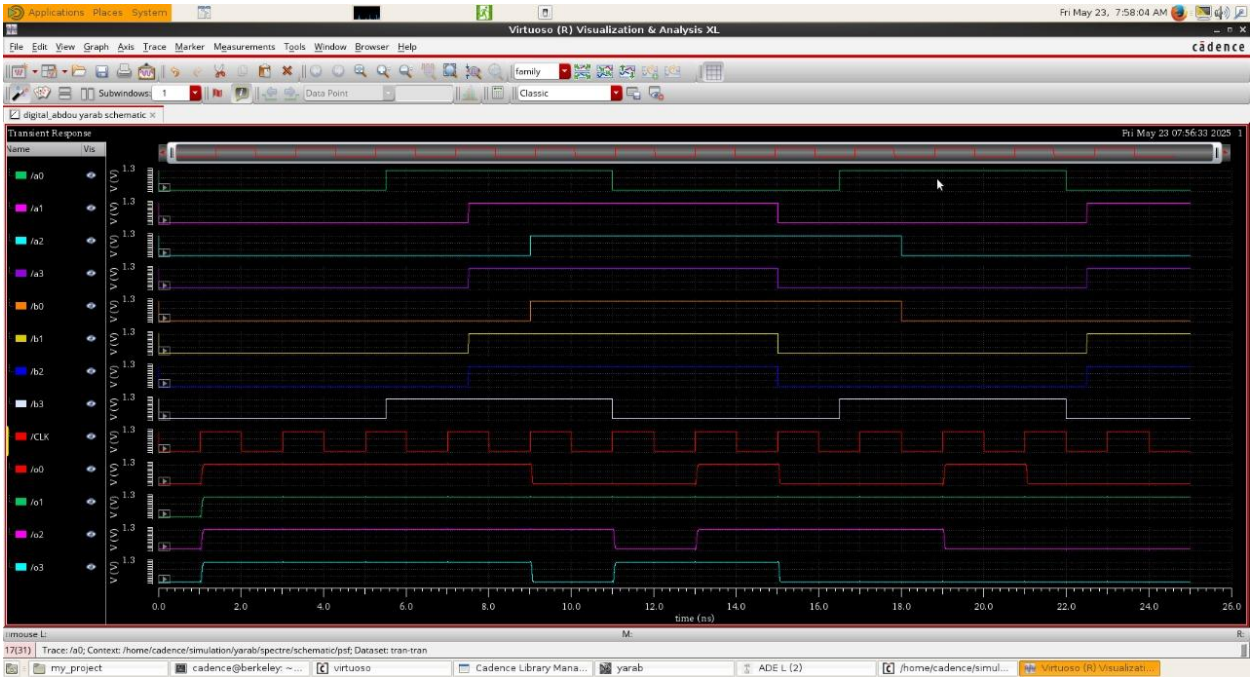
11-A Complement

12-B Complement

13-AND

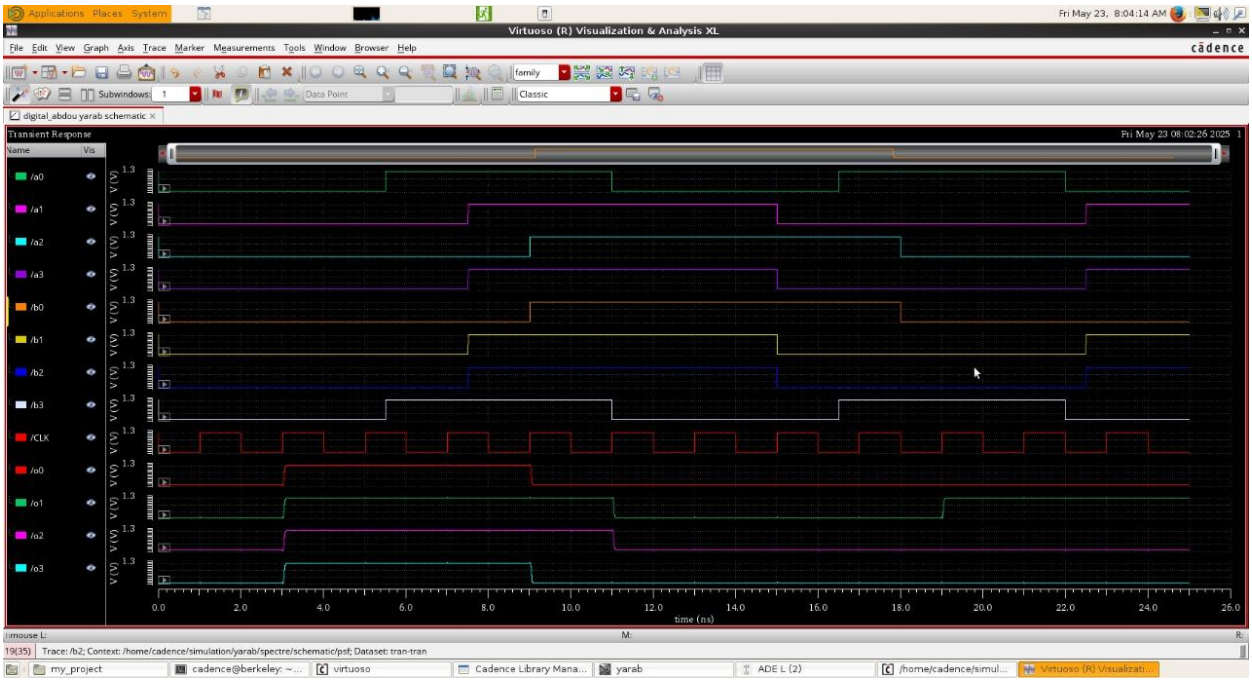


14-X-NOR



15-NAND

16-NOR



Notes including the Loading 2 pF capacitor

We started our design from the beginning with sizing of nanometer and taking a reference inverter of Wp/Wn =320n/160n. Therefore, using a 2 pF capacitor as a load would result in a huge loading effect that increases the delay enormously.

