Digital Circuits

ALU Design

	Name	ID
1	Muhammed Ahmed Abdel-Gawad Nassif	2001771
2	Karim mostafa mohamed roshdy nabhan	2001563
3	Abdelrahman Ahmed Abdelrahman Mahrous	2001760
4	Ahmed Khaled Abdelmaksod Ebrahim	2000218
5	Ahmed Mohammed Bakr Ahmed	2000037
6	Mahmoud mohamed alsayd soliman	2001278
7	Ahmed Samir Tharwat Mohamed	2000217
8	Yousef Shawky Mohamed	2001500
9	Hashim Abdulaziz Hashim	2001506
10	Fathy Abdlhady Fathy	2001152
11	Marwan Wael Mahmoud abbas	2001244

Table of contents

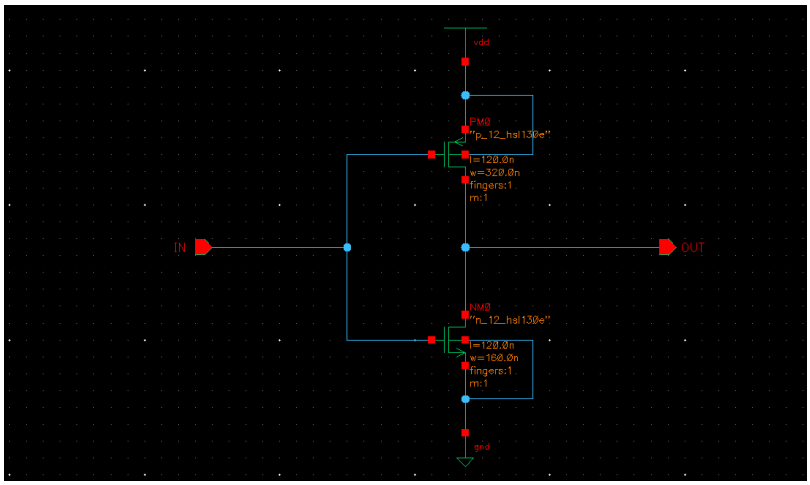
- Inverter
- NAND GATE
- NOR GATE
- XOR GATE
- XNOR GATE
- AND GATE
- BUFFER GATE
- 2X1 MUX
- 4X1 MUX
- 8X1 MUX
- FULL ADDER
- LOGIC UNIT
- SIGNED I/O ARITHEMATIC UNIT WITH A NEGATIVE FLAG (BONUS)
- FLIP FLOP
- SIGNED I/O ALU WITH FLIP FLOP & NEGATIVE FLAG (BONUS)
- Transient simulations of some operations
- WORST CASE DELAY
- POWER CONSUMPTION AT MAX FREQUENCY
- UNSIGNED LOGIC VERILOG
- UNSIGNED TESTBENCH
- SIGNED I/O code VERILOG (BONUS)
- SIGNED I/O TESTBENCH (BONUS)

Note:

The report includes all main requirements + 3 points for bonus marks.

Cadence

Inverter



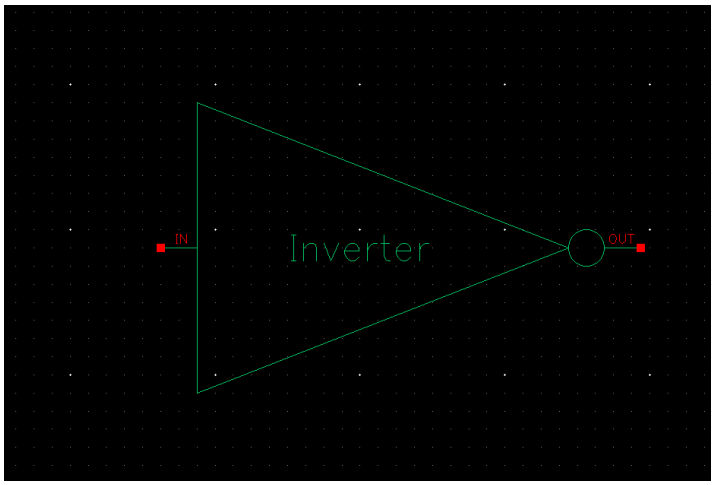
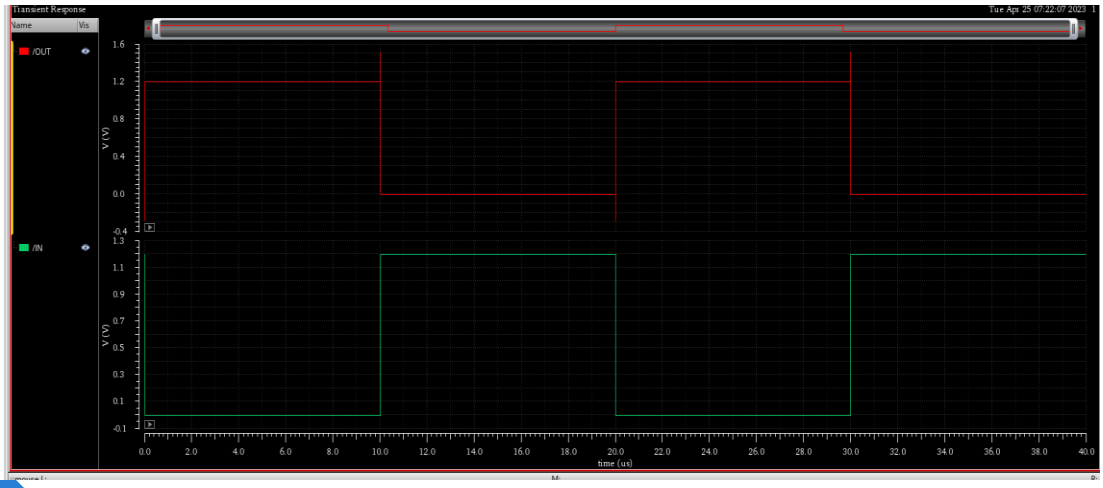
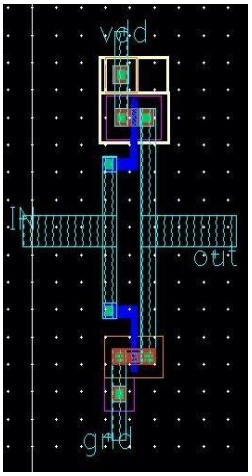


Figure: inverter schematic and symbol

We designed the inverter as $WP/\ Wn$ = 2: 1 and used the minimum length and provided in 130nm technology (120nm), so if $Wnmin$ = 160nm, WP = 160 $*$ 2 = 320nm. Then we tested the results for all possible inputs (square waves) as shown in figure and table below:

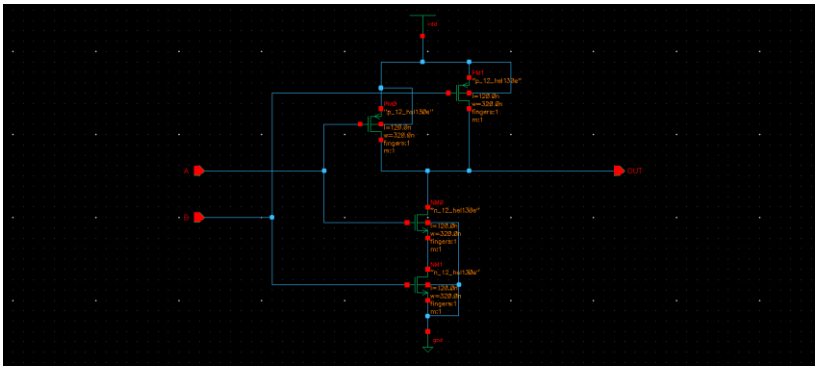




Inverter layout "Bonus"

IN	OUT
1	0
0	1

NAND Gate



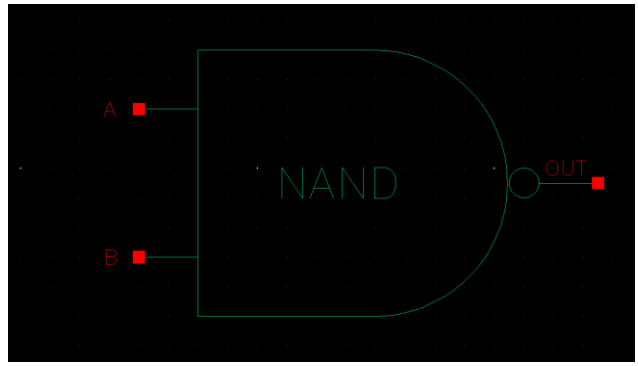
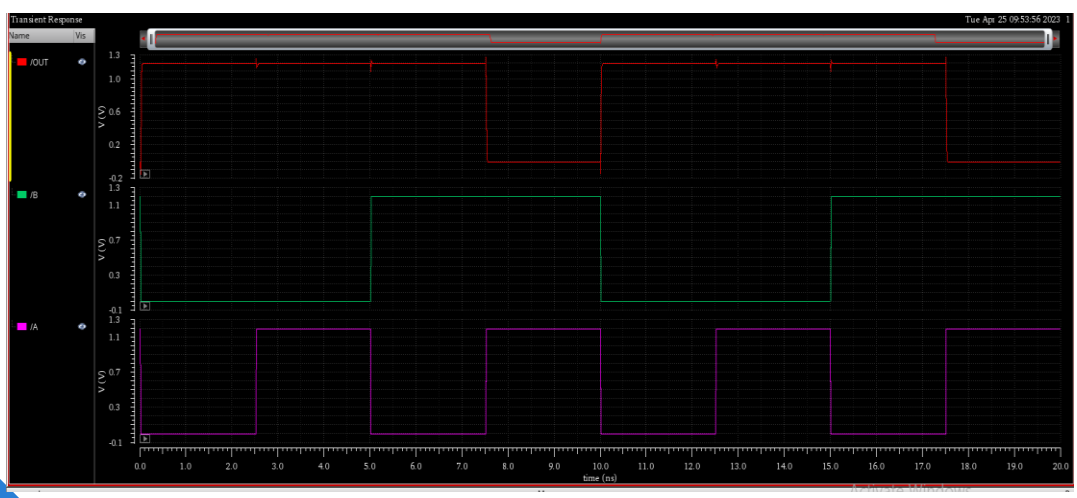


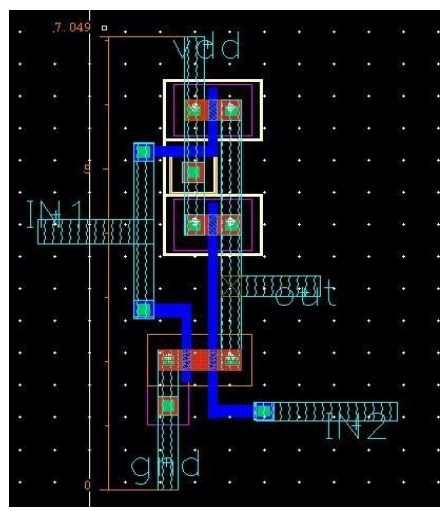
Figure: NAND gate schematic and symbol

Considering the reference inverter, we designed the NAND gate using Static CMOS Inverting logic family.

If $Wnref$ = 160nm and $WPref$ = 320nm, the widths for MOSFET Transistors in NAND gate are Wn = 2 $*$ $Wnref$ = 2 $*$ 160 = 320nm and WP = $WPref$ = 320nm.

Then we tested the results for all possible inputs (square waves) as shown in figure and table below

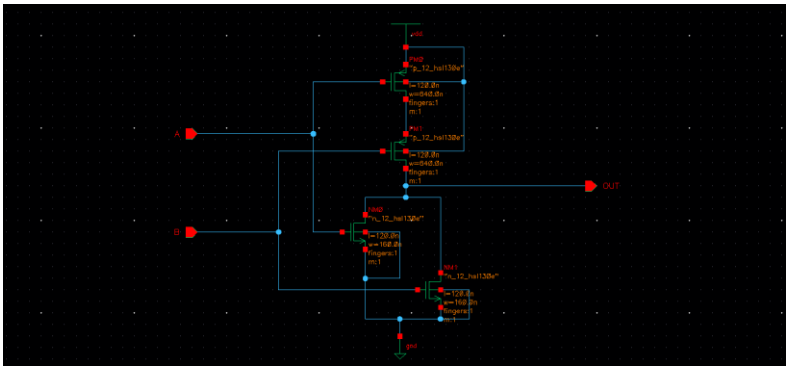




NAND layout **"Bonus"**

A	0	0	1	1
B	0	1	0	1
Out	1	1	1	0

NOR Gate



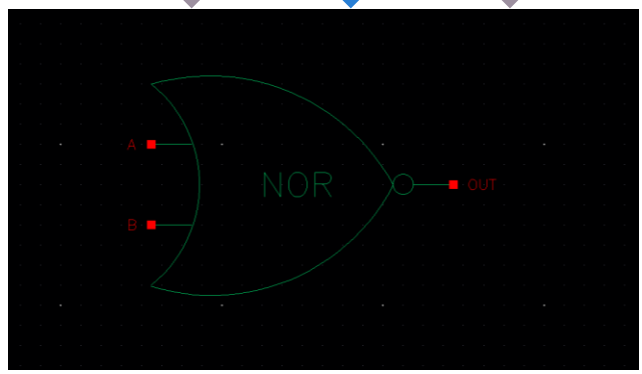
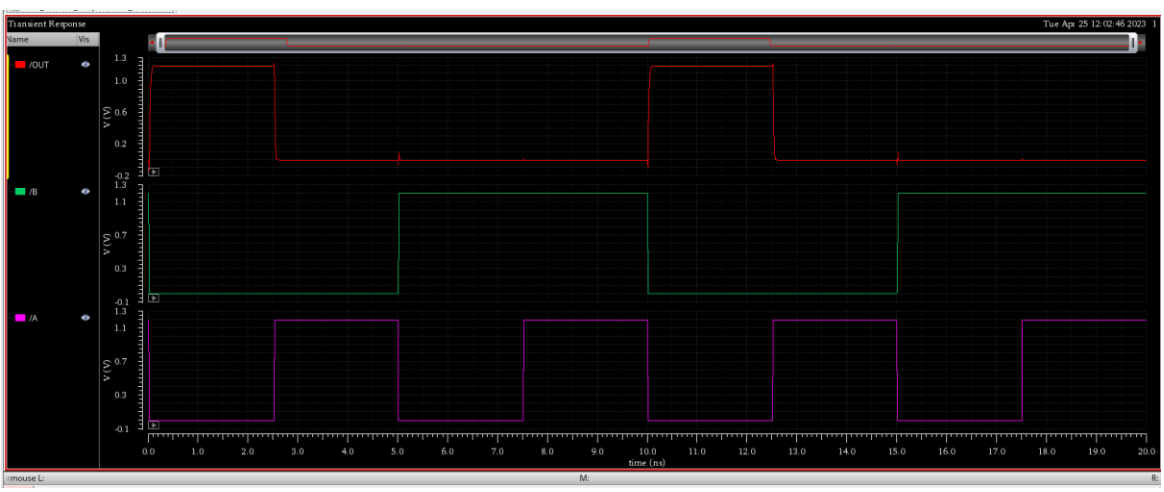
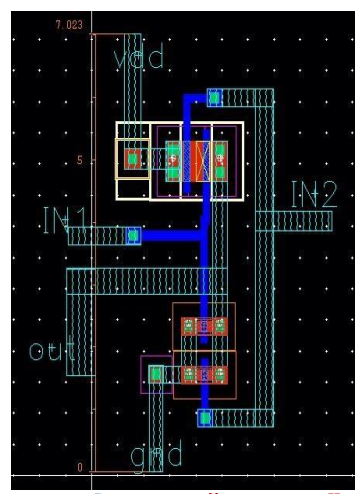


Figure: NOR gate schematic and symbol

Considering the reference inverter, we designed the NOR gate using Static CMOS Inverting logic family. If $Wnref$ = 160nm and $WPref$ = 320nm, the widths for MOSFET Transistors in NOR gate are Wn = $Wnref$ = 160nm and WP = 2 $*$ $WPref$ = 2 $*$ 320 = 640nm. Then we tested the results for all possible inputs (square waves) as shown in figure and table below

