ALU Design Report

Written 03/20/2024

Mico Santiago

Trinh Cong Luan

Project Introduction

This project entails designing and implementing a 4-bit Arithmetic Logic Unit (ALU) in 0.18µm CMOS technology. The ALU will perform various arithmetic and logical operations based on a 3-bit opcode input. These operations include addition, subtraction, bitwise AND, OR, XOR, as well as passing input A, input B, or setting all outputs to high.

The design follows a hierarchical structure, from transistor level to block level, and utilize static CMOS logic, with a 2:1 ratio of PMOS width to NMOS width for equivalent drive strength.

Additionally, the project requires comprehensive simulation-based testing to demonstrate functionality. Design decisions, including architecture, device sizing, and implementation choices, must be justified in the project report. Additionally, the core full adder layout must be provided and meet design rule check (DRC) and layout versus schematic (LVS) requirements.

Overall, the assignment aims to consolidate knowledge of CMOS logic design, combinational and sequential logic, and microprocessor building blocks, culminating in the creation of a fully functional 4-bit ALU.

ADDITIONAL PROJECT REQUIREMENTS FOR Trinh Cong Luan ATTATCHED AT THE BOTTOM

Design Introduction

We will start from the block level and work our way to down to the transistor level, finishing off with the testbench results.

Our ALU design takes in three inputs. A 3-bit instruction and two 4-bit user inputs. The instructions are listed below.

INSTR<2:0>	OUT<3:0>	Description
000	= A + B	Add
001	= A - B	Subtract (using two's complement)
010	= (A AND B)	Bitwise AND
011	= (A OR B)	Bitwise OR
100	= (A XOR B)	Bitwise XOR
101	A	Pass A
110	B	Pass B
111	= 1111	All high

The 3-bit instruction is fed into the Read Only Memory (ROM) while the 4-bit inputs are processed in parallel. Finally, the output is selected by the MUX.

Block Level Design is shown below

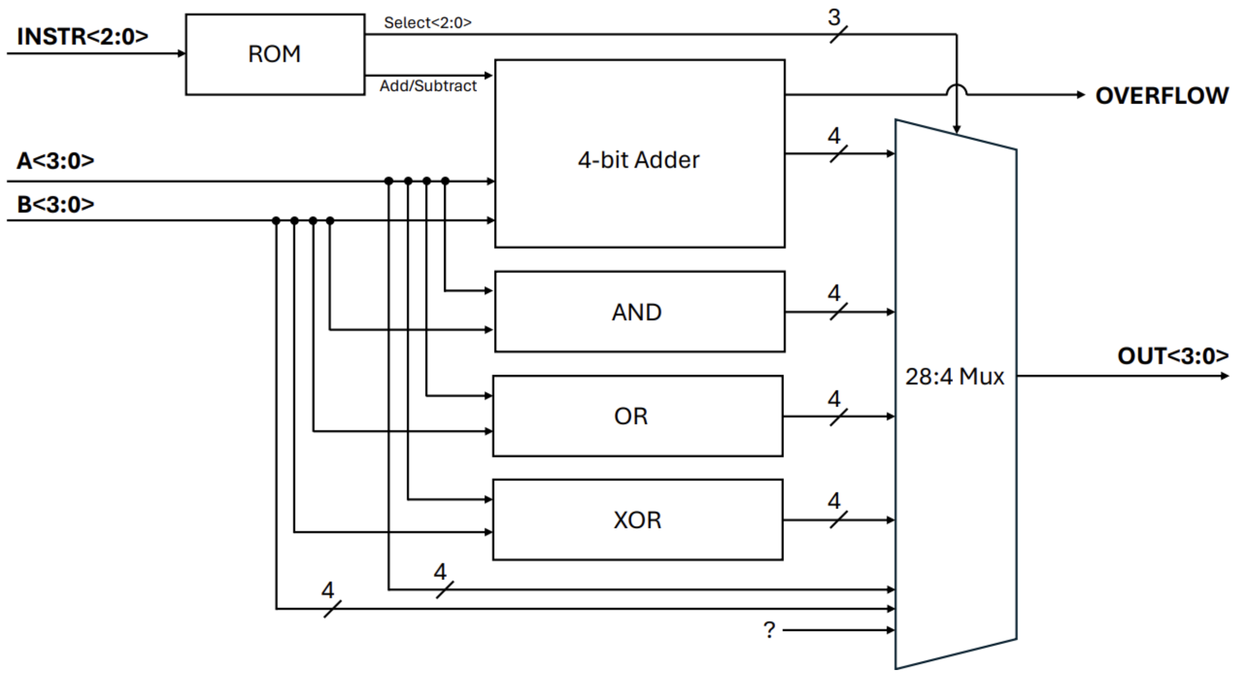


Fig. 1 ALU Block Schematic

The ALU implemented in Cadence below. More detail for each block is explained furthur in this document.