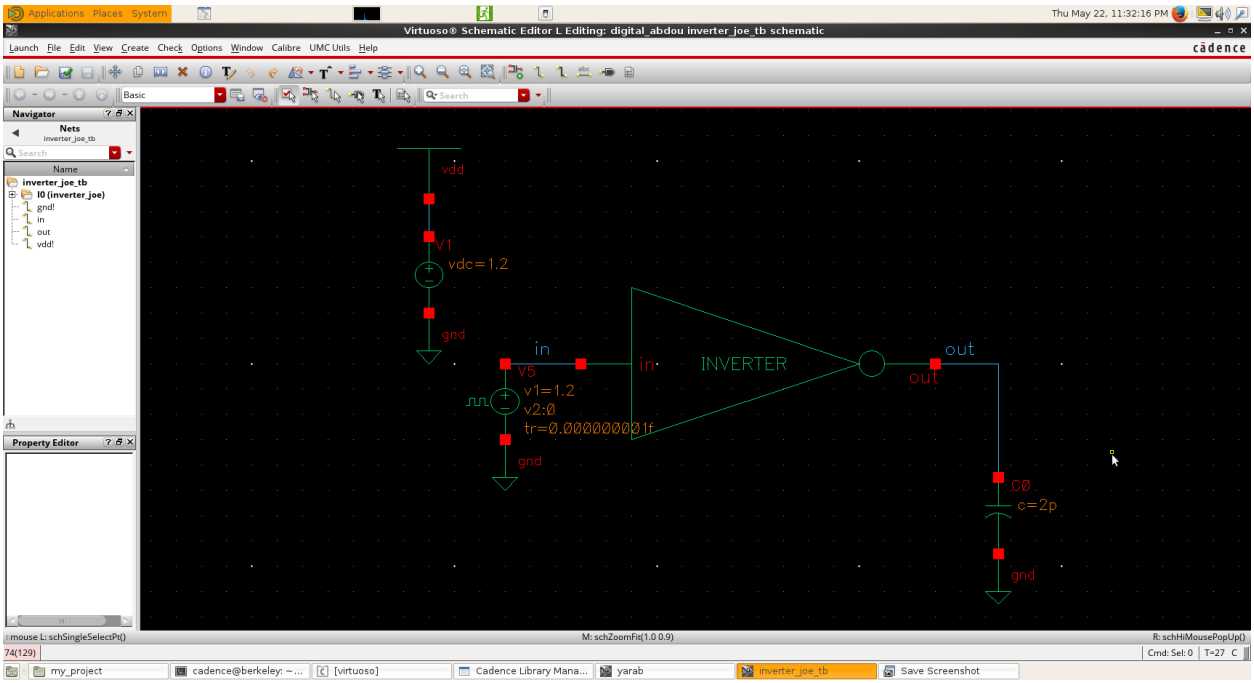
The increase in the delay would lead to such a worse clock frequency than the one we obtained. So, since it is not a requirement in the project description, we preferred not to add it, and if we were to add a loading capacitance, it would be of Femto Farad scale

The following figures would illustrated the delay due to intrinsic delay of the inverter and the delay after using the loading capacitor.