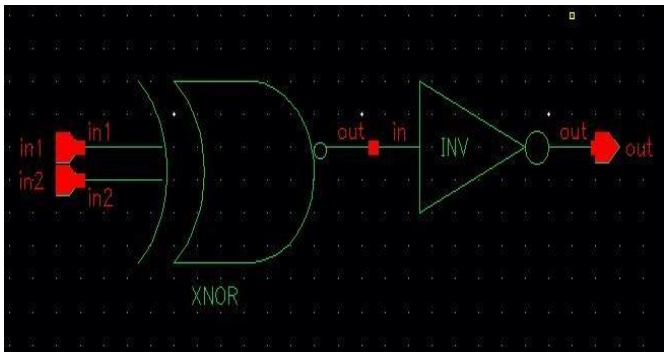




NOR layout **"Bonus"**

A	0	0	1	1
B	0	1	0	1
Out	1	0	0	0

XOR Gate



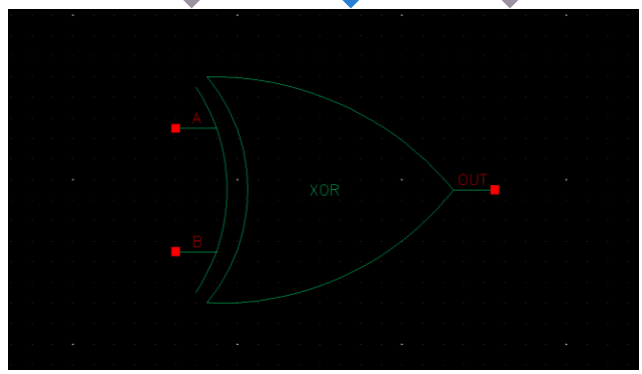
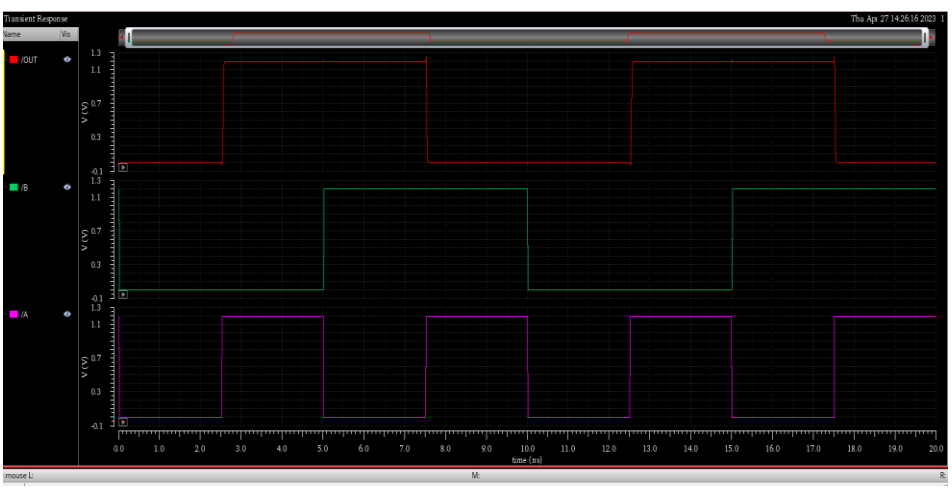
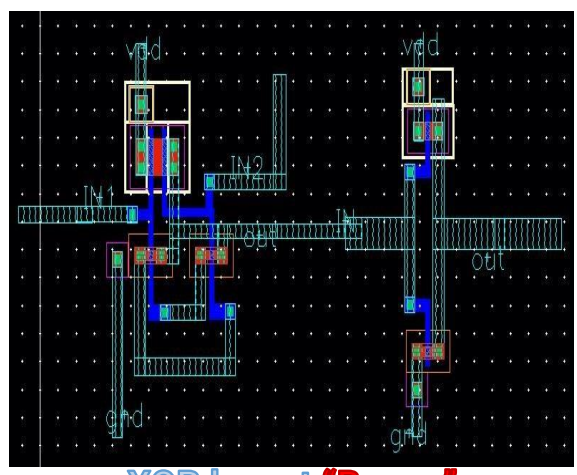


Figure: XOR gate schematic and symbol

Considering the reference inverter for input A, we designed the XOR using XNOR followed by inverter . Then we tested the results for all possible inputs (square waves) as shown in figure and table below

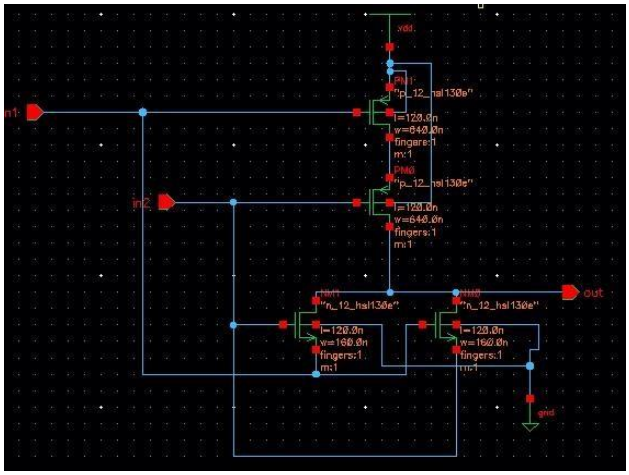




XOR layout **"Bonus"**

A	0	0	1	1
B	0	1	0	1
Out	0	1	1	0

XNOR Gate



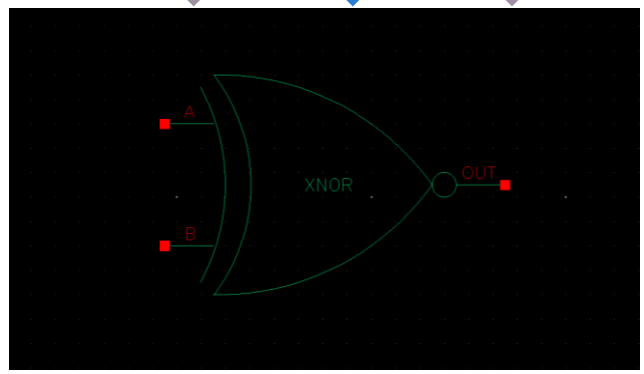
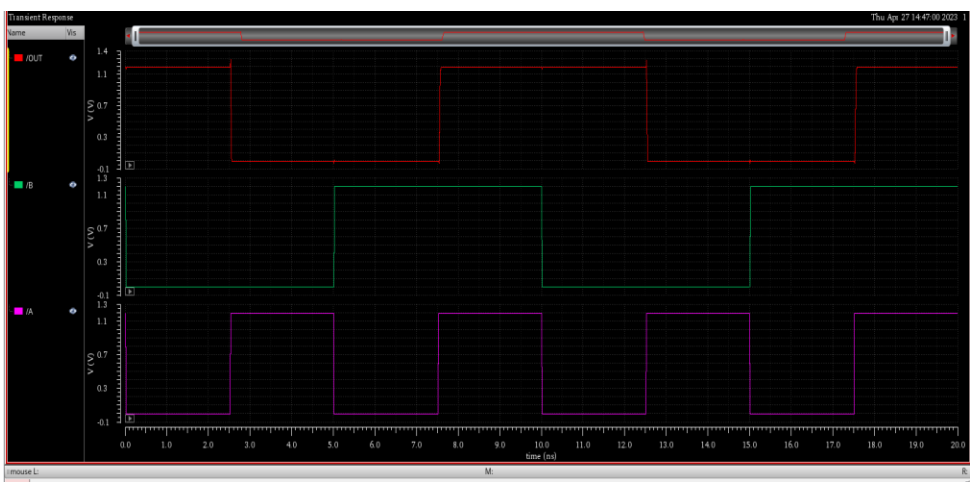
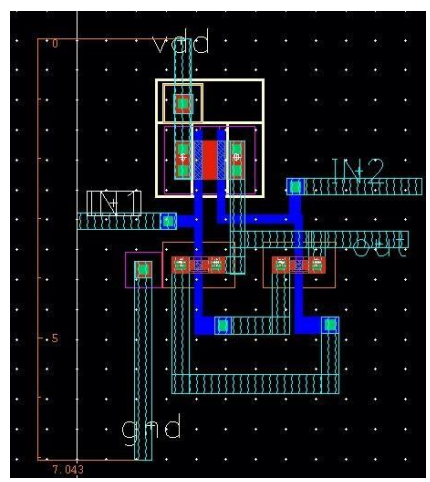


Figure: XNOR gate schematic and symbol

Considering the reference inverter for input A, we designed the XNOR gate using PTL family. If $Wnref$ = 160nm and $WPref$ = 320nm, the widths for MOSFET Transistors in XNOR gate are Wn = $Wnref$ = 160nm and WP = 2*$WPref$ = 640nm .Then we tested the results for all possible inputs (square waves) as shown in figure and table below

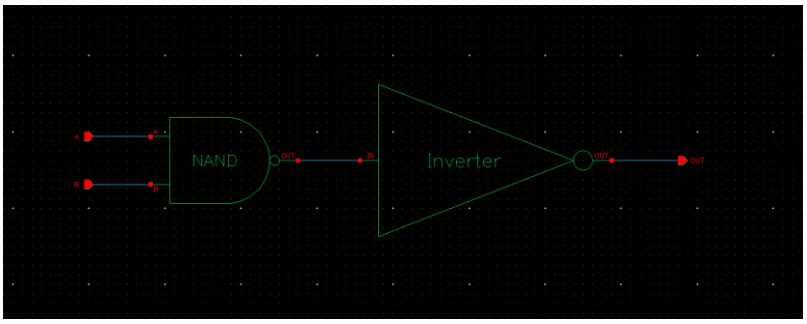




XNOR layout **"Bonus"**

A	0	0	1	1
B	0	1	0	1
Out	1	0	0	1

AND Gate



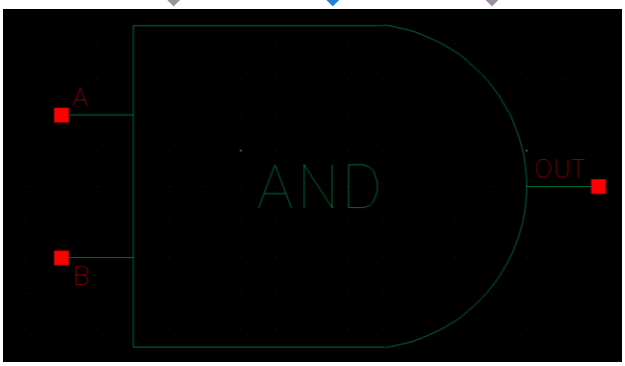
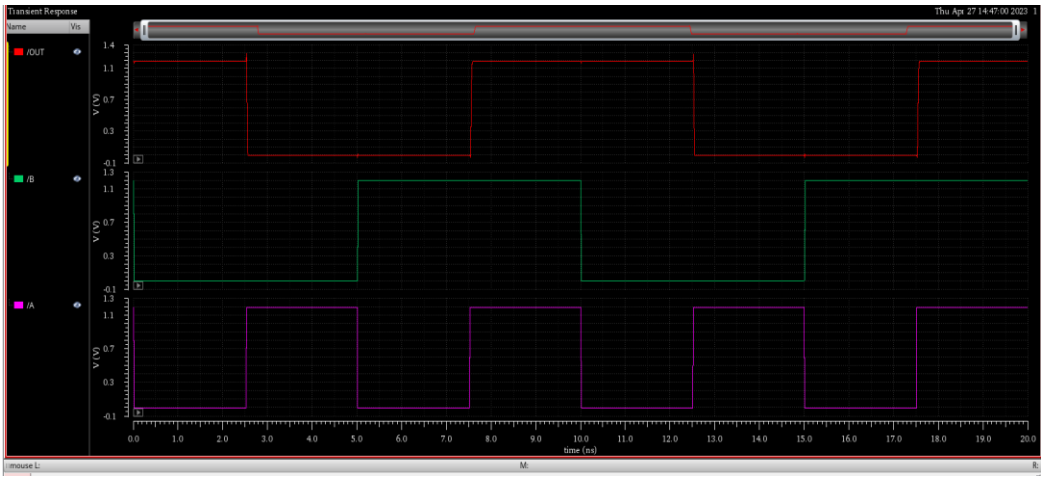
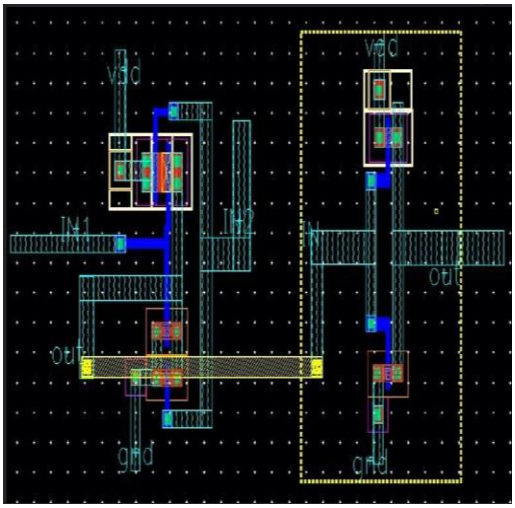


Figure: AND gate schematic and symbol

There was no need to implement the AND gate from scratch. Simply, it's a NAND gate followed by an inverter. These are the results for testing all possible inputs (square waves) as shown in figure and table below.





AND layout **"Bonus"**

A	0	0	1	1
B	0	1	0	1
Out	0	0	0	1

Buffer

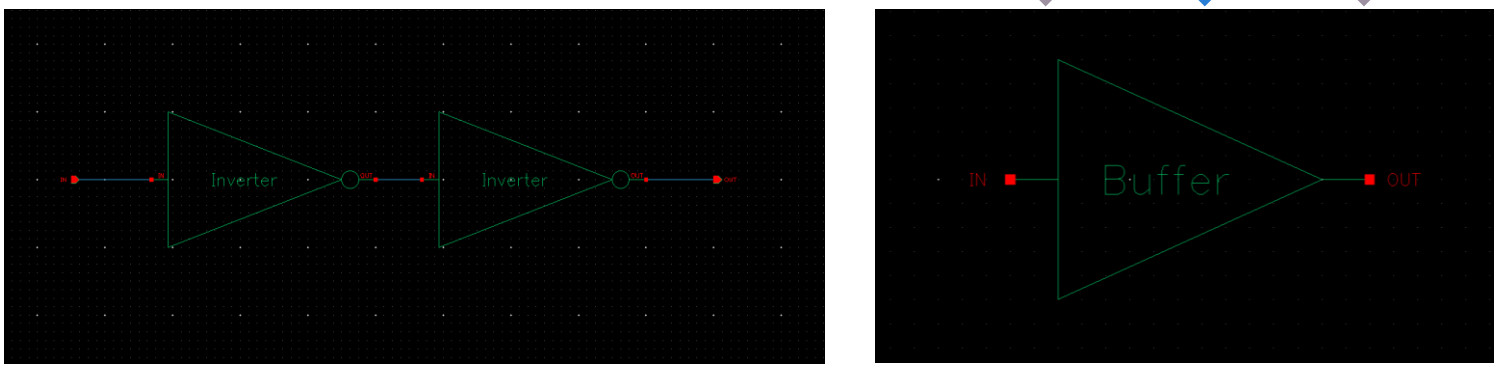
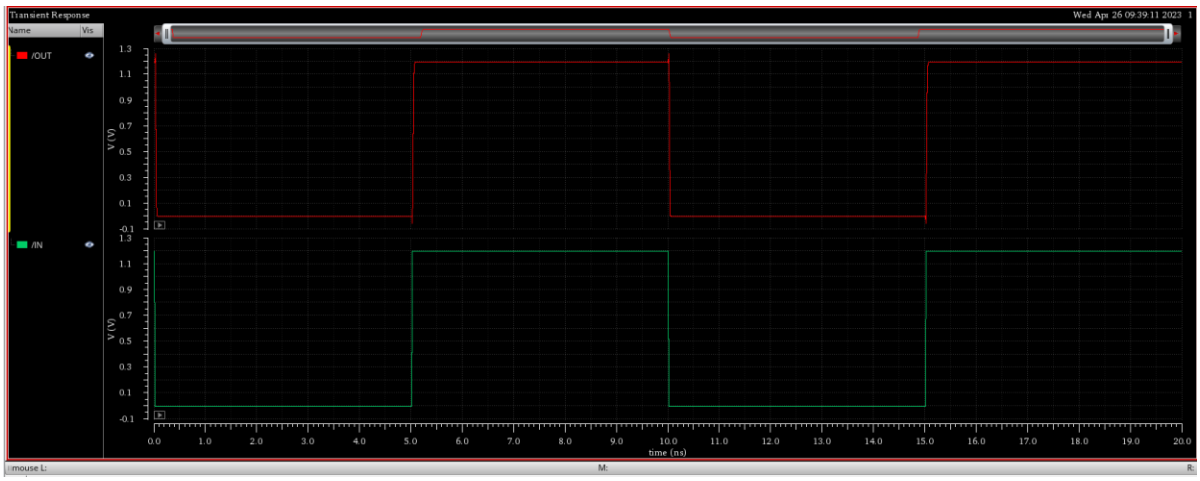


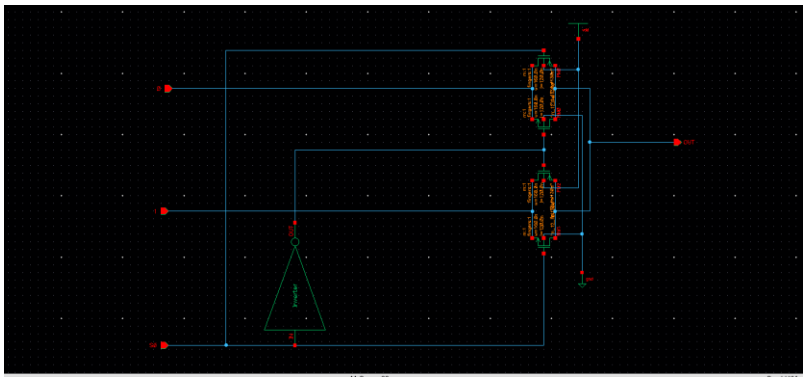
Figure: Buffer schematic and symbol

There was no need to implement the Buffer from scratch. Simply, it's an inverter followed by another inverter. These are the results for testing all possible inputs (square waves) as shown in figure and table below.