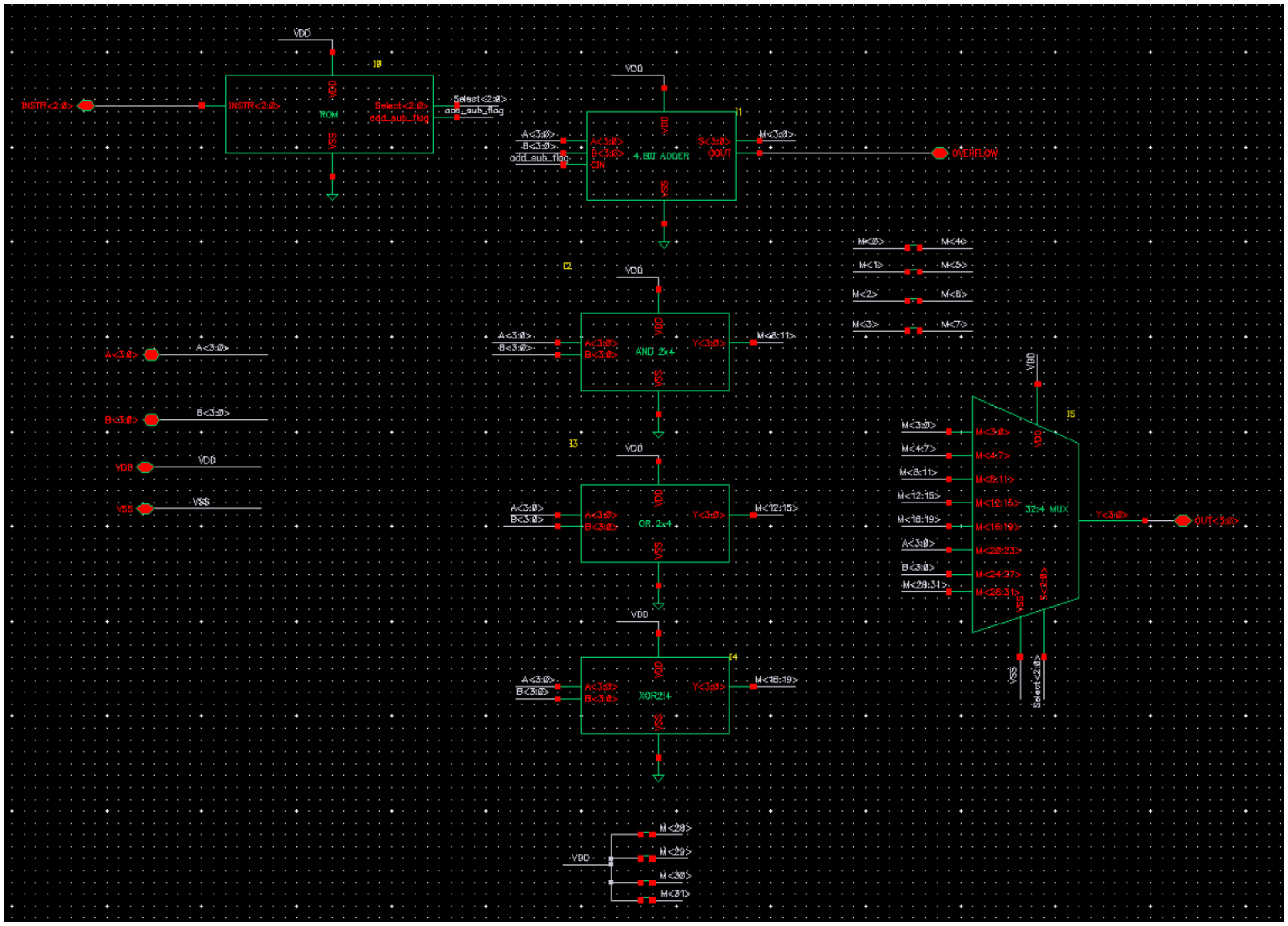


Fig. 2 ALU Schematic in Cadence

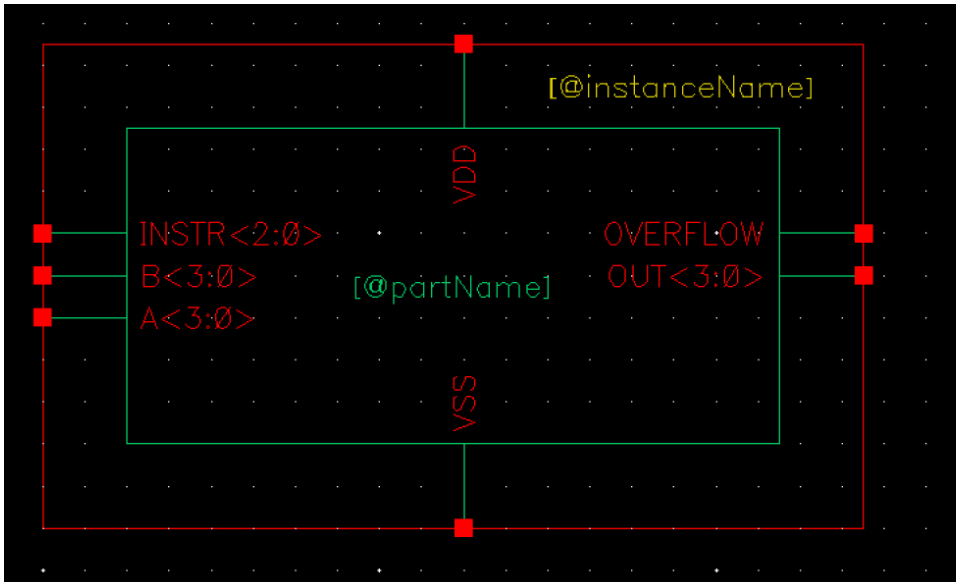


Fig 3. ALU Block Symbol

ALU Testbench

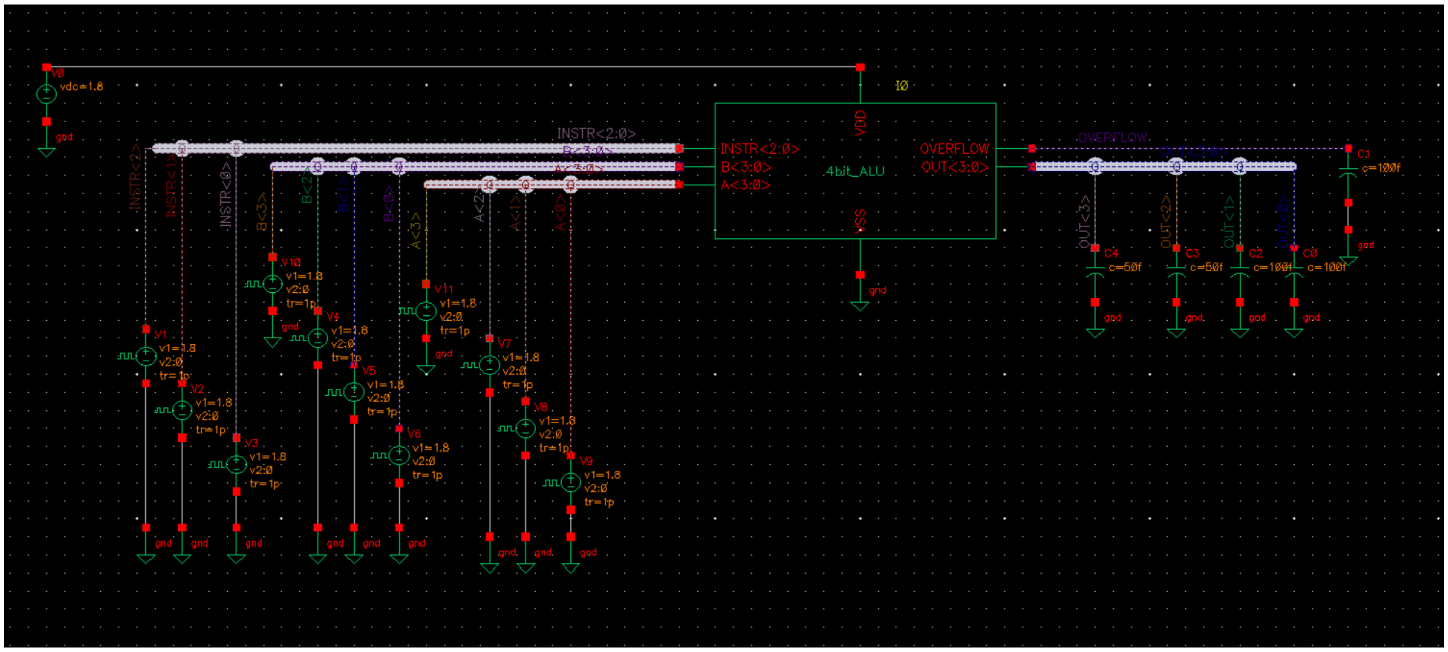
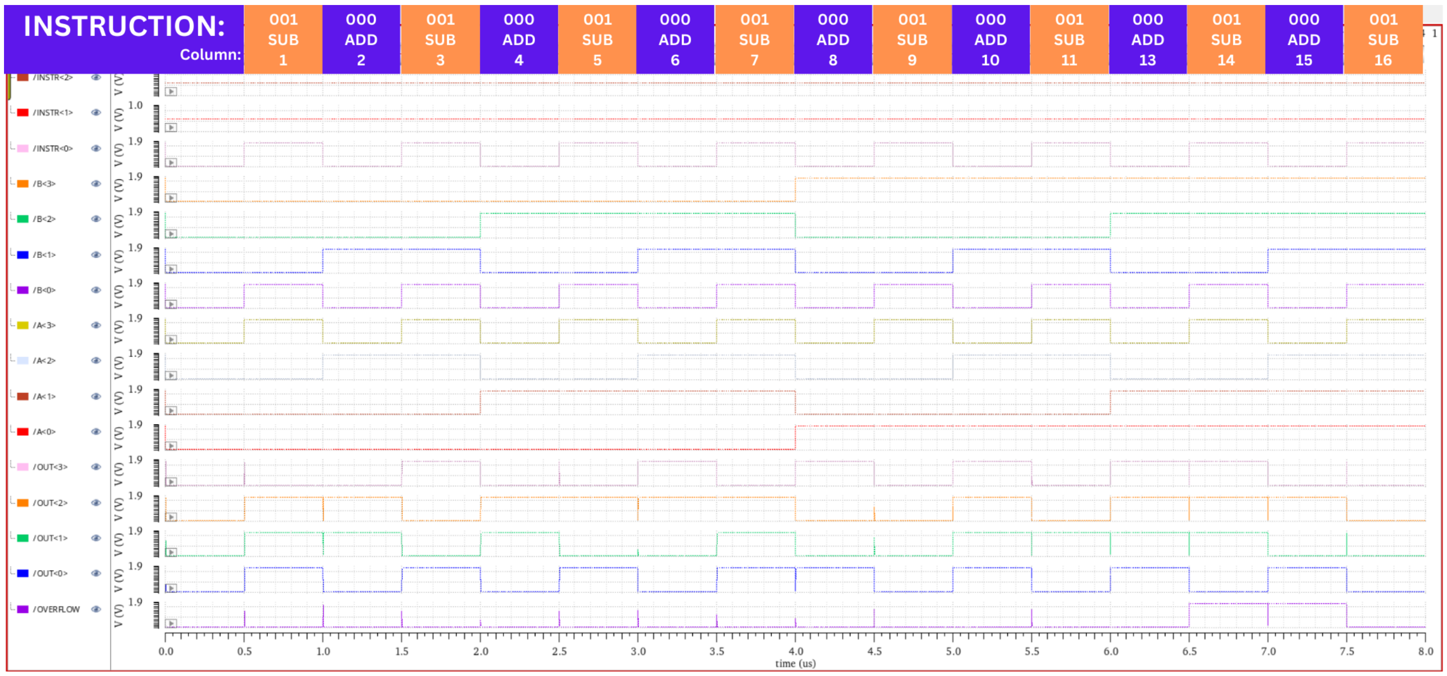


Fig 4. ALU Testbench

ALU Testbench – ADD/SUB Functionality



INSTRUCTION: Column:	001 SUB 1	000 ADD 2	001 SUB 3	000 ADD 4	001 SUB 5	000 ADD 6	001 SUB 7	000 ADD 8	001 SUB 9	000 ADD 10	001 SUB 11	000 ADD 13	001 SUB 14	000 ADD 15	001 SUB 16

Fig. 5 Testbench results for testing the addition and subtraction instructions

Results

Notice that **OUT** will be 5 bits in two's complement when the **SUBTRACT** instruction is called, where the **OVERFLOW** bit will be attached to the leftmost side. Column 12 was mislabeled.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