Intrinsic Delay:

Tphl = 8.24 ps

Loading Delay:



Tphl after loading = 18 ns

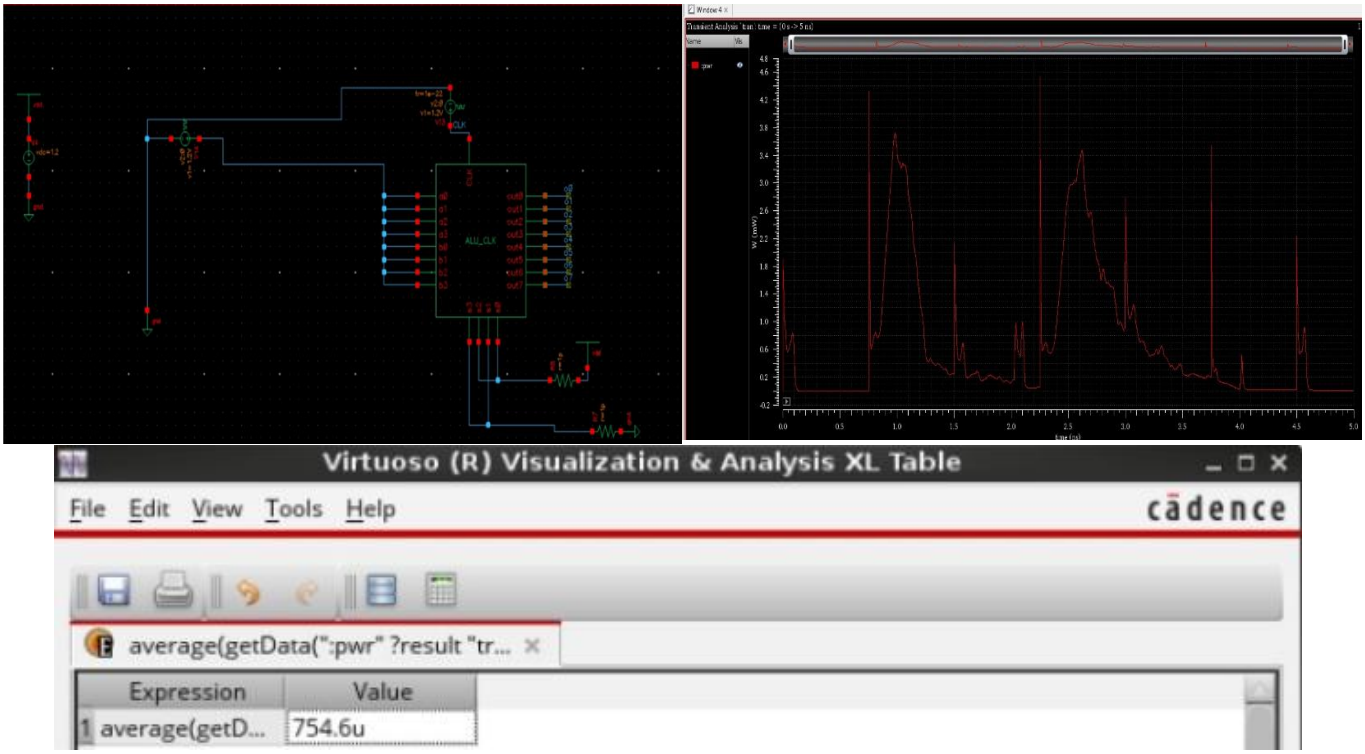
This means that the delay became larger by a factor of 2185, which is huge due to the difference between the gates input capacitances and the 2 pF

Power at Highest Frequency

We adjusted the clock to **1.5 nano seconds** to obtain the power in two most probable worst dynamic power. The cases are of the **Multiply and ADD** operations.