IN	OUT
1	1
0	0

2x1 MUX



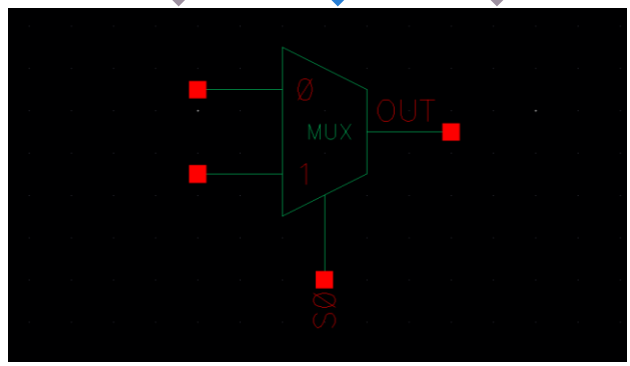
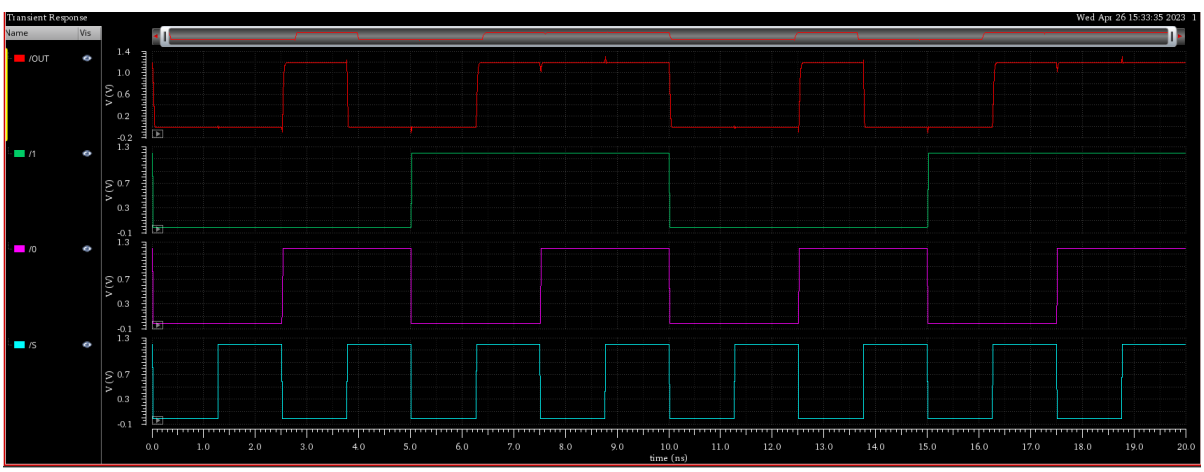


Figure: 2x1 MUX schematic and symbol

We used PTL family with minimum sizing WP = Wn = $Wmin$ = 160nm for shorter delay time to implement a 2X1 MUX selecting the operation from the left inputs depending on the bottom selector pin voltage (high → out = whatever on pin 1, low → out = whatever on pin 0). Then we tested the results for all possible inputs (square waves) as shown in figure and table below



S	0	0	0	0	1	1	1	1
/0	0	0	1	1	0	0	1	1
/1	0	1	0	1	0	1	0	1
Out	0	0	1	0	0	1	1	1

4x1 MUX

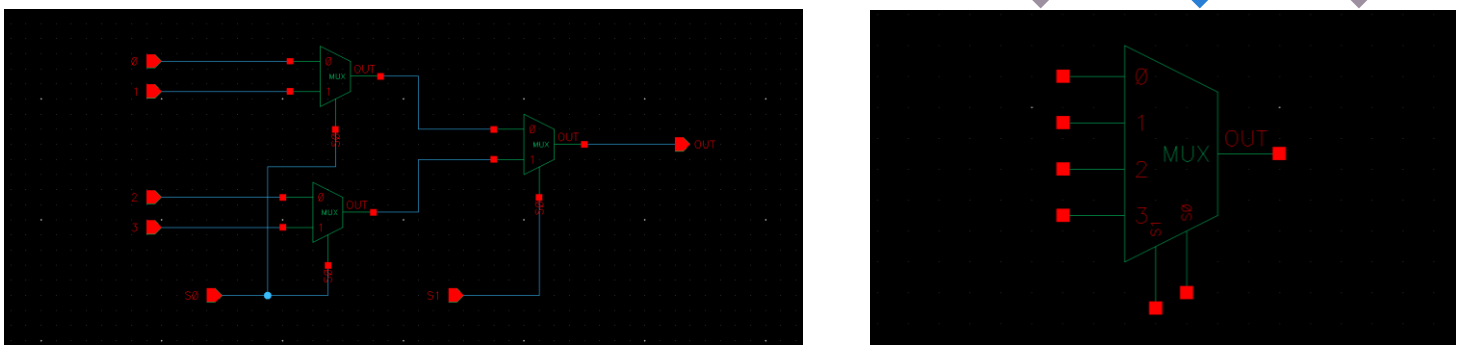


Figure: 4x1 MUX schematic and symbol

It's really no need to have 4X1 MUX but we prepared it to simplify implementing 8X1 MUX. Its functionality is to select one of four operations at the left pins by the 2 selectors at the bottom pins.

8x1 MUX

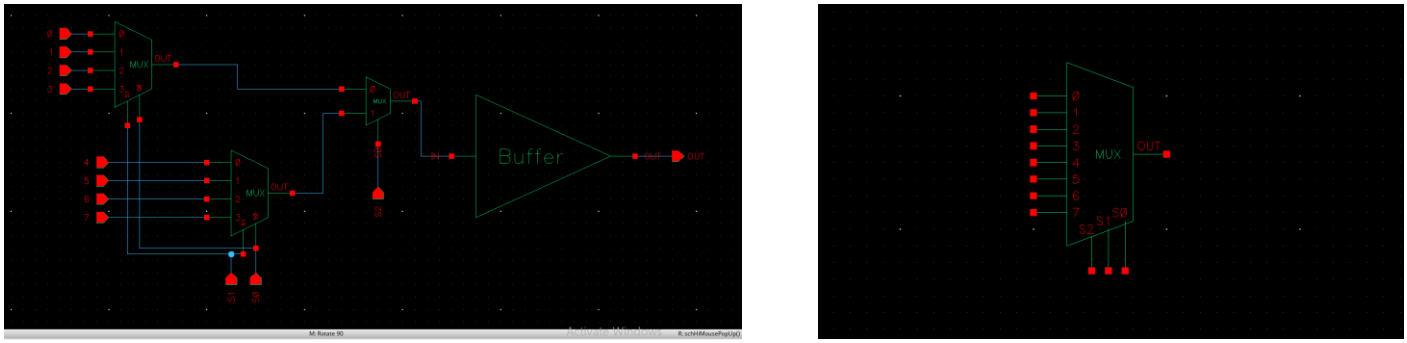
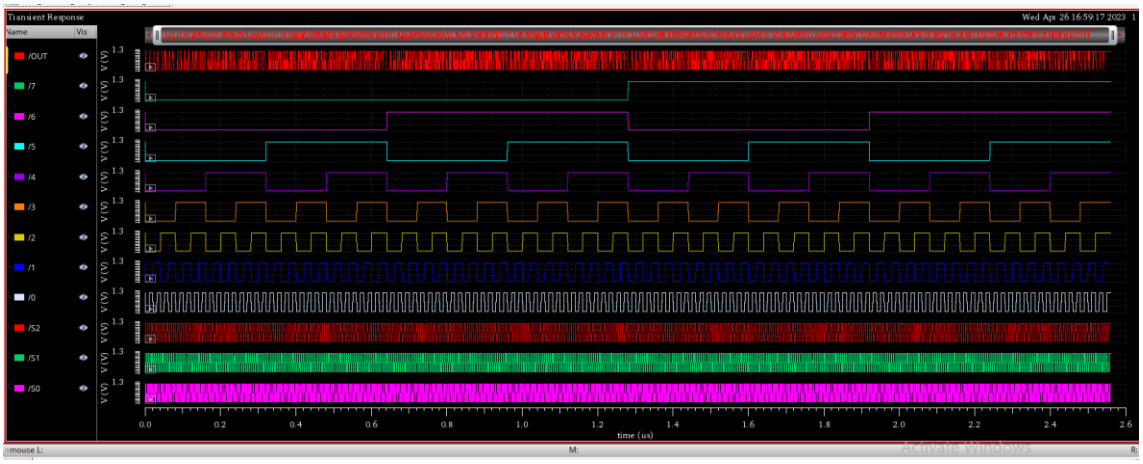
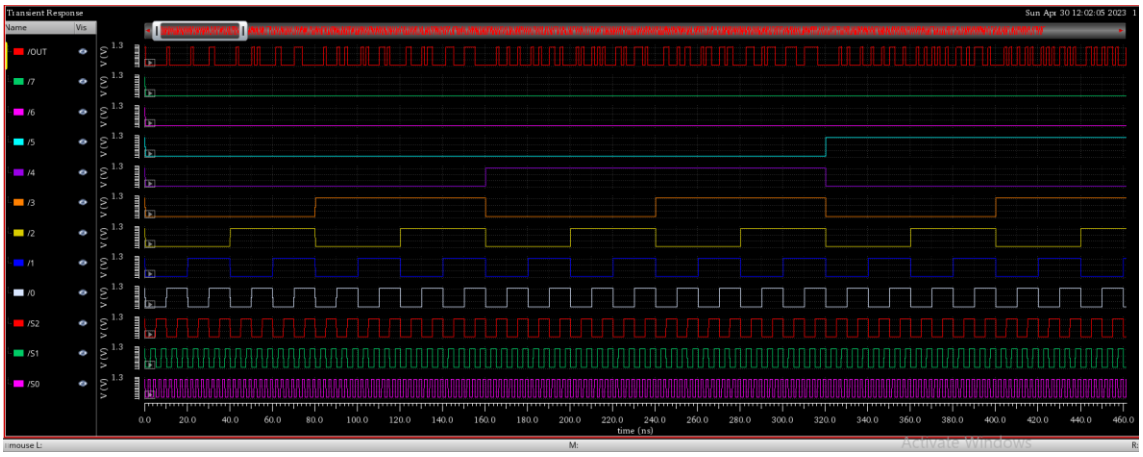


Figure: 8x1 MUX schematic and symbol

This is the main selecting element in our design as it selects one of the eight operations of the Logical or Arithmetic Units to be on the output. Then we tested the results for all possible inputs (square waves) as shown in figure below





8x1 mux_tb

S1	S2	S3	OUT
0	0	0	/0
1	0	0	/1
0	1	0	/2
1	1	0	/3
0	0	1	/4
1	0	1	/5
0	1	1	/6
1	1	1	/7

Logical Unit

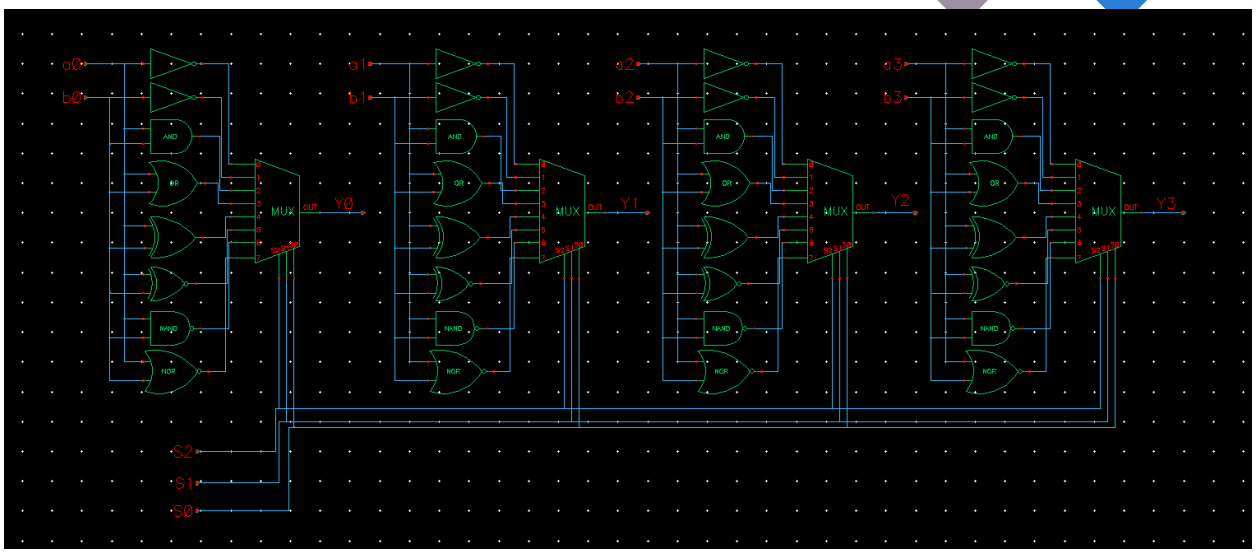


Figure: LU schematic

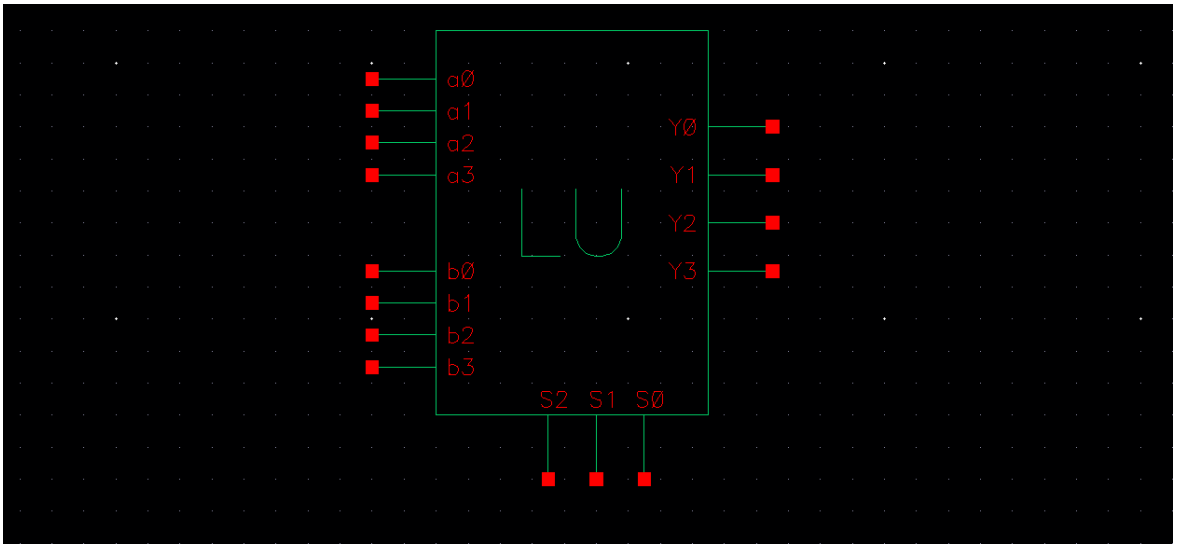


Figure: LU symbol

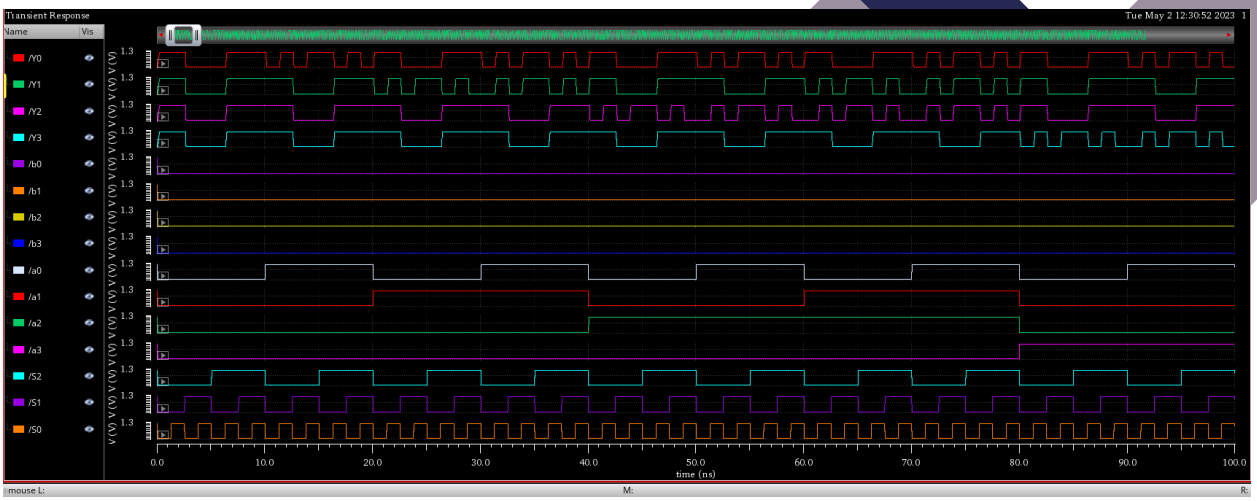
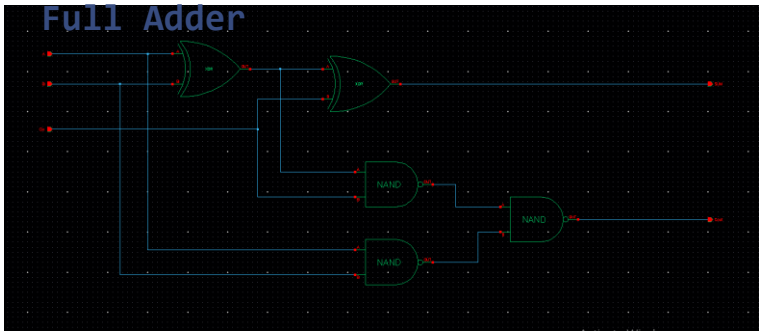