A<3:0>	4'b1000	4'b0010	4'b1100	4'b0010	4'b1010	4'b0110	4'b1110	4'b0010
B<3:0>	4'b0001	4'b0100	4'b0011	4'b0100	4'b0101	4'b0110	4'b0111	4'b0100
OUT	5'b00111	4'b0110	5'b01001	4'b0110	5'b00101	4'b1100	5'b00111	4'b0110
OVRFLW	0	0	0	0	0	0	0	0
A (+/-) B	8-1=7	2+4=6	12-3=9	2+4=6	10-5=5	6-6=0	14-7=7	2+4=6
	9	**10**	**11**	**13**	**14**	**15**	**16**	
A<3:0>	4'b1001	4'b0101	4'b1101	4'b0011	4'b1011	4'b0111	4'b1111	
B<3:0>	4'b1001	4'b1010	4'b1011	4'b1100	4'b1101	4'b1110	4'b1111	
OUT	5'b00000	4'b1111	5'b00010	4'b1111	5'b11110	5'b10101	5'b00000	
OVRFLW	0	0	0	0	1	1	0	
A (+/-) B	7-7=0	5+10=15	13-11=2	3+12=15	11-13= -2	7+14=21	15-15=0	

Fig 6. Results from Fig.5 test setup

Notice that these results are correct. In columns 14 and 15, we witness overflow and proof that our circuit uses two's complement to represent negative numbers with 5 bits.