Multiply Power

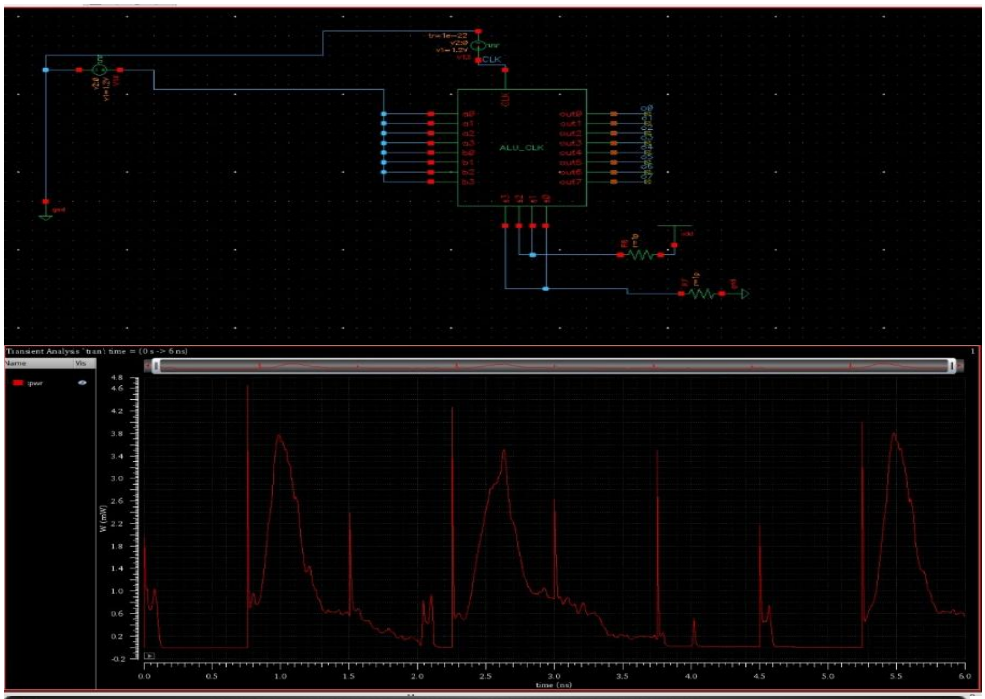
To get worst case power we made the inputs transition from 0 → 1 and back from 1 → 0 and took average power of simulation

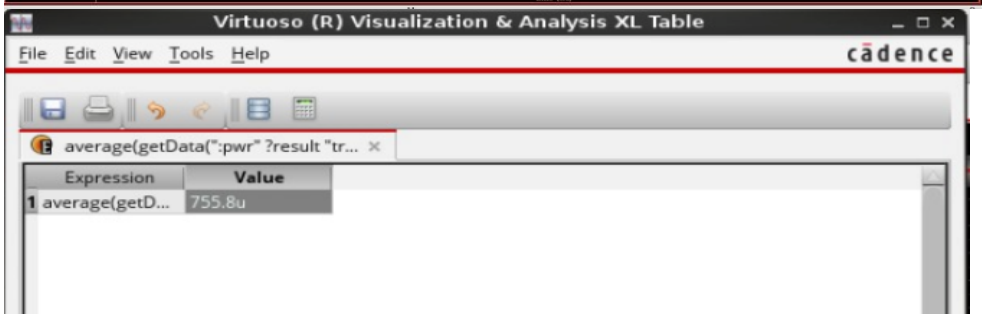


It is observed that the Average power from Simulation is **0.7546 mW**

ADD Power

Like Multiply we made the inputs transition from 0 → 1 and back from 1 → 0 and took average power of simulation





The average Power of ADD Operation came out ass **0.7558 mW**

Power of ALU

The Dynamic power of **Multiply and ADD** operation came out relatively the same so we could fairly assume that average worst-case power is equal to about **755 uW or 0.755 mW**

Verilog Code and Test-Bench

Note: The clock here is of 1.5ns as it isn't affected by noise as much as in cadence (Just for the sake of Optimization)

The codes are test-benched and configured on a clock period of 1.5 nsec

alu.v

```
module alu_registered(
    input clk,
    input signed [3:0] A,
    input signed [3:0] B,
    input [3:0] opcode,
    output reg signed [7:0] result
    );

    reg signed [3:0] A_reg, B_reg;
    reg signed [7:0] alu_result;

    // Register inputs A and B
    always @(posedge clk) begin
        A_reg <= A;
        B_reg <= B;
    end

    // Combinational ALU logic using registered inputs and combinational opcode
    always @(*) begin
        case(opcode)
            4'b0000: alu_result = A_reg + 1;
            4'b0001: alu_result = B_reg + 1;
            4'b0010: alu_result = A_reg;
            4'b0011: alu_result = B_reg;
            4'b0100: alu_result = A_reg - 1;
            4'b0101: alu_result = A_reg * B_reg;
            4'b0110: alu_result = A_reg + B_reg;
```

```
            4'b0111: alu_result = A_reg - B_reg;
            4'b1000: alu_result = ~A_reg;
            4'b1001: alu_result = ~B_reg;
            4'b1010: alu_result = A_reg & B_reg;
            4'b1011: alu_result = A_reg | B_reg;
            4'b1100: alu_result = A_reg ^ B_reg;
            4'b1101: alu_result = ~(A_reg ^ B_reg);
            4'b1110: alu_result = ~(A_reg & B_reg);
            4'b1111: alu_result = ~(A_reg | B_reg);
            default: alu_result = 0;
        endcase
    end

    // Register the ALU output result on clk (single flip-flop)
    always @(posedge clk) begin
        result <= alu_result;
    end

endmodule
```