Figure: LU Transient Response



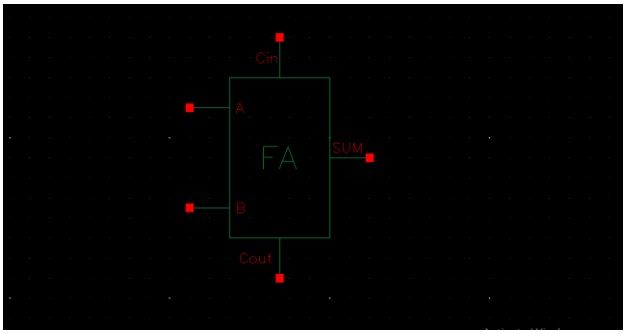
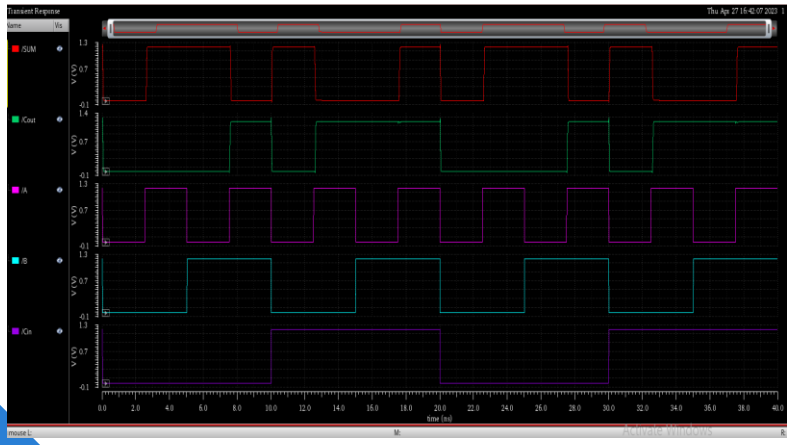
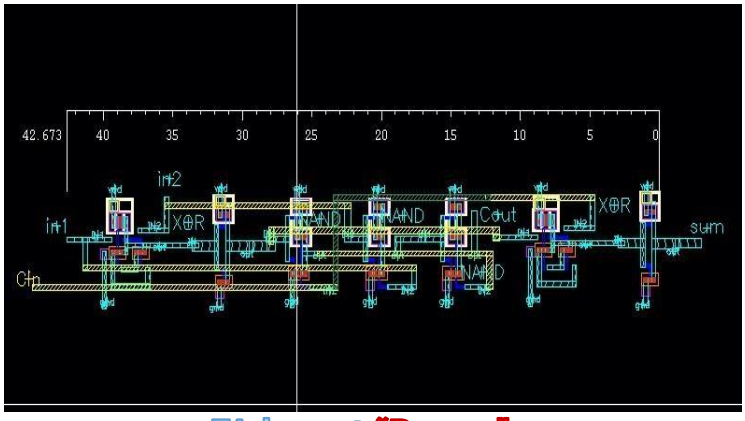


Figure: FA schematic and symbol

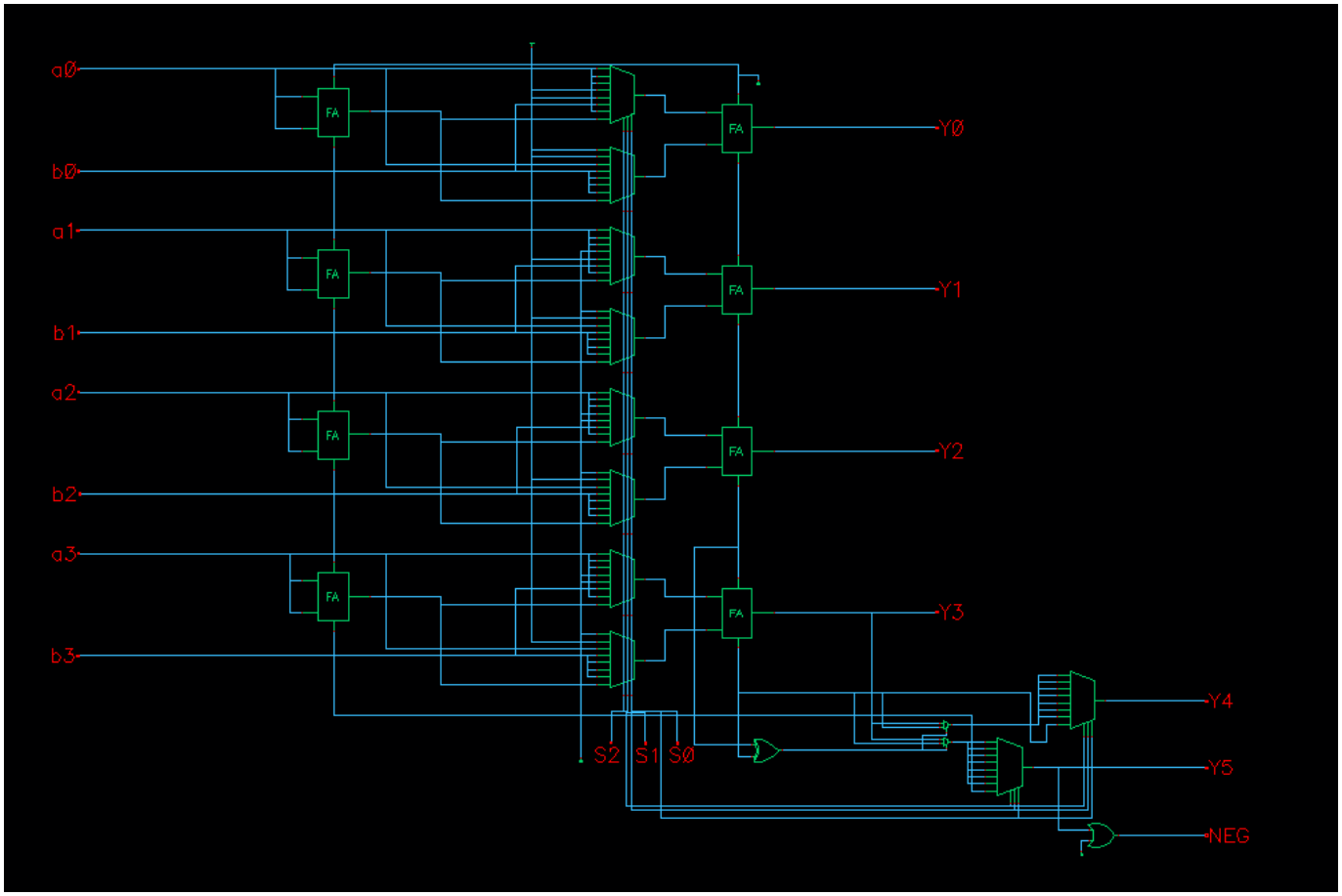




FA layout **"Bonus"**

Figure: FA Transient Response

Signed I/O Arithmetic Unit with a negative flag (BONUS)



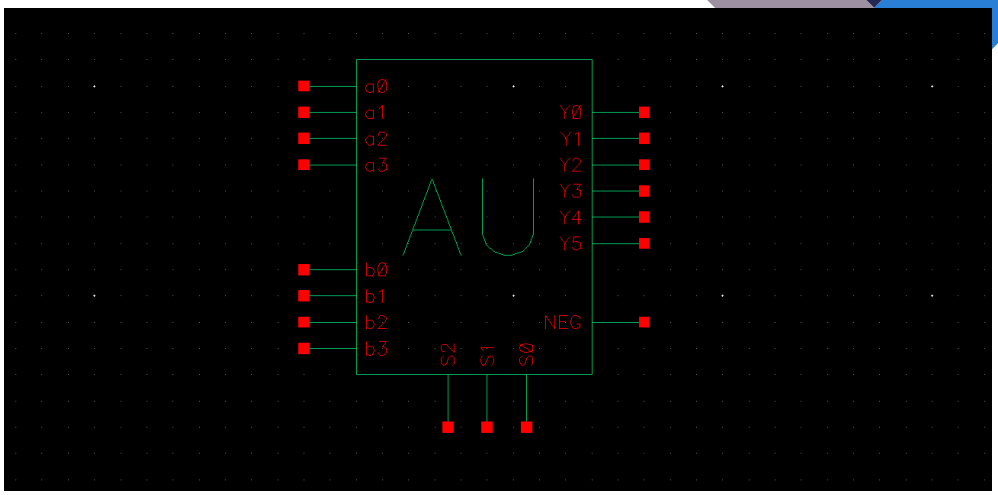


Figure: Arithmetic unit schematic and symbol

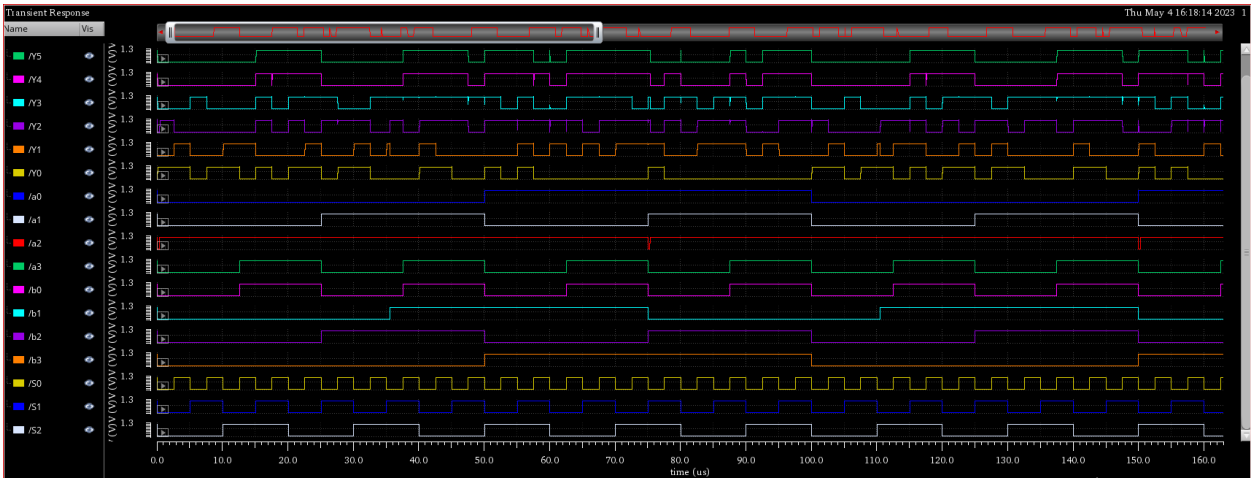
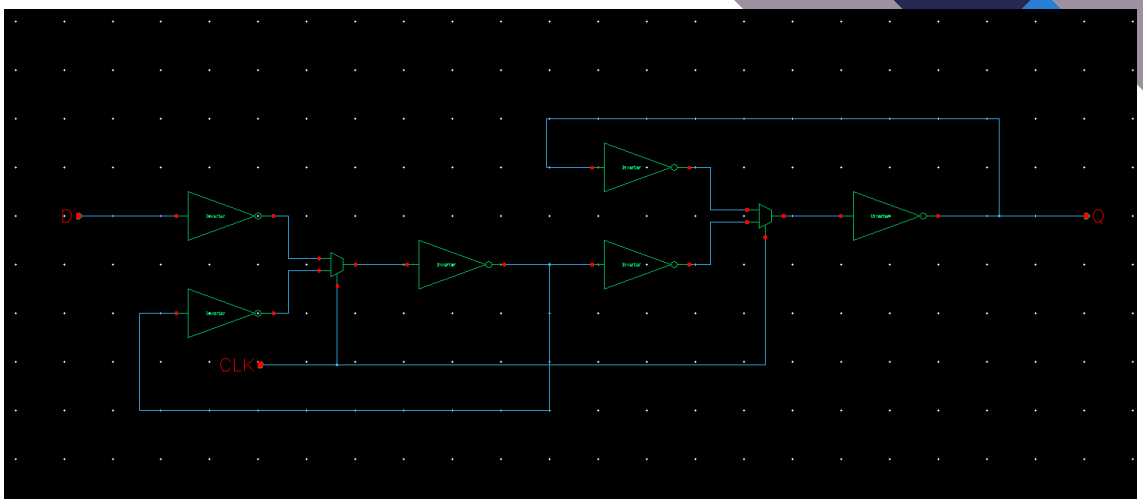
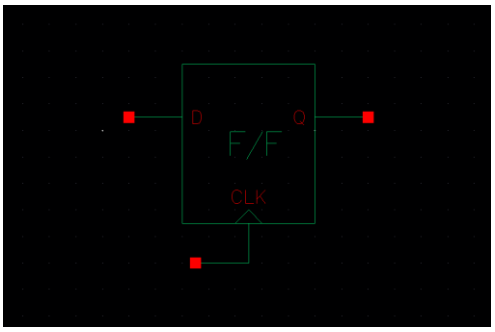


Figure: Arithmetic unit TB

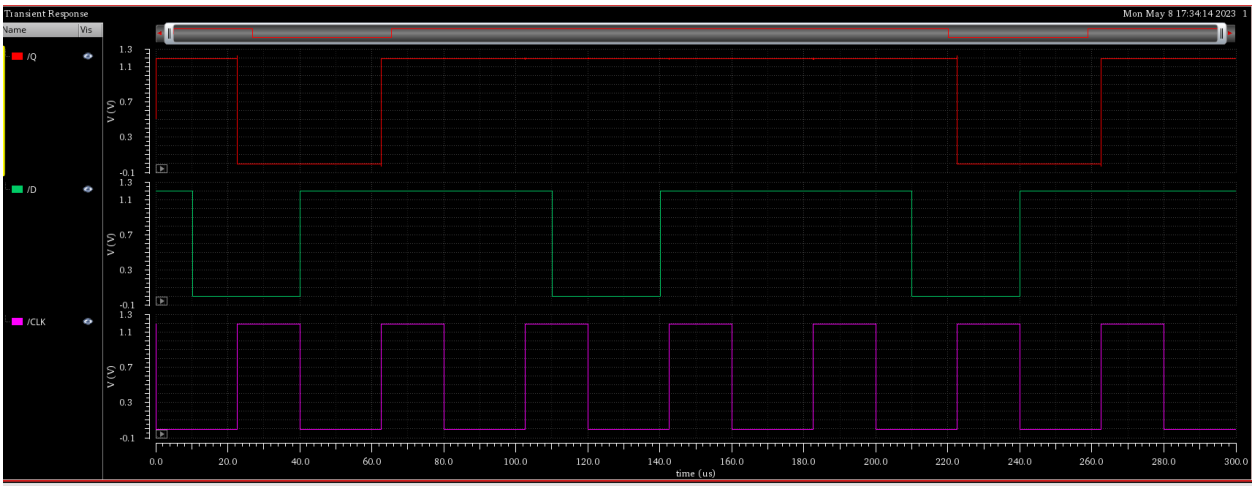
Flip Flop





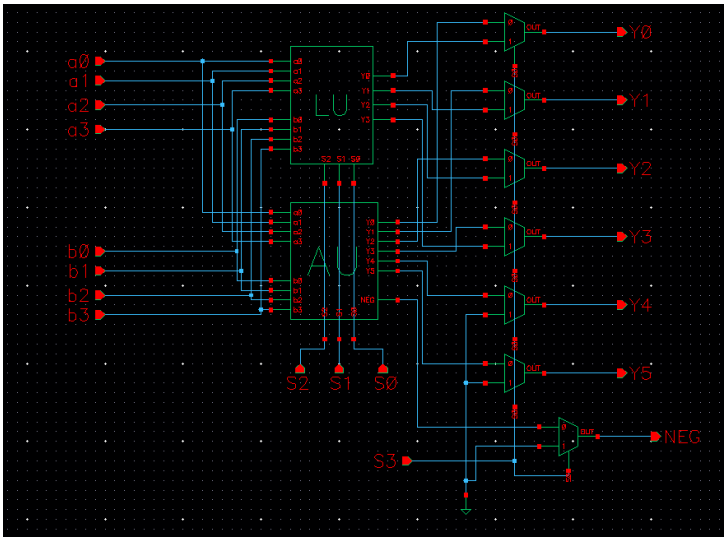
Figures: positive edge triggered D-type Flip/Flop

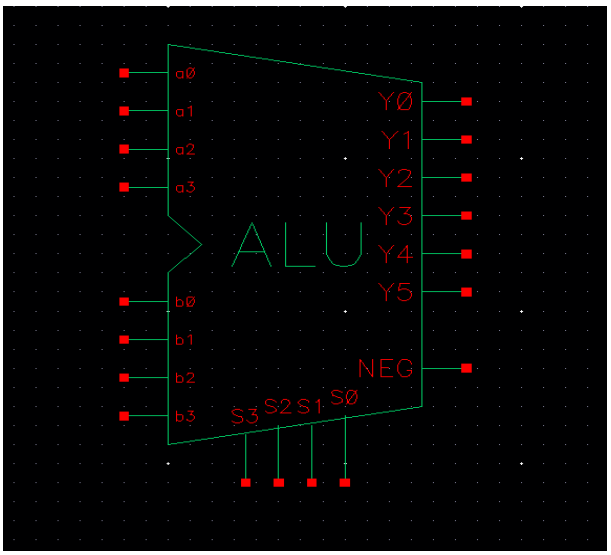
The flip flop is implemented using static design (2-mux based latches). We used Inverters after MUXs to buffer the output and restore levels. Then we tested the results for all possible inputs (square waves) as shown in figure and table below



testbench

Signed I/O ALU with a negative flag (BONUS)