ALU Testbench – All Other Functionality

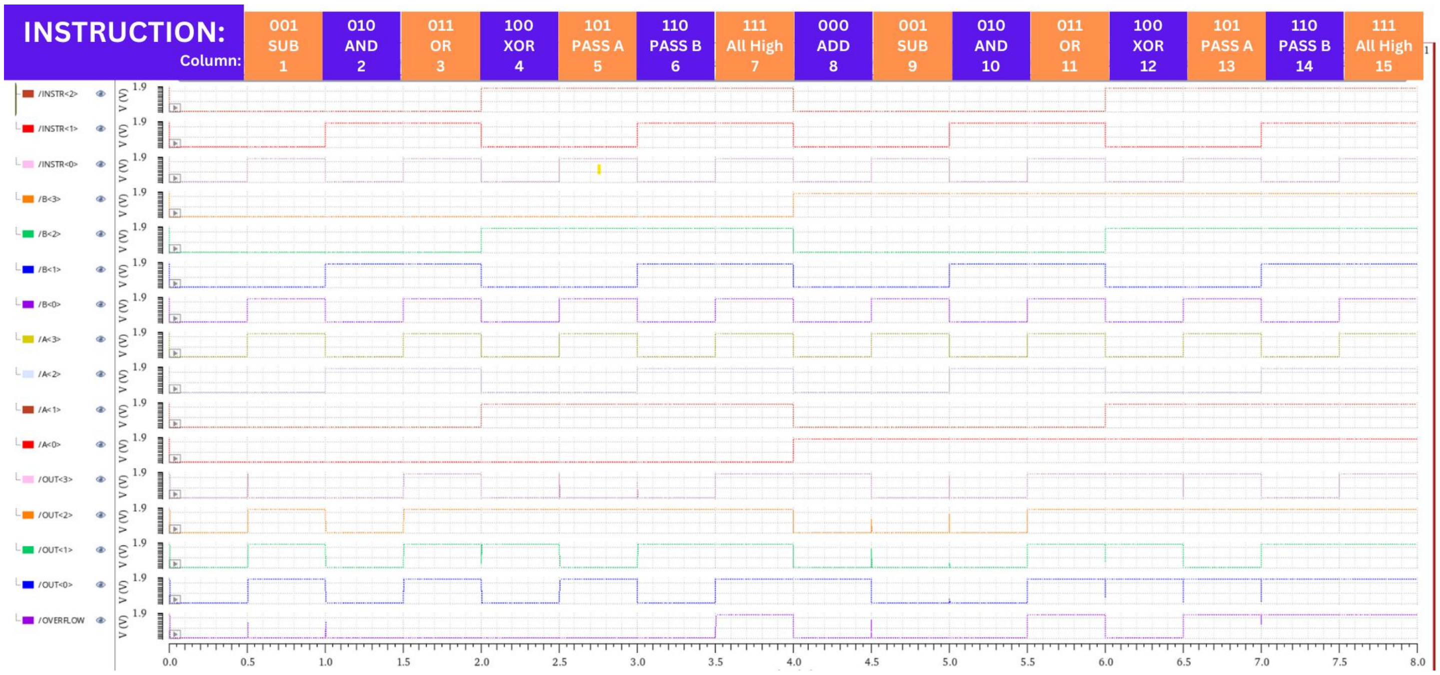
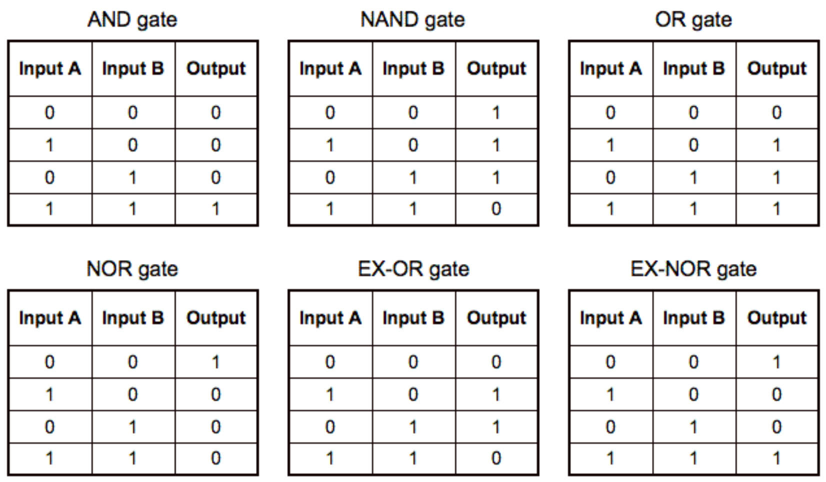


Fig 7. Testing ALL functions

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
A<3:0>	4'b1000	4'b1000	4'b1100	4'b0010	4'b1010	4'b0110	4'b1110	4'b0001
B<3:0>	4'b0001	4'b0010	4'b0011	4'b0100	4'b0101	4'b0110	4'b0111	4'b1000
OUT	5'b00111	4'b0000	4'b1111	4'b0110	4'b0101	4'b0110	5'b1111	4'b1001
OVRFLW	0	0	0	0	0	0	1	0

A (+/-) B	8-1=7	AND	OR	XOR	PASS A	PASS B	ALL HIGH	1+8=9
	9	**10**	**11**	**12**	**13**	**14**	**15**	
A<3:0>	4’b1001	4’b0101	4’b1101	4’b0011	4’b1011	4’b0111	4’b1111	
B<3:0>	4’b1001	4’b1010	4’b1011	4’b1100	4’b1101	4’b1110	4’b1111	
OUT	5’b00000	4’b1111	5’b11111	4’b1111	4’b1011	4’b0111	5’b11111	
OVRFLW	0	0	1	0	1	1	1	
A (+/-) B	9-9=0	5+10=15	OR	XOR	PASS A	PASS B	ALL HIGH	

Fig 8. Table of results from testing ALL functions setup



AND gate

Input A	Input B	Output
0	0	0
1	0	0
0	1	0
1	1	1

NAND gate

Input A	Input B	Output
0	0	1
1	0	1
0	1	1
1	1	0

OR gate

Input A	Input B	Output
0	0	0
1	0	1
0	1	1
1	1	1

NOR gate

Input A	Input B	Output
0	0	1
1	0	0
0	1	0
1	1	0

EX-OR gate

Input A	Input B	Output
0	0	0
1	0	1
0	1	1
1	1	0

EX-NOR gate

Input A	Input B	Output
0	0	1
1	0	0
0	1	0
1	1	1

Fig 9. Gate table reference

Results Conclusion

These results show conclusive evidence that the ALU works as intended from the logical functions of AND, OR, XOR to the Adder and Subtractor to the Pass A, Pass B and All High instructions.