---

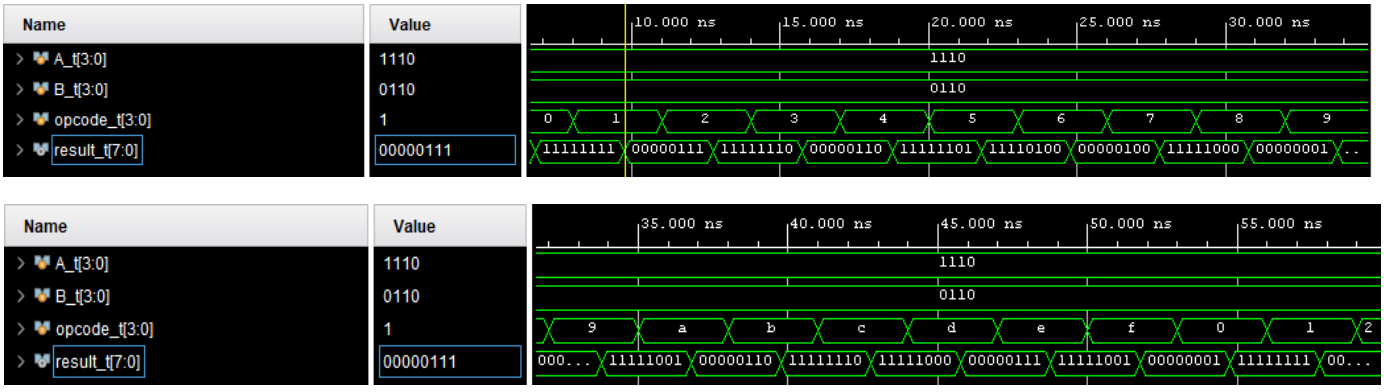
alu_tb.v

```
`timescale 1ns / 1ps
// Description: Testbench for ALU with registered inputs/outputs
module alu_tb;

    reg clk;
    reg signed [3:0] A_t;
    reg signed [3:0] B_t;
    reg [3:0] opcode_t;
    wire signed [7:0] result_t;
```

```
    // Instantiate DUT
    alu uut (
        .clk(clk),
        .A(A_t),
        .B(B_t),
        .opcode(opcode_t),
        .result(result_t)
    );

    // Clock generation: 670 MHz (T = 1.5 ns)
    initial clk = 0;
    always #0.75 clk = ~clk; // Toggle every 0.75ns → 1.5ns period

    initial begin
        $display("Time    | A     B   | opcode | result");
        $display("--------|-------------|--------|--------");

        A_t = 0; B_t = 0; opcode_t = 0;
        #5; // Wait for initial setup

        for(opcode_t = 4'b0000; opcode_t <= 4'b1111; opcode_t = opcode_t + 1) begin
            A_t = 4'b1110;
            B_t = 4'b0110;
            #3; // Wait for clock
            $display("%0t | %b %b | %b  | %b", $time, A_t, B_t, opcode_t, result_t);
        end

        $stop;
    end

endmodule
```

Vivado Waveforms

The followings are Waveforms of Flip Floped ALU:



A	B	opcode	result
1110	0110	0000	11111111
1110	0110	0001	00000111
1110	0110	0010	11111110
1110	0110	0011	00000110
1110	0110	0100	11111101
1110	0110	0101	11110100
1110	0110	0110	00000100
1110	0110	0111	11111000
1110	0110	1000	00000001
1110	0110	1001	11111001
1110	0110	1010	00000110
1110	0110	1011	11111110
1110	0110	1100	11111000
1110	0110	1101	00000111
1110	0110	1110	11111001
1110	0110	1111	00000001

Final Vivado Schematic