Figures: 4-bit signed I/O ALU Schematic & symbol with negative flag

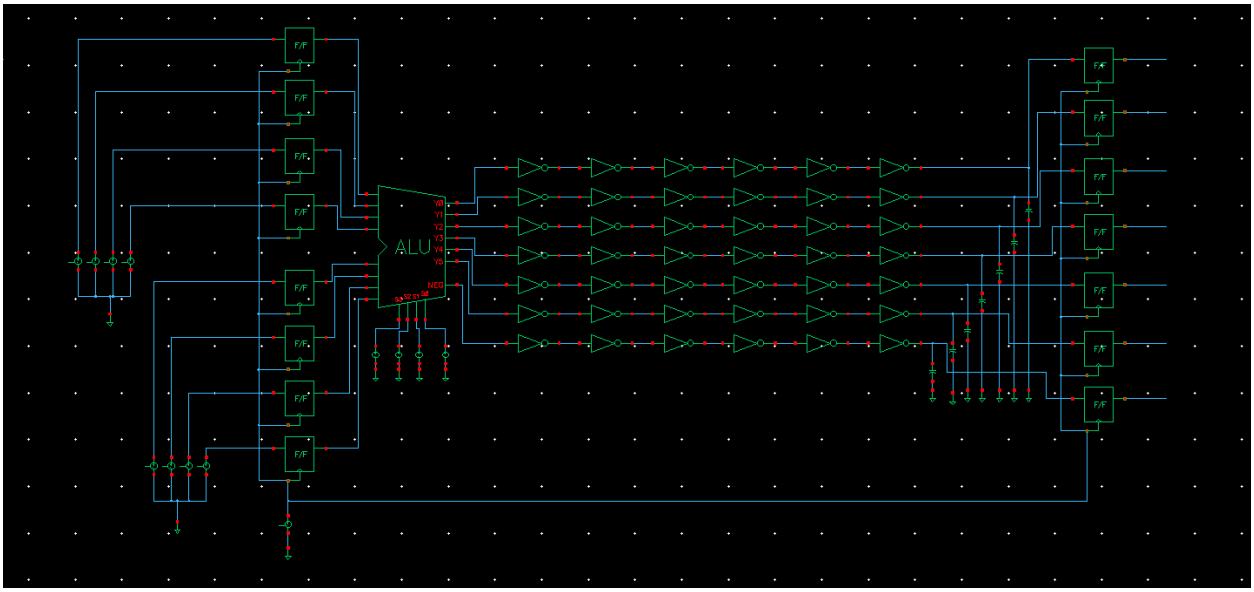
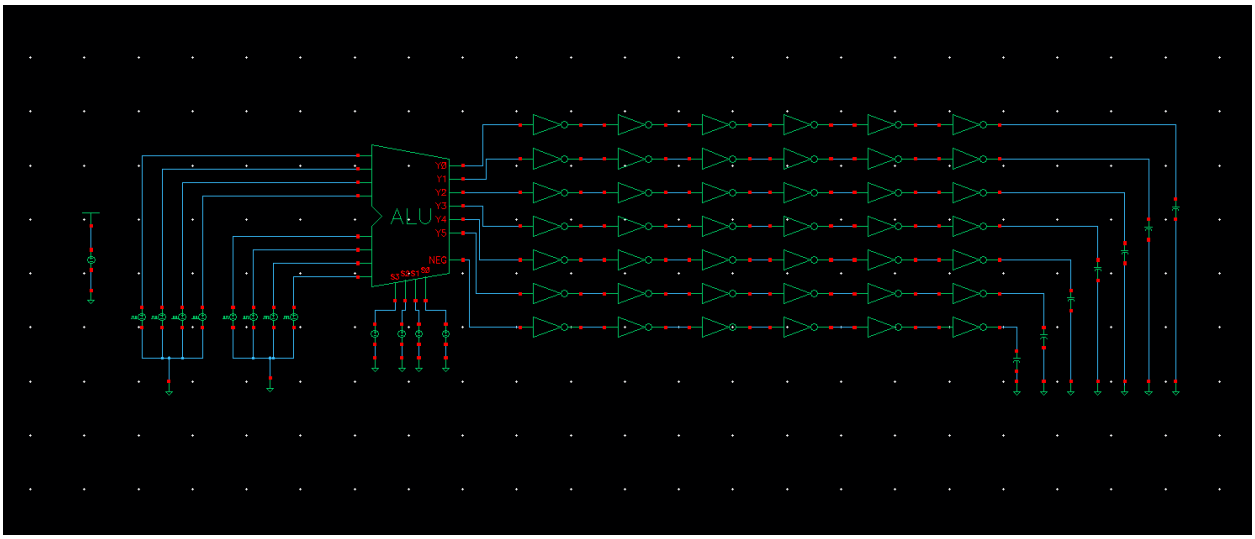


Figure: final design of 4-bit-ALU with f/f and FO4 (chain of inverters).

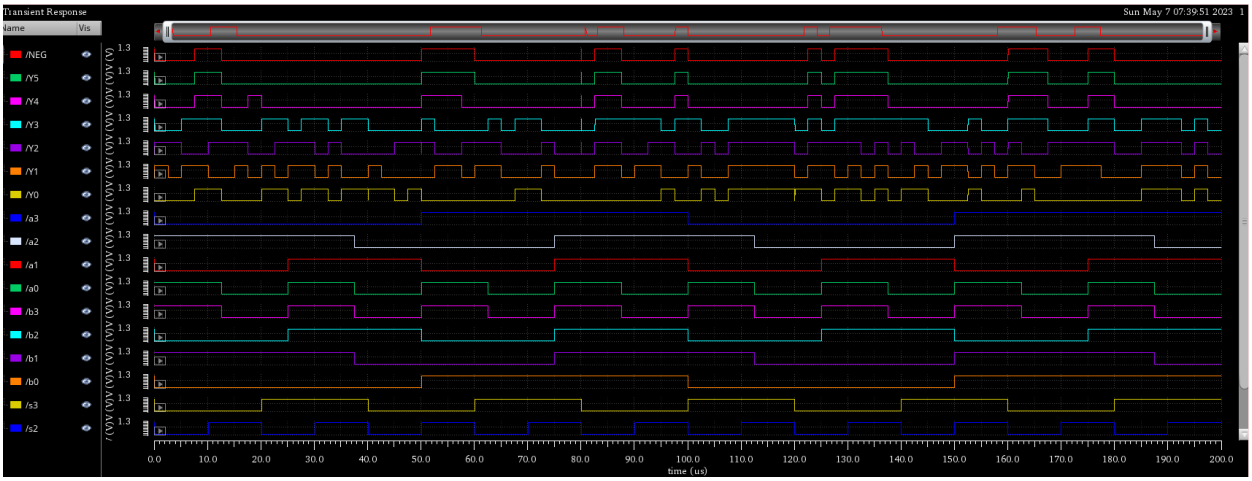
At the end, we assembled the LU, AU, MUXs, Inverters and F/F to implement the 4-bit-ALU. As it's clearly shown inputs are the left pins: A0 A1 A2 A3 B0 B1 B2 B3, outputs are the right pins: Y0 Y1 Y2 Y3 Y4 Y5 NEG(negative flag), Selectors are the bottom pins: S0 S1 S2 S3 Then we tested the inputs .

Connecting 2pf capacitors to output

To prevent the massive delay caused by connecting the capacitors to the output, we used inverter sections with each section bigger than the previous one by 4 times (fanout of 4).



This figure shows ALU All operations.



testbench

Transient simulations of some operations of ALU with flip flop

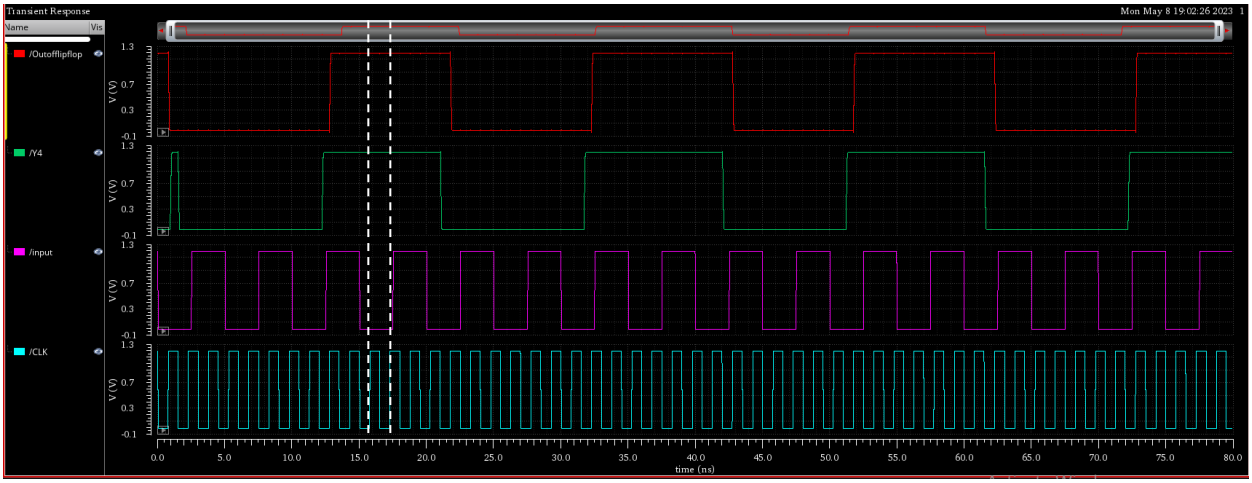


Figure: Random simulation to check outputs before flip flop and after flip flop at max. frequency