The 4-bit ALU works!

Read Only Memory (ROM)

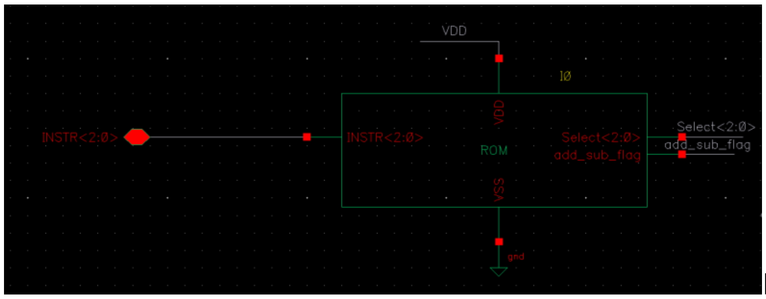


Fig 10. ROM symbol on the ALU schematic

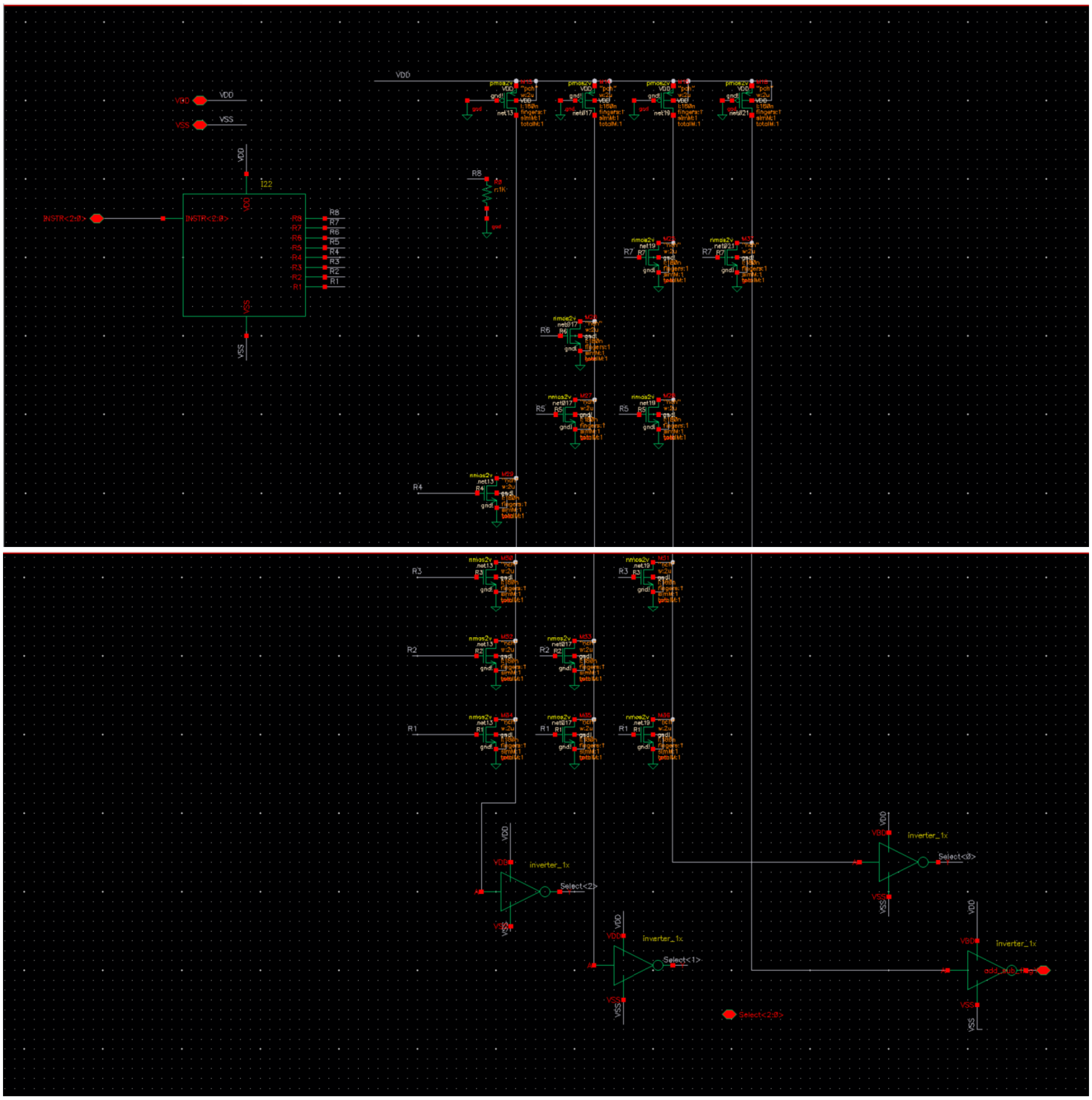


Fig 11. ROM Schematic

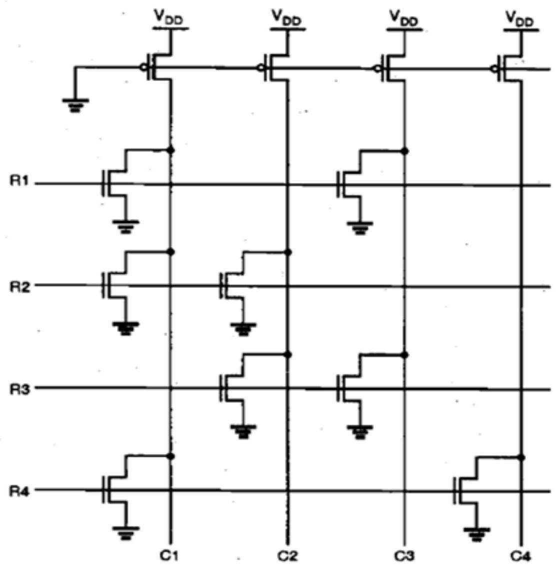
ROM Explained

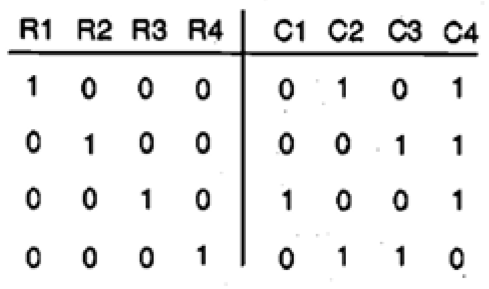
The decoder will receive a 3-bit instruction, determining which "line" is selected. However, it's important to note that the naming convention for the lines has been swapped, where R8 represents line 1 and R1 represents line 8. Each line corresponds to a specific instruction number. For instance, if the instruction 001 is selected, the decoder will output R7, indicating Select<2:0> = 001, and raise a subtraction flag.

In this 8-bit by 4-bit memory array, each column has a nMOS NOR gate controlled by row signals (word lines). Only one row, or word line, is active (selected) at a time by raising its voltage to VDD, while all other rows are kept at a low voltage level.

When an active transistor exists at the intersection of a column and the selected row, it pulls down the column voltage to a logic low level. If there's no active transistor at the intersection, the column voltage is pulled high by the pMOS load device.

Therefore, a logic "1" bit is stored when there's no active transistor (absence), and a logic "0" bit is stored when there's an active transistor at the intersection.