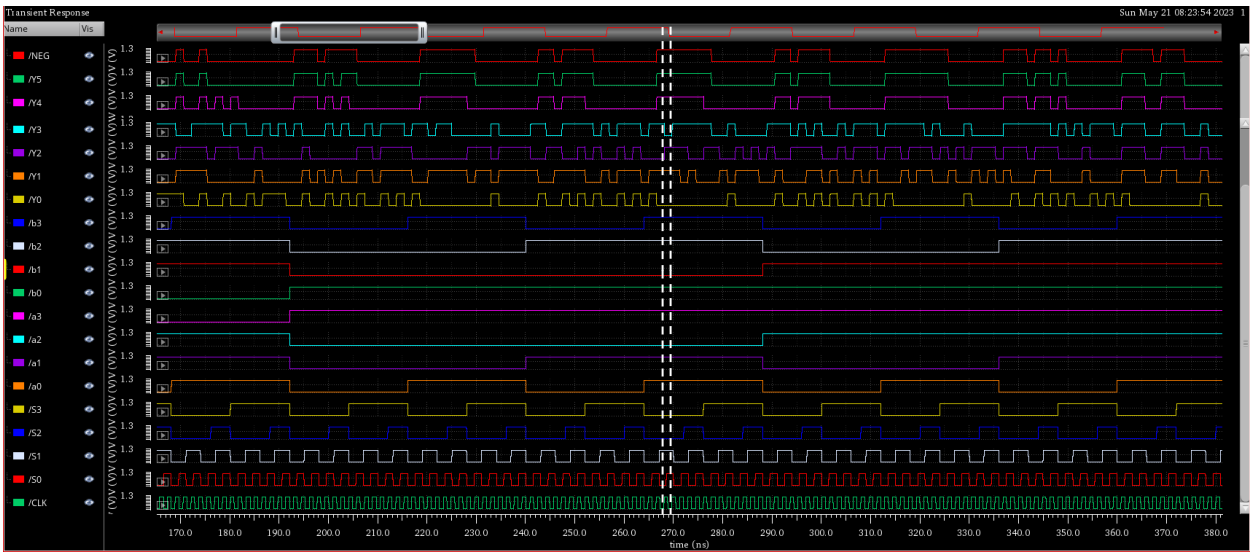


Figure: All operations of ALU with flip flops for all possible inputs

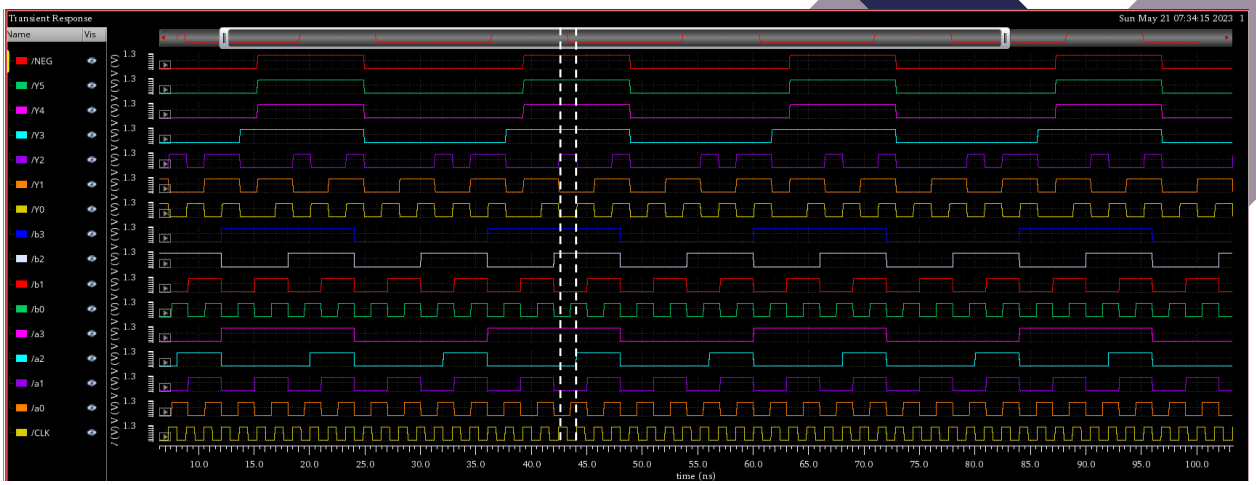


Figure: increment a selector: 0000

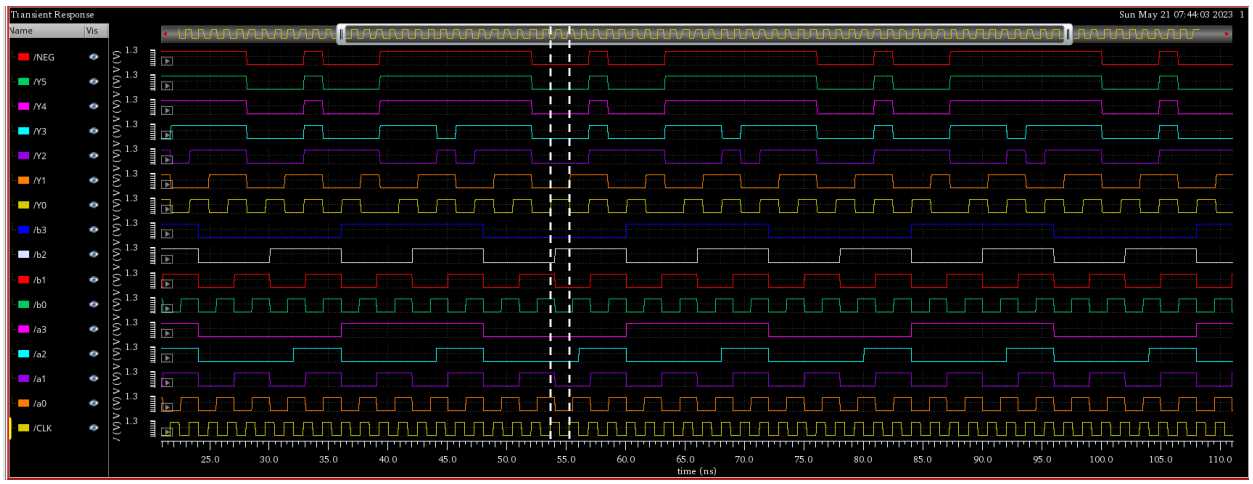


Figure: decrement a selector: 0001

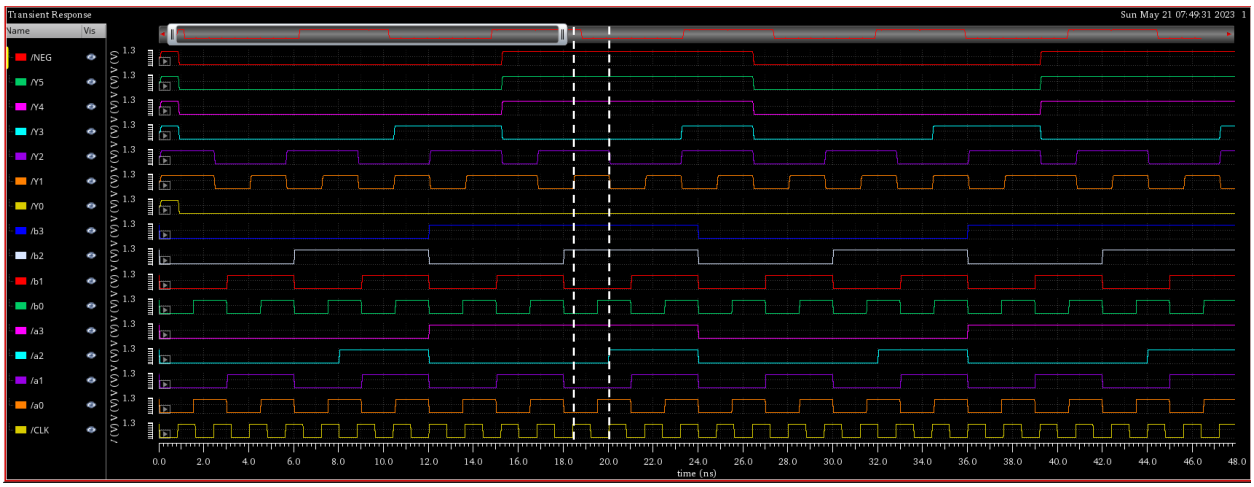


Figure: Multiply a*2 selector: 0010

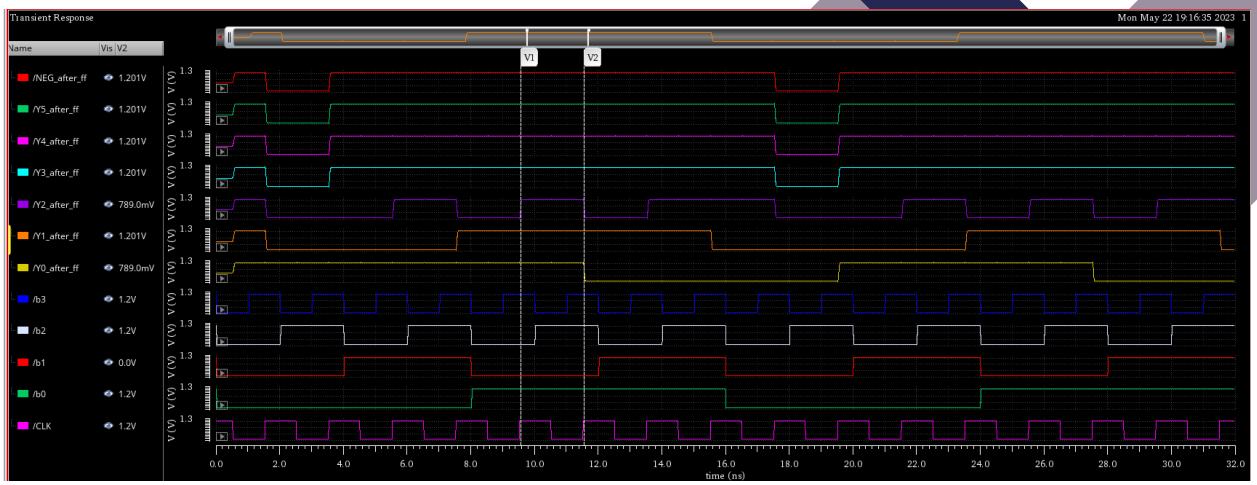


Figure: increment b selector: 0011

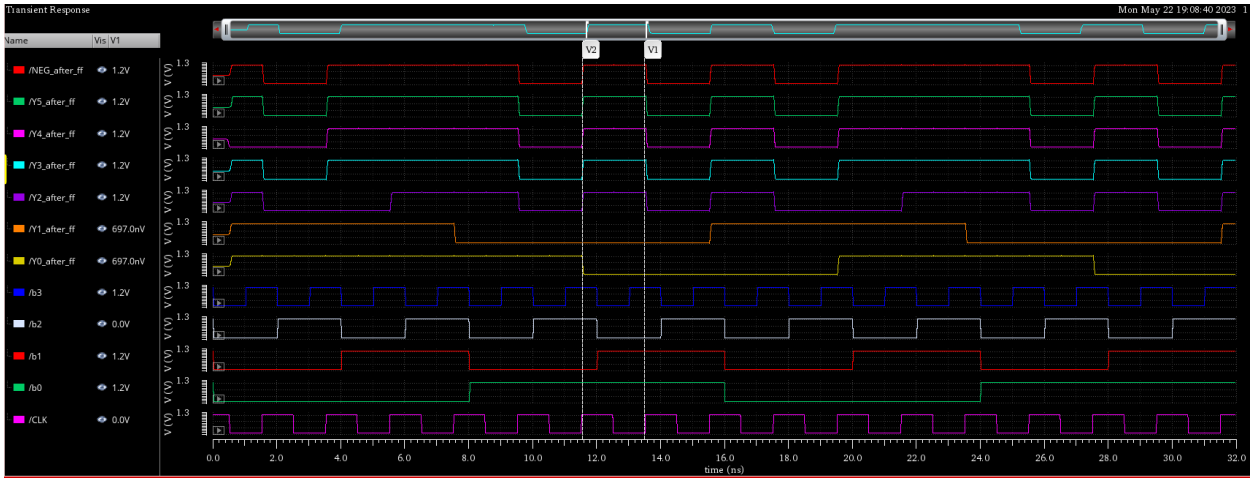


Figure: decrement b selector: 0100

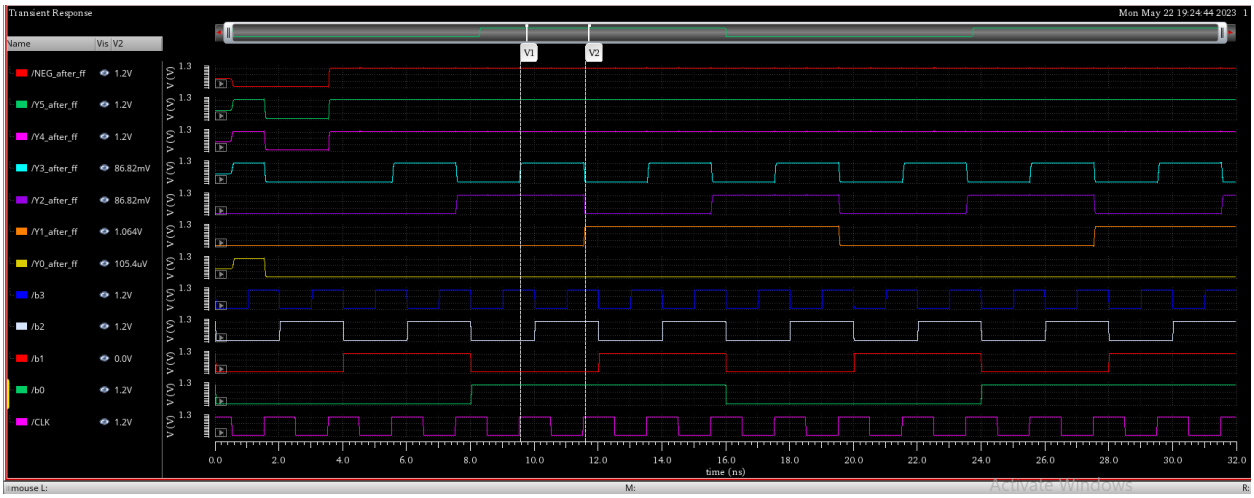


Figure: Multiply b*2 selector: 0101

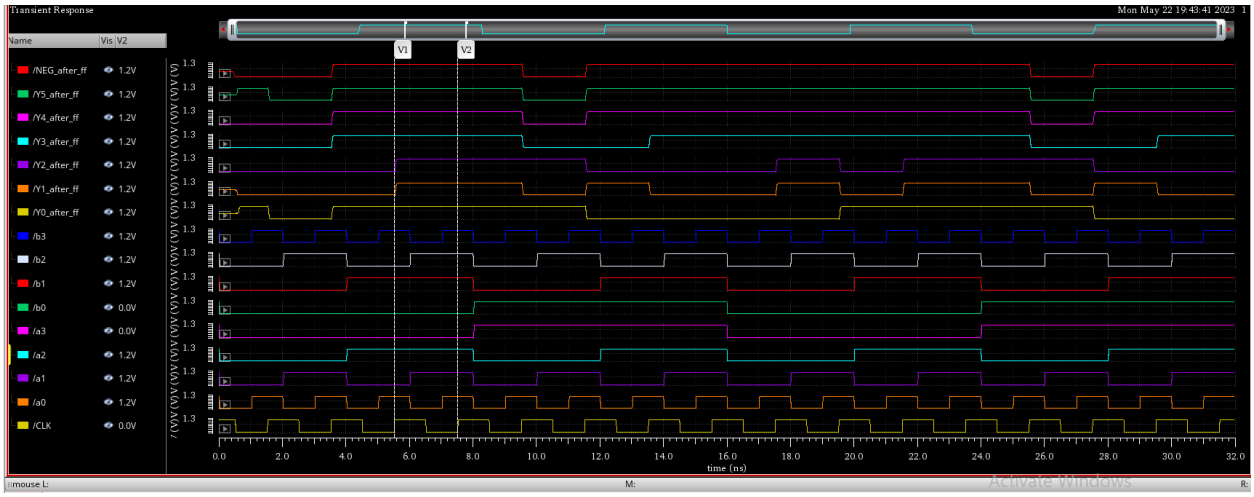


Figure: Add a and b selector: 0110

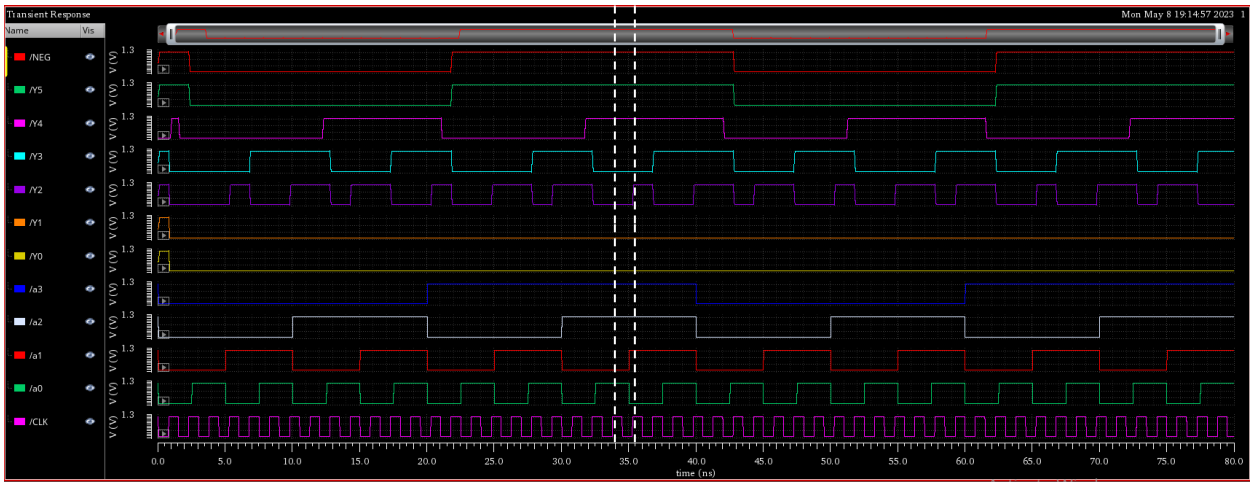


Figure: 4*a operations for inputs selector: 0111

Figure: complement(a) selector: 1000

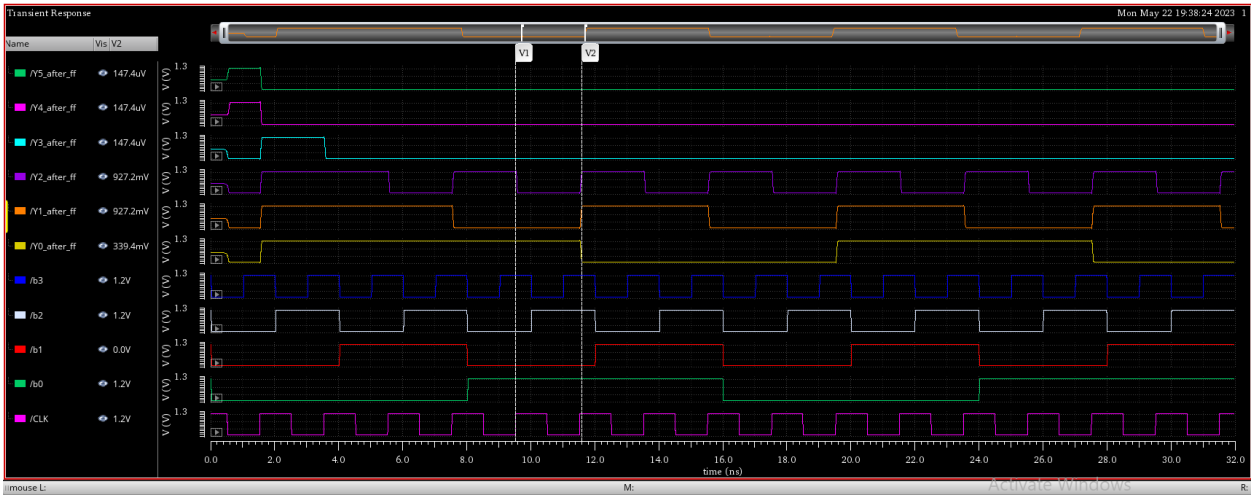


Figure: complement(b) selector: 1001

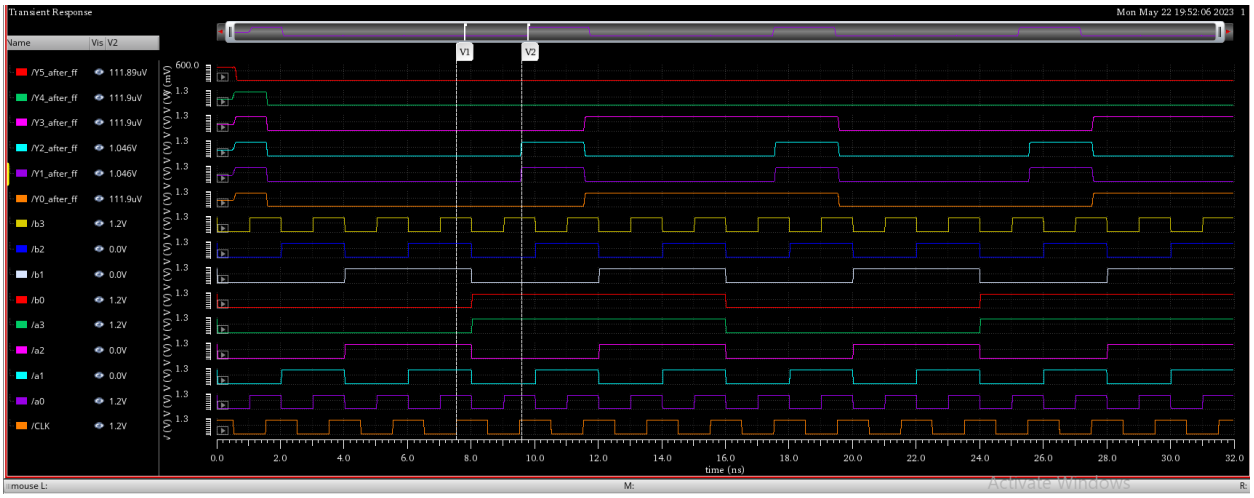


Figure: A & B selector: 1010

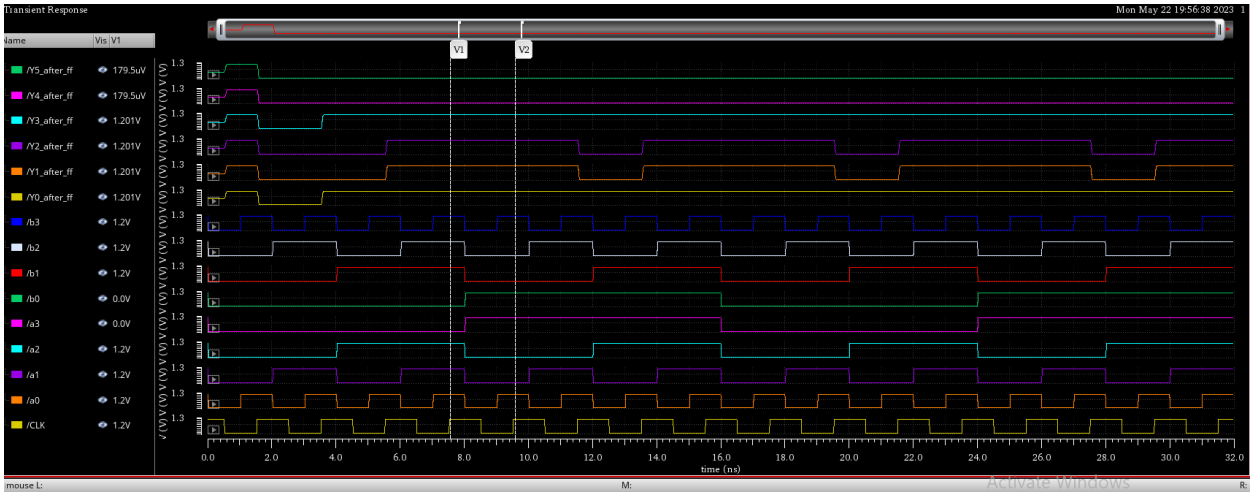


Figure: a or b selector: 1011

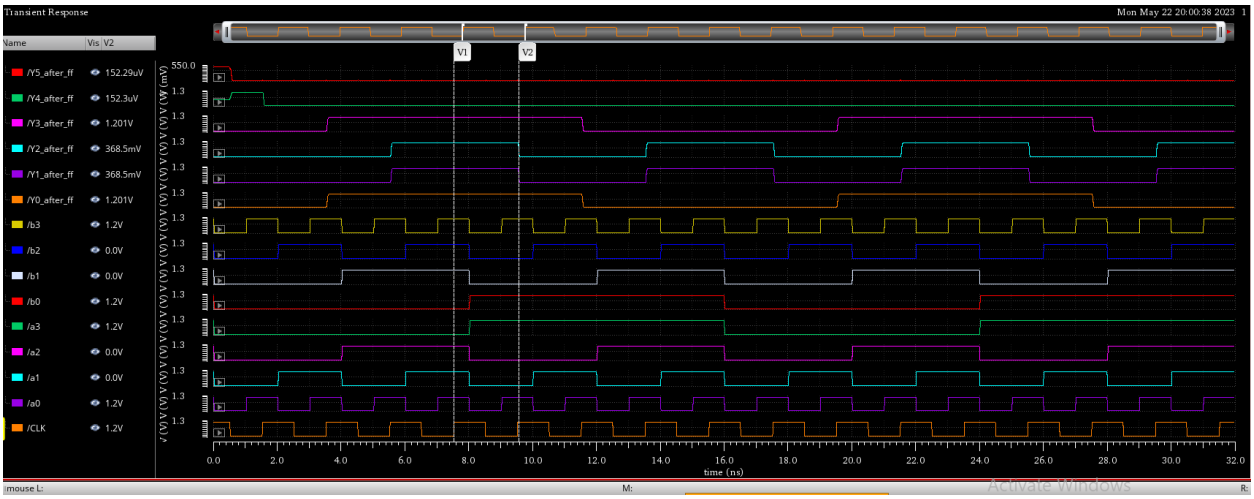


Figure: a xor b selector: 1100

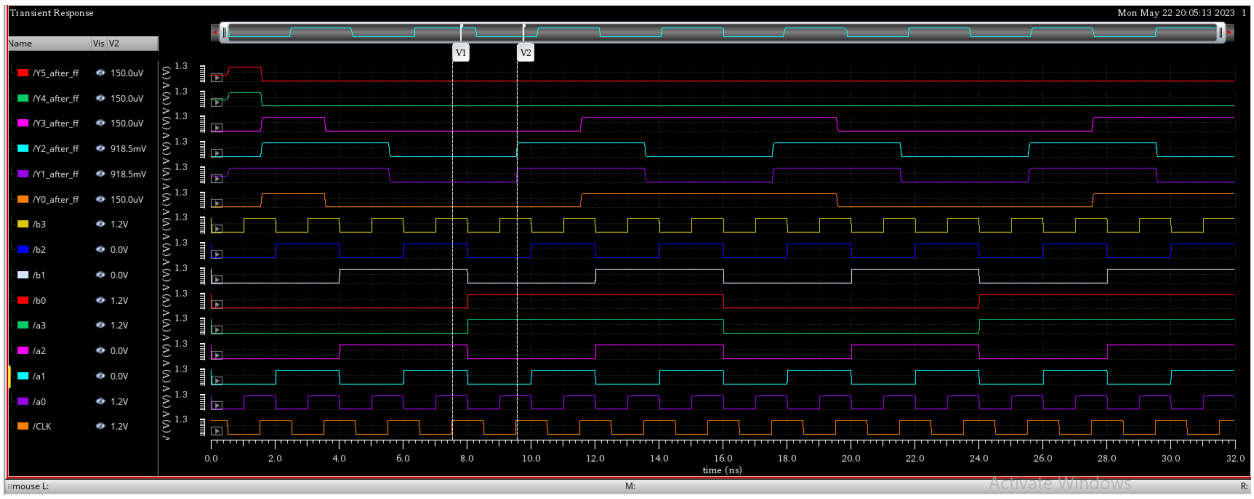


Figure: a xnor b selector: 1101

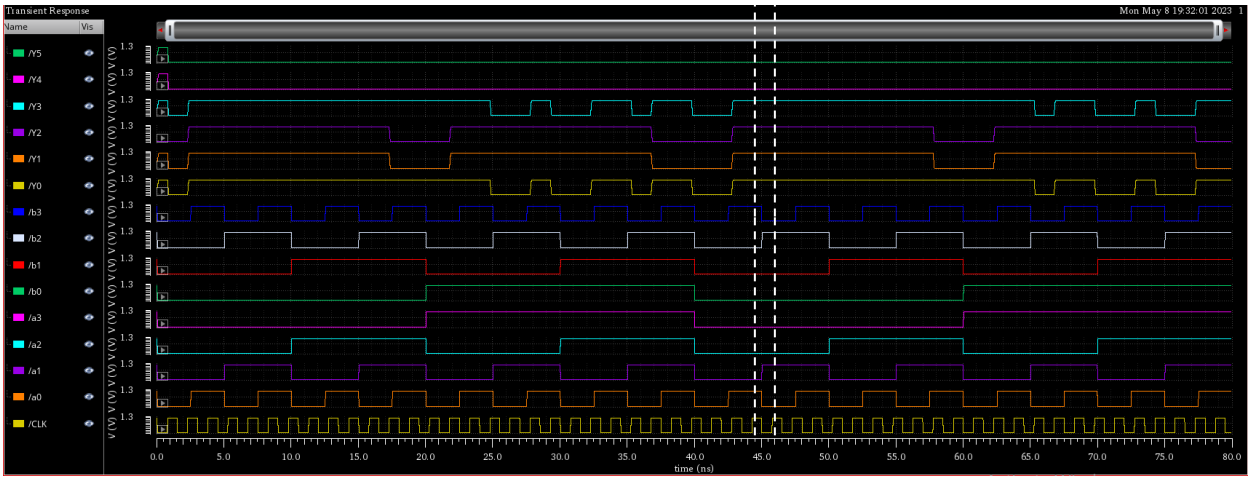


Figure: a NAND b operations for inputs selector: 1110

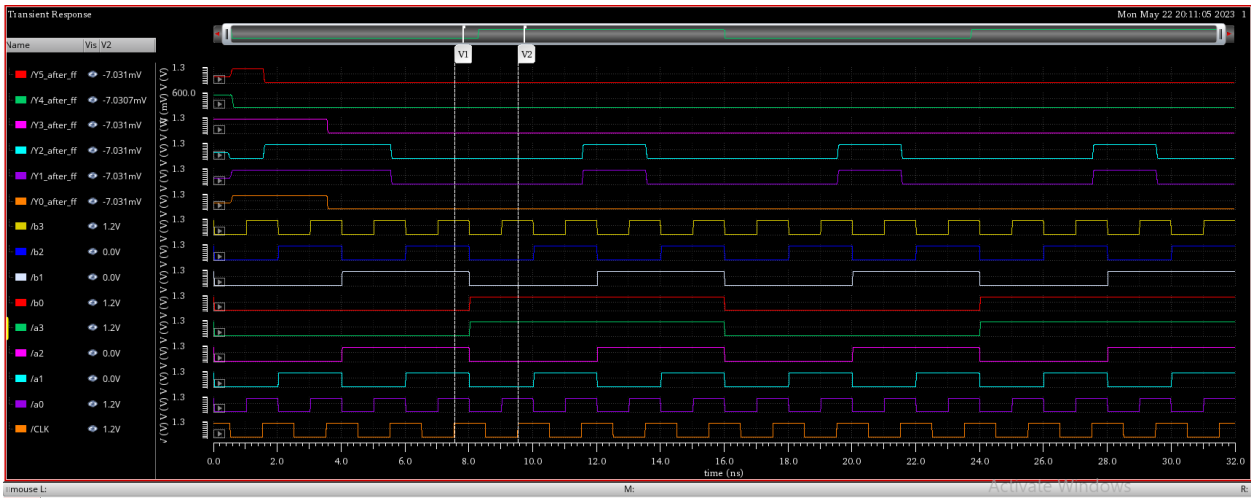


Figure: a nor b selector: 1111

Worst Case Delay:

In this part we are going to determine the delay needed for our circuit to work at the worst-case delay possible to get maximum frequency of operation. The worst-case delay is defined for the most complex operation or the operation that gives the last output The worst-case delay will be calculated for our Full adder which is also called Carry ripple adder and the carry is the last output which is calculated from our adder and our ALU to get the worst case delay we should calculate the delay between the input and the carry out. The orange curve is our input, and the green curve is our carry so by measuring the delay from 600mv of input to 600mv of output we get that the Worst case is delay is about 700ps. $t_{propagation} = 1.4ns$