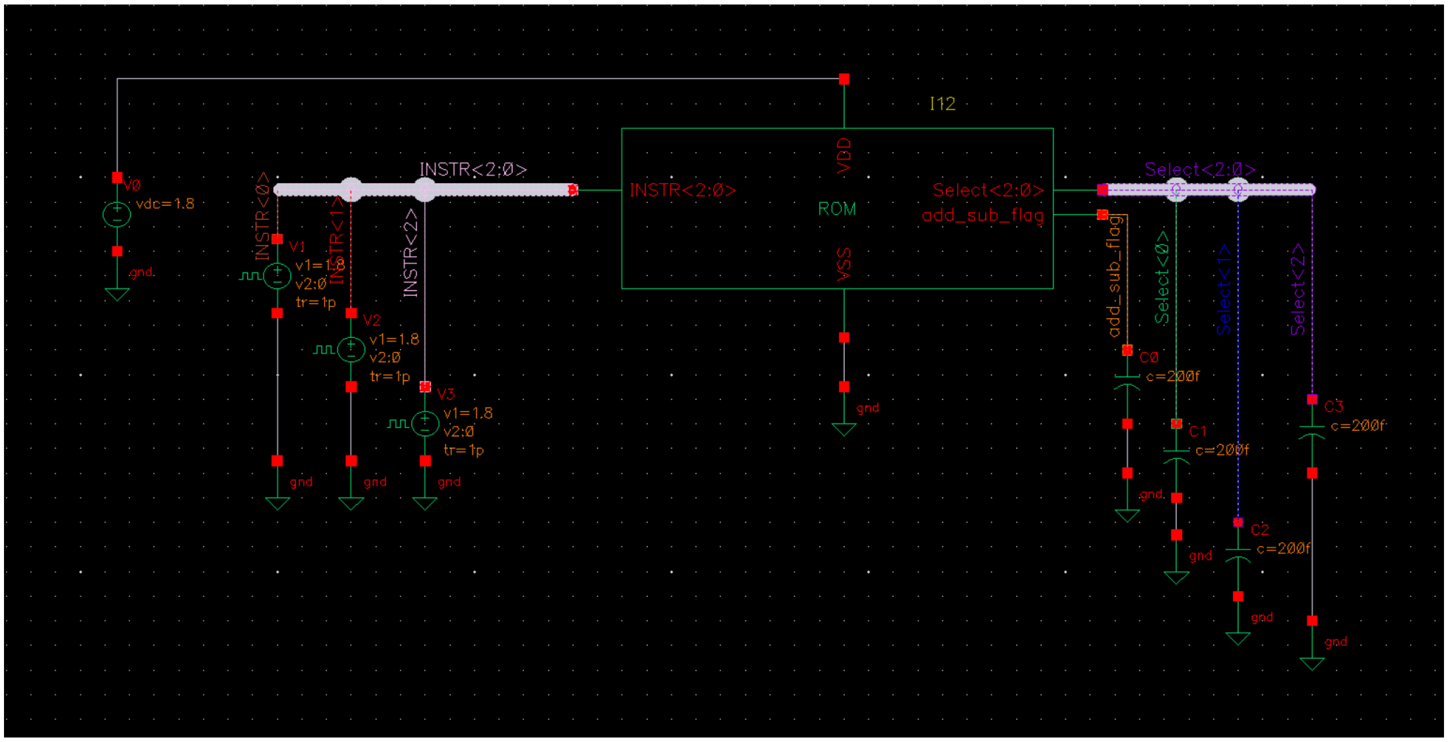




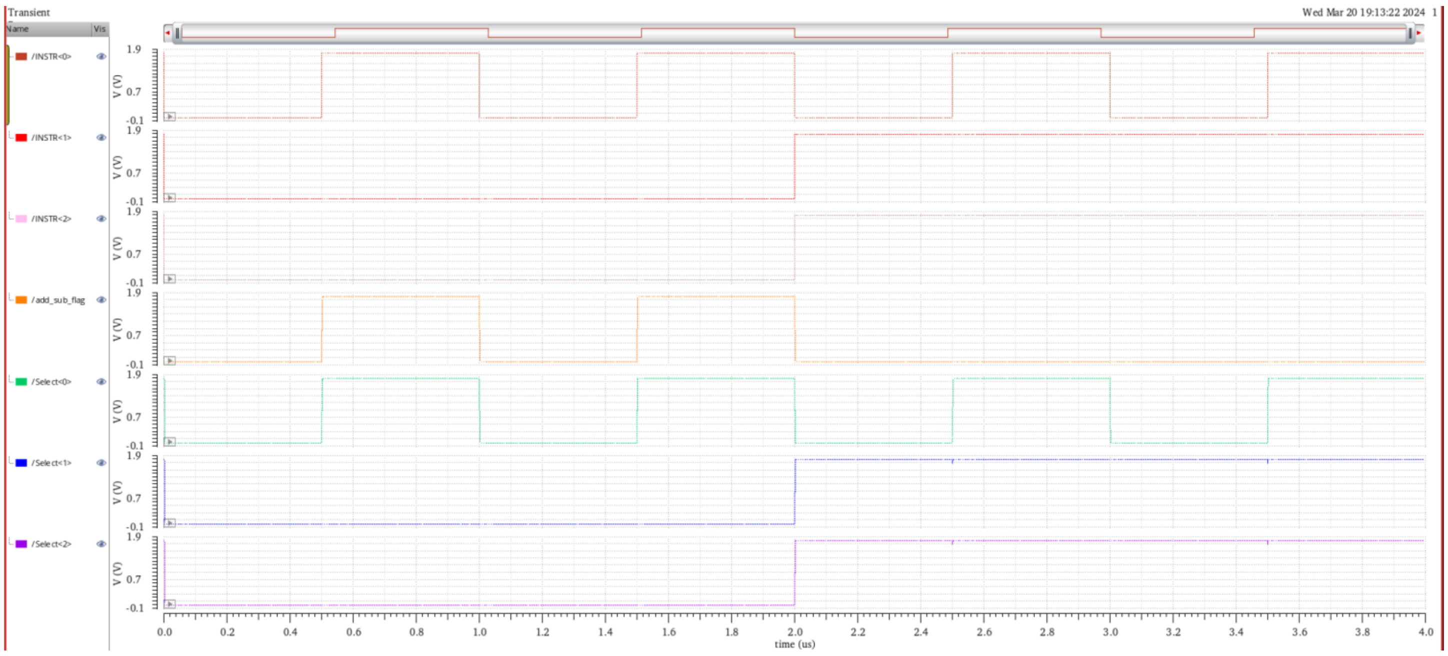
R1	R2	R3	R4	C1	C2	C3	C4
1	0	0	0	0	1	0	1
0	1	0	0	0	0	1	1
0	0	1	0	1	0	0	1
0	0	0	1	0	1	1	0

Fig 12. 4x4 NOR based ROM array and it’s truth table

ROM Testbench



ROM Testbench Results



At close inspection, we can see that INSTR<2:0> which equals 001 @ 0.5us and @ 1.5us will raise the add_sub_flag which will trigger the subtract functionality inside the 4-bit adder. The Select<2:0> bits mirror the INSTR<2:0>, therefore, the functionality is working.

3 to 8 Decoder

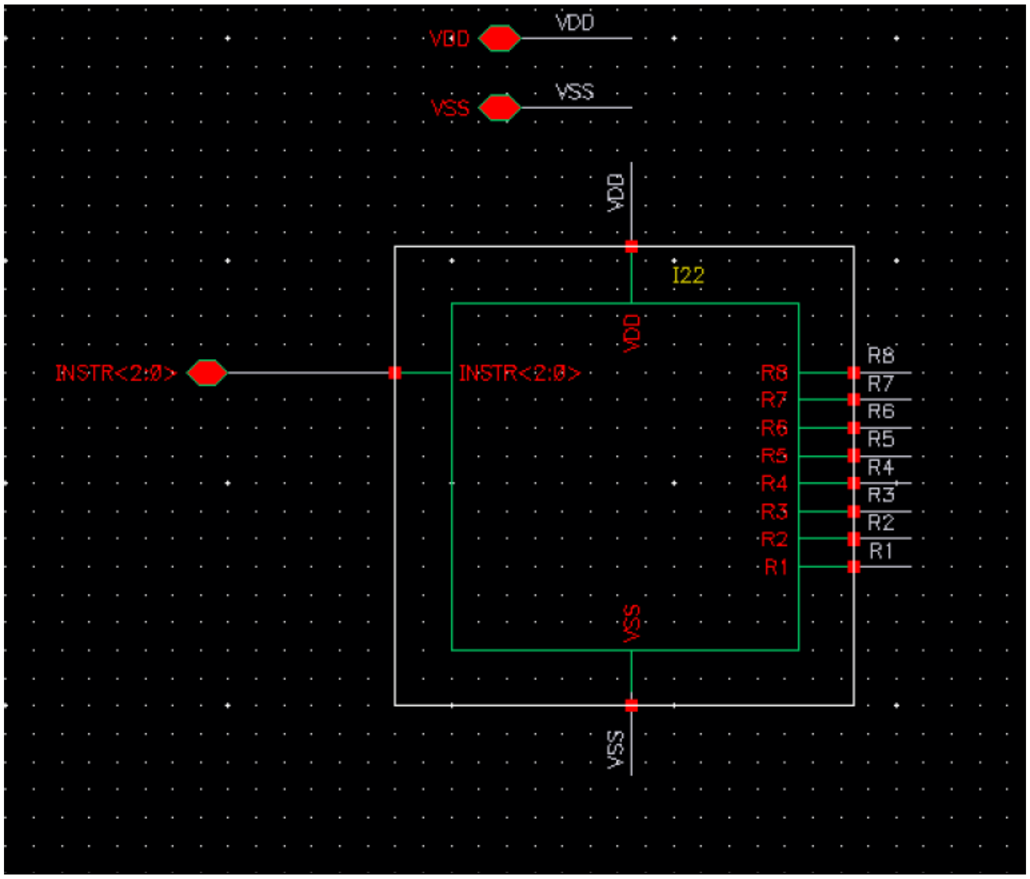


Fig 6. Decoder Block Symbol

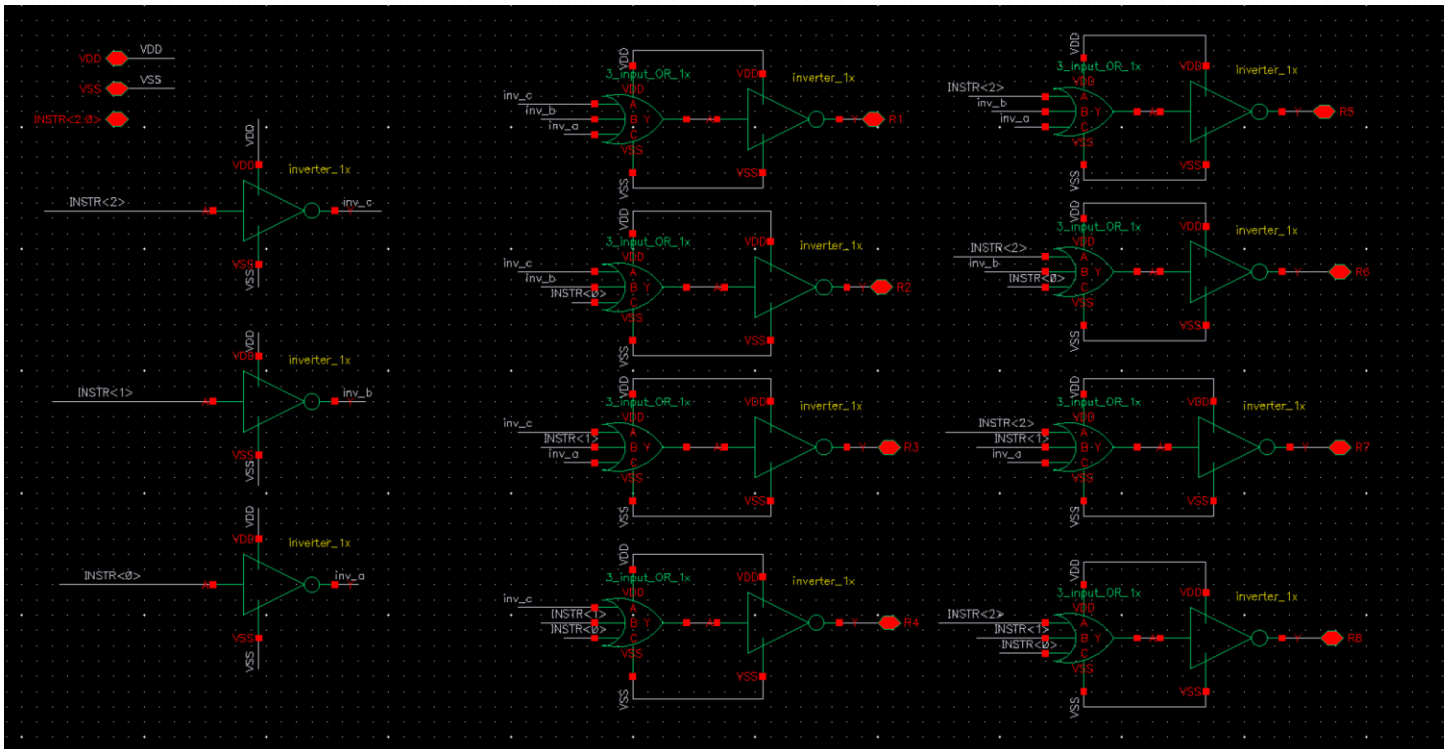


Fig 7. Decoder Schematic

3 to 8 Decoder Explained

This decoder circuit, known as a 3 to 8 decoder, has 8 logic outputs based on 3 inputs and an enable pin. It's built using inverters and NOR logic gates. With 3 binary inputs (S0, S1, S2), it