$$t_{clock} = t_{propagation} + t_{setup} + tcq \approx 2ns$$

$$f_{max} = \frac{1}{2ns} = 0.5\ GHZ$$

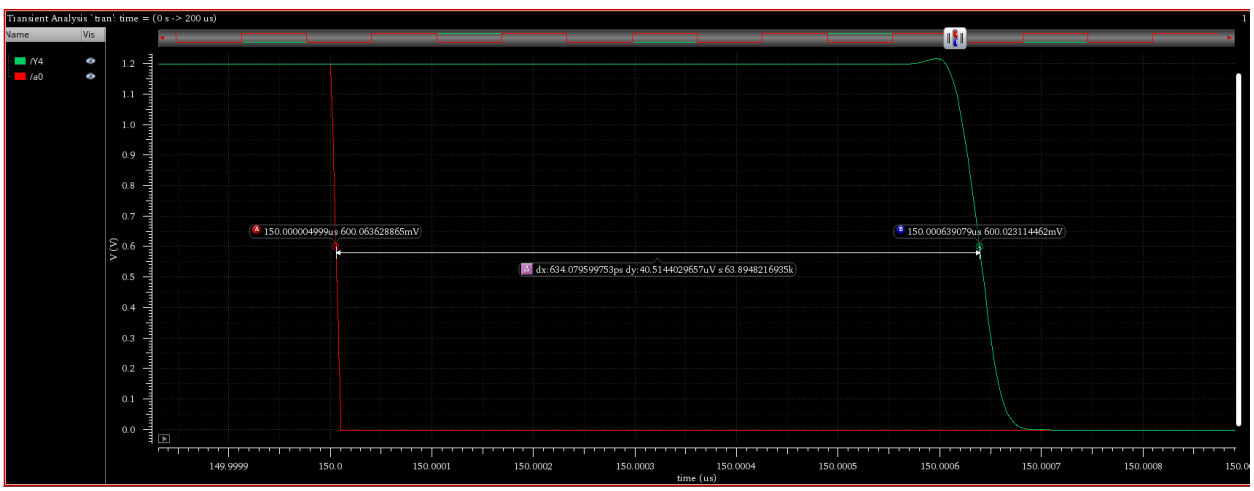


Figure: T_pHL

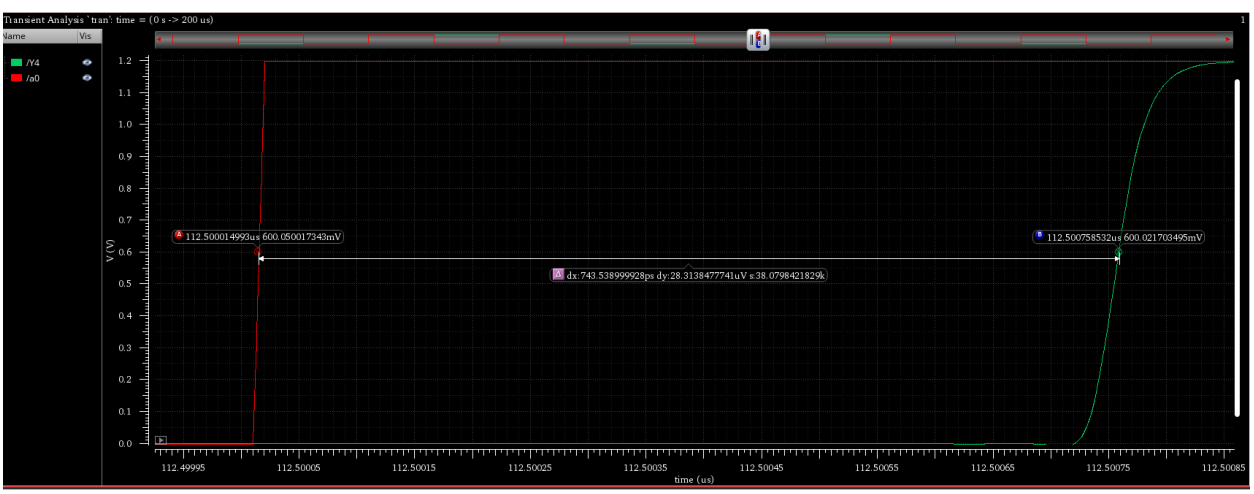
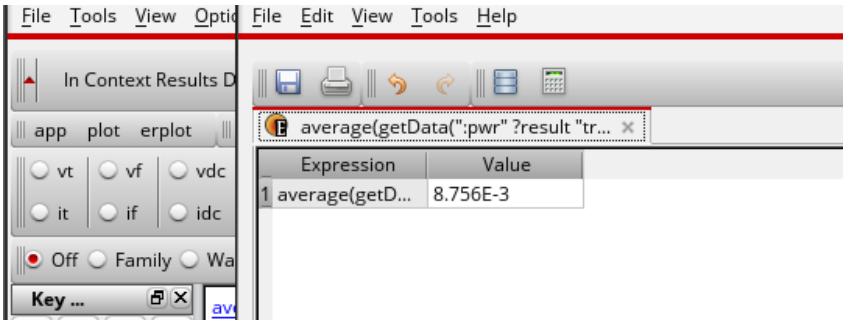
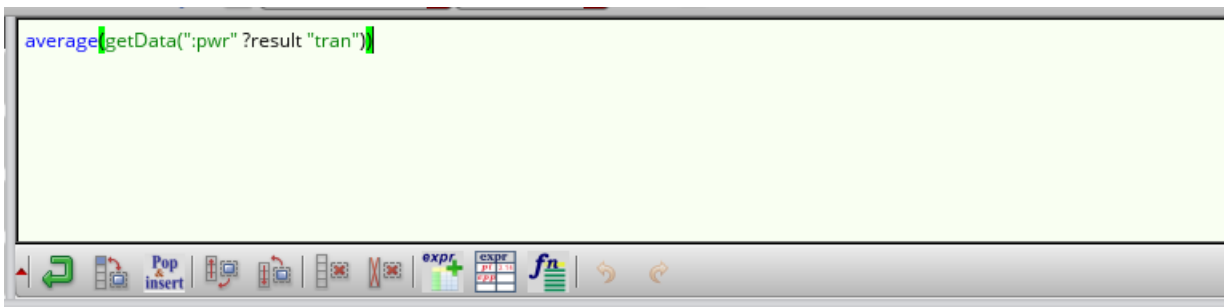


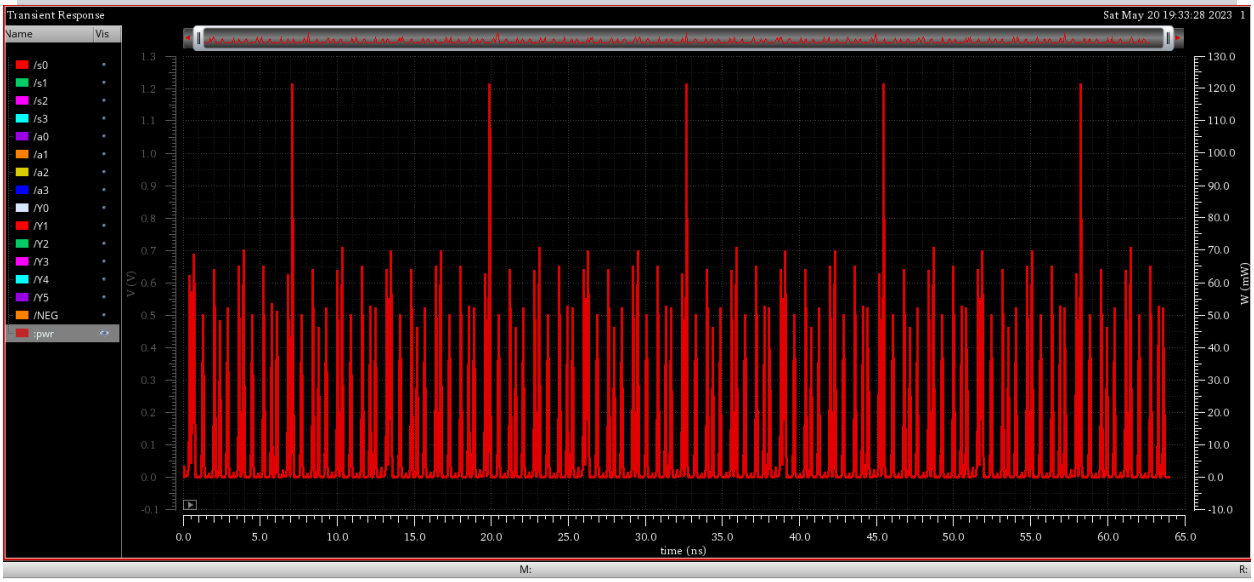
Figure: T_pLH

Power Consumption at Maximum Frequency:

Now in this part we are going to calculate the Average power consumption needed for our circuit at the maximum frequency of operation to get the maximum Frequency of operation we have previously determined that the needed period is about 2 ns so our frequency should be about 0.5 GHz. fmax = 0.5 GHz Now we set all our inputs and selectors to have a period of 2 ns then we use simulations to calculate our Power consumption, using calculator in Cadence, we calculated the average power in our ALU, So *P = 8.756 mw.*







Power Graph

Verilog code & snapshots

Unsigned logic Verilog

```
module main(
    input [3:0] a,
    input [3:0] b,
    input [3:0] s,
    output reg [5:0] y
);

reg [5:0] z1;
reg [3:0] z2;

always @ (a, b, s) begin
    case(s)
        4'b0000: z1 = $signed(a) + 1;
        4'b0001: z1 = $signed(a) - 1;
        4'b0010: z1 = $signed(a) *2;
        4'b0011: z1 = $signed(b) + 1;
        4'b0100: z1 = $signed(b) - 1;
        4'b0101: z1 = $signed(b) + $signed(b);
        4'b0110: z1 = $signed(a) + $signed(b);
        4'b0111: z1 = $signed(a)*4;
        4'b1000: z2 = ~a;
        4'b1001: z2 = ~b;
        4'b1010: z2 = a & b;
        4'b1011: z2 = a | b;
        4'b1100: z2 = a ^ b;
        4'b1101: z2 = ~(a ^ b);
        4'b1110: z2 = ~(a & b);
        4'b1111: z2 = ~(a | b);
        default:;
    endcase
end

always @(*) begin
if(s[3] == 0) begin
     y = z1;
end else begin
    y = z2;
end
end
endmodule
`timescale 1ns / 1ns
```

Unsigned logic testbench

```
module main_tb;

  // Inputs
  reg [3:0] a;
  reg [3:0] b;
  reg [3:0] s;

  // Outputs
  wire [5:0] y;

  // Instantiate the Unit Under Test (UUT)
  main uut (
    .a(a),
    .b(b),
    .s(s),
    .y
  );

  initial begin
    // Initialize inputs
    a = 4'b0000;
    b = 4'b0000;
    s = 4'b0000;

    // Wait 10 ns for global reset to finish
    #10;
 // Test case 1
  a = 4'b1000;
  b = 4'b0010;
  s = 4'b0000;
  #10
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 2
  a = 4'b1100;
  b = 4'b0010;
  s = 4'b0001;
  #10
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
   // Test case 3
  a = 4'b1100;
  b = 4'b0010;
  s = 4'b0010;
  #10
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
   // Test case 4
  a = 4'b1100;
  b = 4'b0010;
  s = 4'b0011;

  #10
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
```

```
 // Test case 5
a = 4'b1100;
b = 4'b1111;
s = 4'b0100;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 6
a = 4'b1100;
b = 4'b0010;
s = 4'b0101;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 7
a = 4'b1100;
b = 4'b0010;
s = 4'b0110;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 8
a = 4'b1100;
b = 4'b0010;
s = 4'b0111;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 9
a = 4'b1100;
b = 4'b0010;
s = 4'b1000;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 10
a = 4'b1100;
b = 4'b0110;
s = 4'b1001;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
   // Test case 11
a = 4'b1100;
b = 4'b0010;
s = 4'b1010;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 12
a = 4'b1100;
b = 4'b0010;
s = 4'b1011;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 13
a = 4'b1100;
b = 4'b0010;
s = 4'b1100;
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
```

```
  #10
   // Test case 14
  a = 4'b1100;
  b = 4'b0010;
  s = 4'b1101;
  #10
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
   // Test case 15
  a = 4'b1100;
  b = 4'b0010;
  s = 4'b1110;
  #10
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
   // Test case 16
  a = 4'b1101;
  b = 4'b0010;
  s = 4'b1111;
  #10
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);

  end

endmodule
```

Signed I/O Code "Bonus"

```
module main(
    input [3:0] a,
    input [3:0] b,
    input [3:0] s,
    output reg [5:0] y
);

reg [5:0] z1;
reg [3:0] z2;

always @ (a, b, s) begin
    case(s)
        4'b0000: z1 = (a) + 1;
        4'b0001: z1 = (a) - 1;
        4'b0010: z1 = (a) *2;
        4'b0011: z1 = (b) + 1;
        4'b0100: z1 = (b) - 1;
        4'b0101: z1 = (b) + (b);
        4'b0110: z1 = (a) + (b);
        4'b0111: z1 = (a)*4;
        4'b1000: z2 = ~a;
        4'b1001: z2 = ~b;
        4'b1010: z2 = a & b;
        4'b1011: z2 = a | b;
        4'b1100: z2 = a ^ b;
        4'b1101: z2 = ~(a ^ b);
        4'b1110: z2 = ~(a & b);
        4'b1111: z2 = ~(a | b);
        default:;
    endcase
end

always @(*) begin
if(s[3] == 0) begin
     y = z1;
end else begin
    y = z2;
end
end
endmodule
`timescale 1ns / 1ns
```

Signed I/O testbench “Bonus”

```
module main_tb;

  // Inputs
  reg [3:0] a;
  reg [3:0] b;
  reg [3:0] s;

  // Outputs
  wire [5:0] y;

  // Instantiate the Unit Under Test (UUT)
  main uut (
    .a(a),
    .b(b),
    .s(s),
    .y
  );

  initial begin
    // Initialize inputs
    a = 4'b0000;
    b = 4'b0000;
    s = 4'b0000;

    // Wait 10 ns for global reset to finish
    #10;
 // Test case 1
  a = 4'b1000;
  b = 4'b0010;
  s = 4'b0000;
  #10
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 2
  a = 4'b1100;
  b = 4'b0010;
  s = 4'b0001;
  #10
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
   // Test case 3
  a = 4'b1100;
  b = 4'b0010;
  s = 4'b0010;
  #10
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
   // Test case 4
  a = 4'b1100;
  b = 4'b0010;
  s = 4'b0011;
```

```
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 5
a = 4'b1100;
b = 4'b1111;
s = 4'b0100;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 6
a = 4'b1100;
b = 4'b0010;
s = 4'b0101;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 7
a = 4'b1100;
b = 4'b0010;
s = 4'b0110;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 8
a = 4'b1100;
b = 4'b0010;
s = 4'b0111;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 9
a = 4'b1100;
b = 4'b0010;
s = 4'b1000;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 10
a = 4'b1100;
b = 4'b0110;
s = 4'b1001;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
   // Test case 11
a = 4'b1100;
b = 4'b0010;
s = 4'b1010;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 12
a = 4'b1100;
b = 4'b0010;
s = 4'b1011;
#10
$display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
 // Test case 13
a = 4'b1100;
b = 4'b0010;
```

```
  s = 4'b1100;
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);

  #10
   // Test case 14
  a = 4'b1100;
  b = 4'b0010;
  s = 4'b1101;
  #10
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
   // Test case 15
  a = 4'b1100;
  b = 4'b0010;
  s = 4'b1110;
  #10
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);
   // Test case 16
  a = 4'b1101;
  b = 4'b0010;
  s = 4'b1111;
  #10
  $display("a = %b, b = %b, s = %b, y = %b", a, b, s, y);

  end

endmodule
```

Snapshots

Terminal

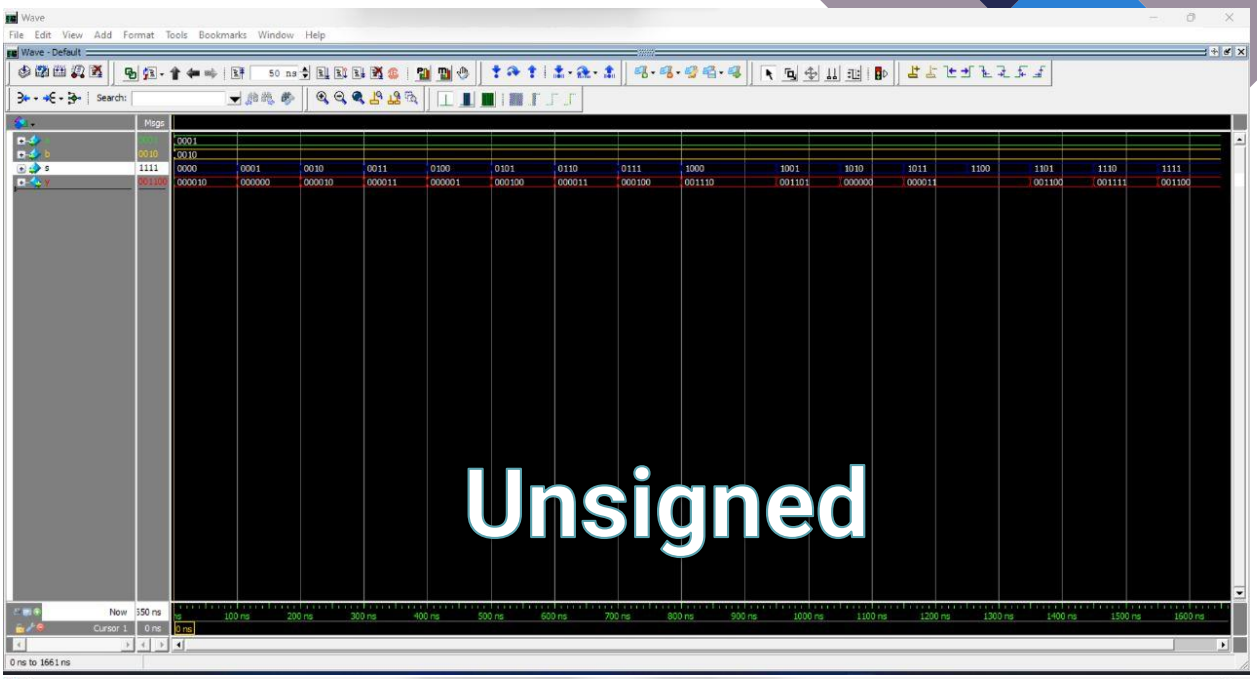
```
a = 1000, b = 0010, s = 0000, y = 001001
a = 1100, b = 0010, s = 0001, y = 001011
a = 1100, b = 0010, s = 0010, y = 011000
a = 1100, b = 0010, s = 0011, y = 000011
a = 1100, b = 1111, s = 0100, y = 001110
a = 1100, b = 0010, s = 0101, y = 000100
a = 1100, b = 0010, s = 0110, y = 001110
a = 1100, b = 0010, s = 0111, y = 110000
a = 1100, b = 0010, s = 1000, y = 000011
a = 1100, b = 0110, s = 1001, y = 001001
a = 1100, b = 0010, s = 1010, y = 000000
a = 1100, b = 0010, s = 1011, y = 001110
a = 1100, b = 0010, s = 1100, y = 001110
a = 1100, b = 0010, s = 1101, y = 000001
a = 1100, b = 0010, s = 1110, y = 001111
a = 1101, b = 0010, s = 1111, y = 000000
```

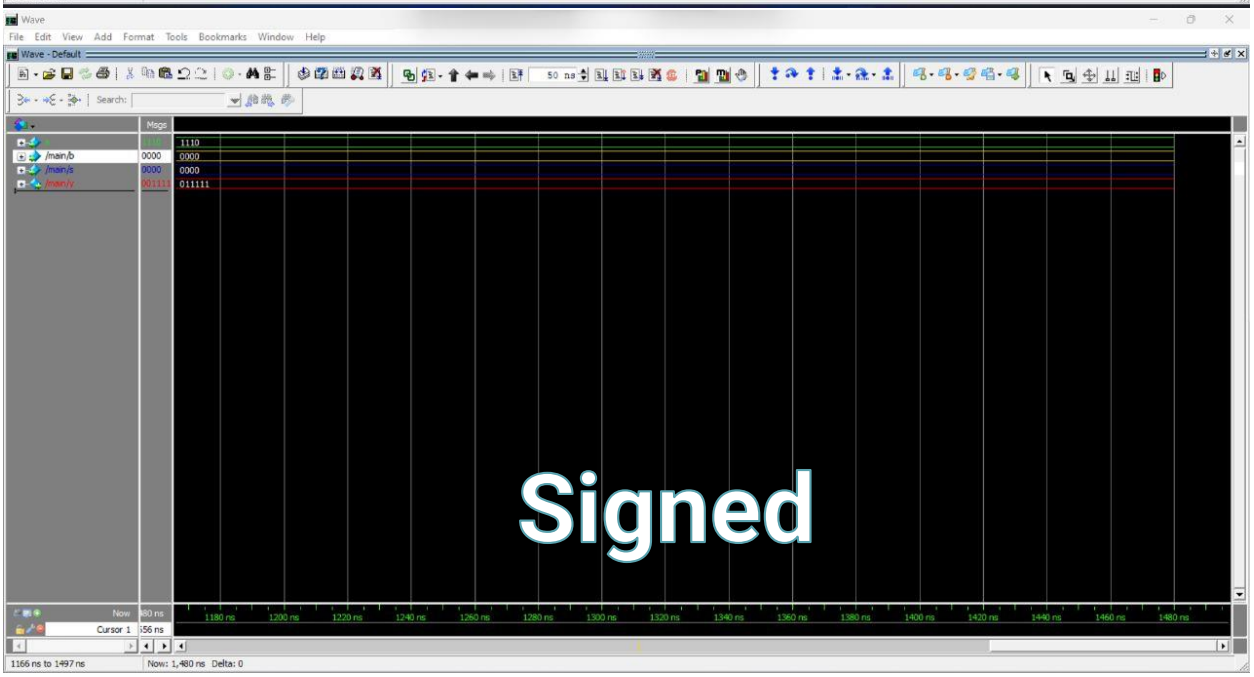
Unsigned Test

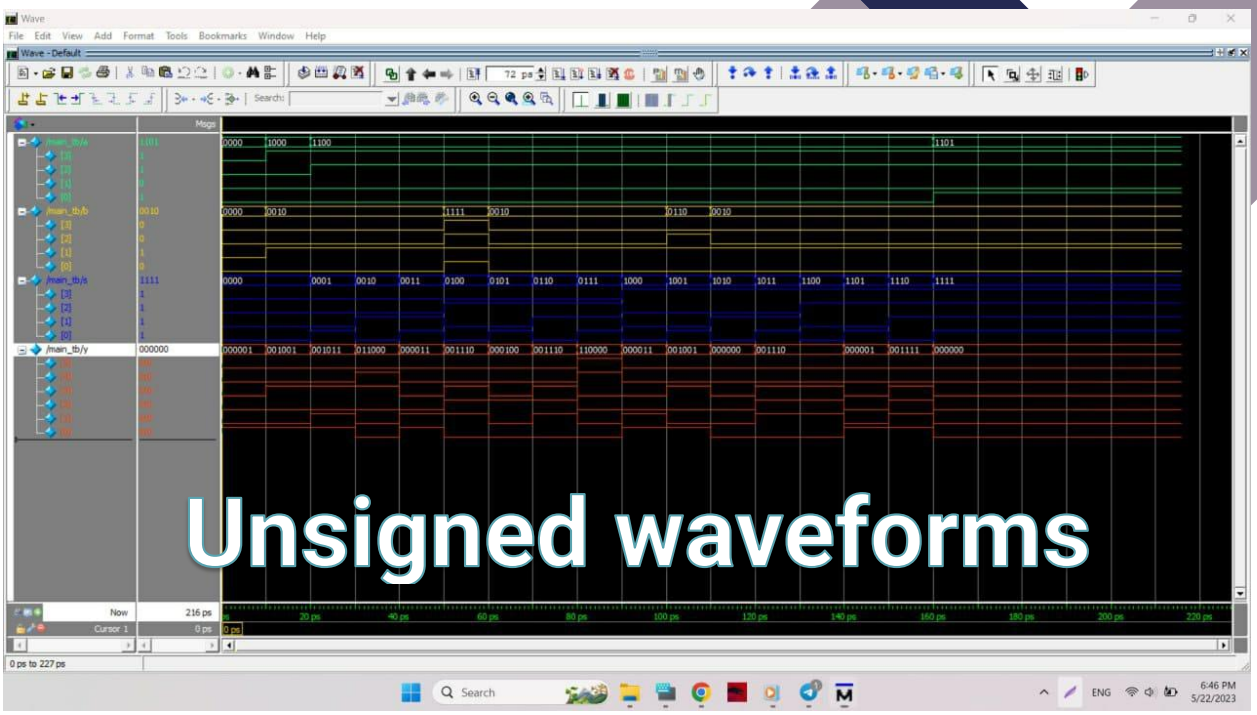
Terminal

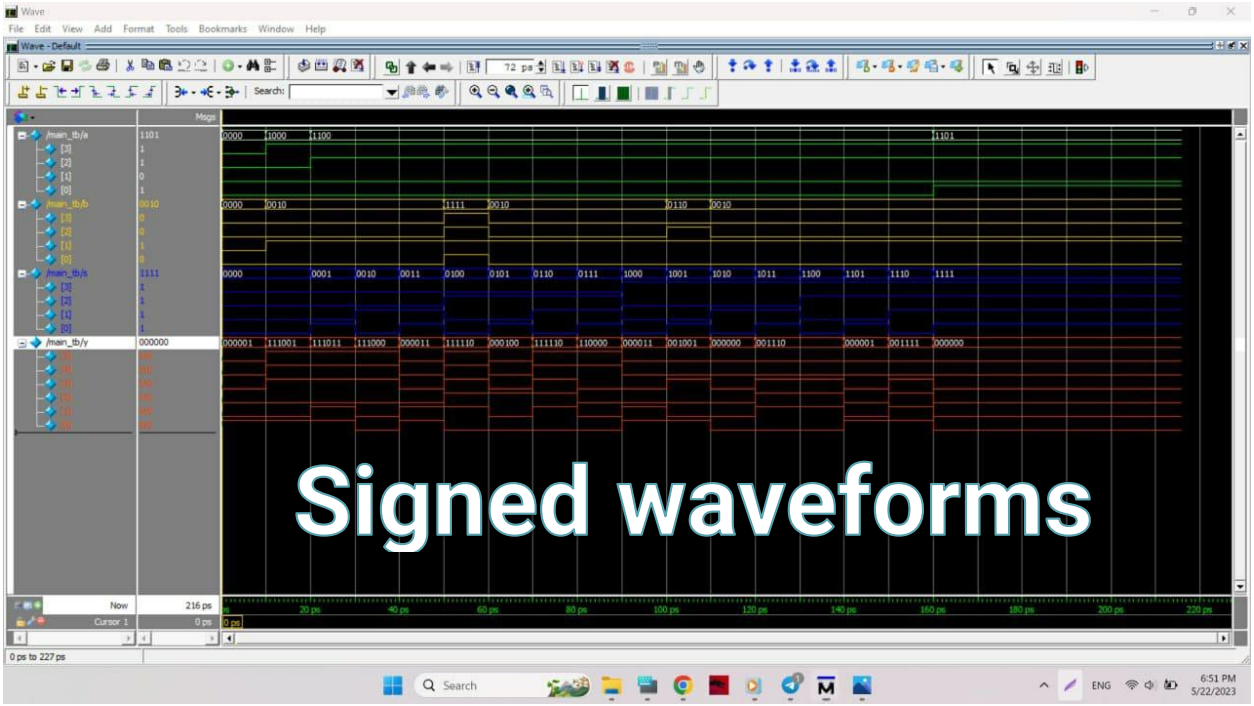
```
a = 1000, b = 0010, s = 0000, y = 111001
a = 1100, b = 0010, s = 0001, y = 111011
a = 1100, b = 0010, s = 0010, y = 111000
a = 1100, b = 0010, s = 0011, y = 000011
a = 1100, b = 1111, s = 0100, y = 111110
a = 1100, b = 0010, s = 0101, y = 000100
a = 1100, b = 0010, s = 0110, y = 111110
a = 1100, b = 0010, s = 0111, y = 110000
a = 1100, b = 0010, s = 1000, y = 000011
a = 1100, b = 0110, s = 1001, y = 001001
a = 1100, b = 0010, s = 1010, y = 000000
a = 1100, b = 0010, s = 1011, y = 001110
a = 1100, b = 0010, s = 1100, y = 001110
a = 1100, b = 0010, s = 1101, y = 000001
a = 1100, b = 0010, s = 1110, y = 001111
a = 1101, b = 0010, s = 1111, y = 000000
```

Signed Test


Unsigned


Signed





The End