activates one of the eight outputs (D0 through D7). Essentially, it translates a 3-bit binary input into one of eight possible outputs.

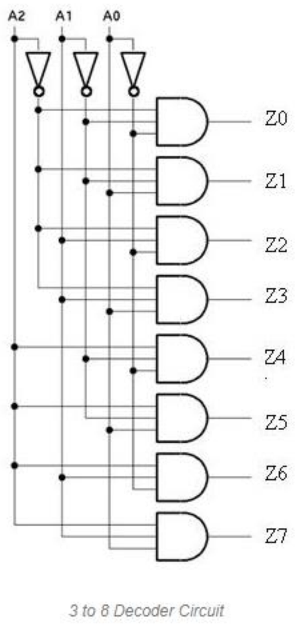


Fig 8. 3 to 8 Decoder circuit

Note that in this diagram Z0 is R8 in this paper's ALU.

Decoder Testbench

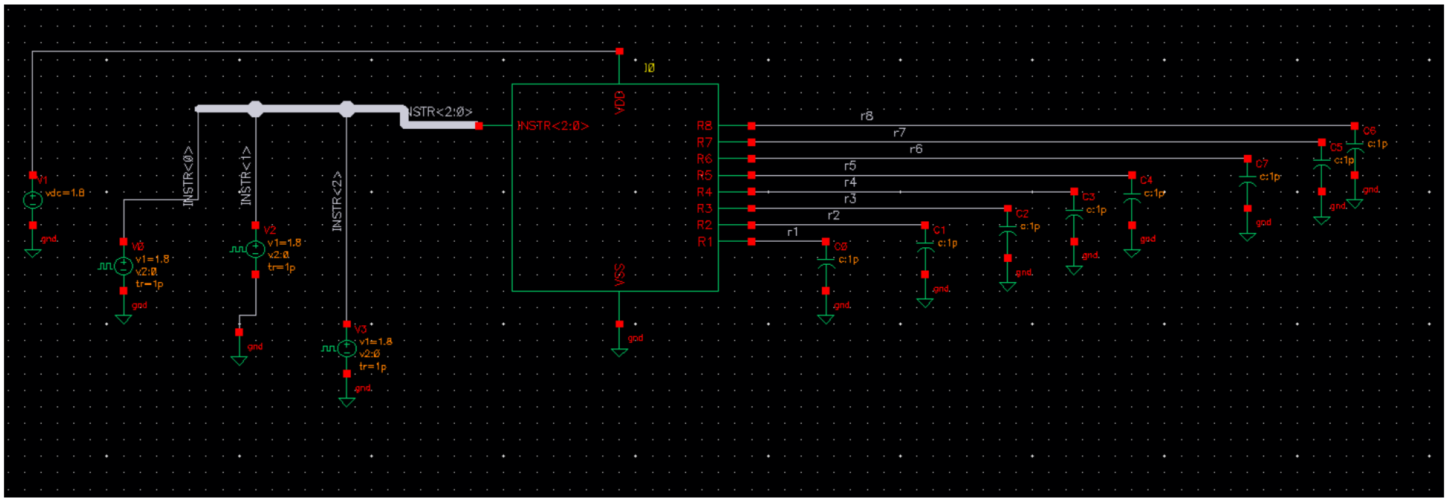


Fig 9. Decoder Testbench

Decoder Testbench Results

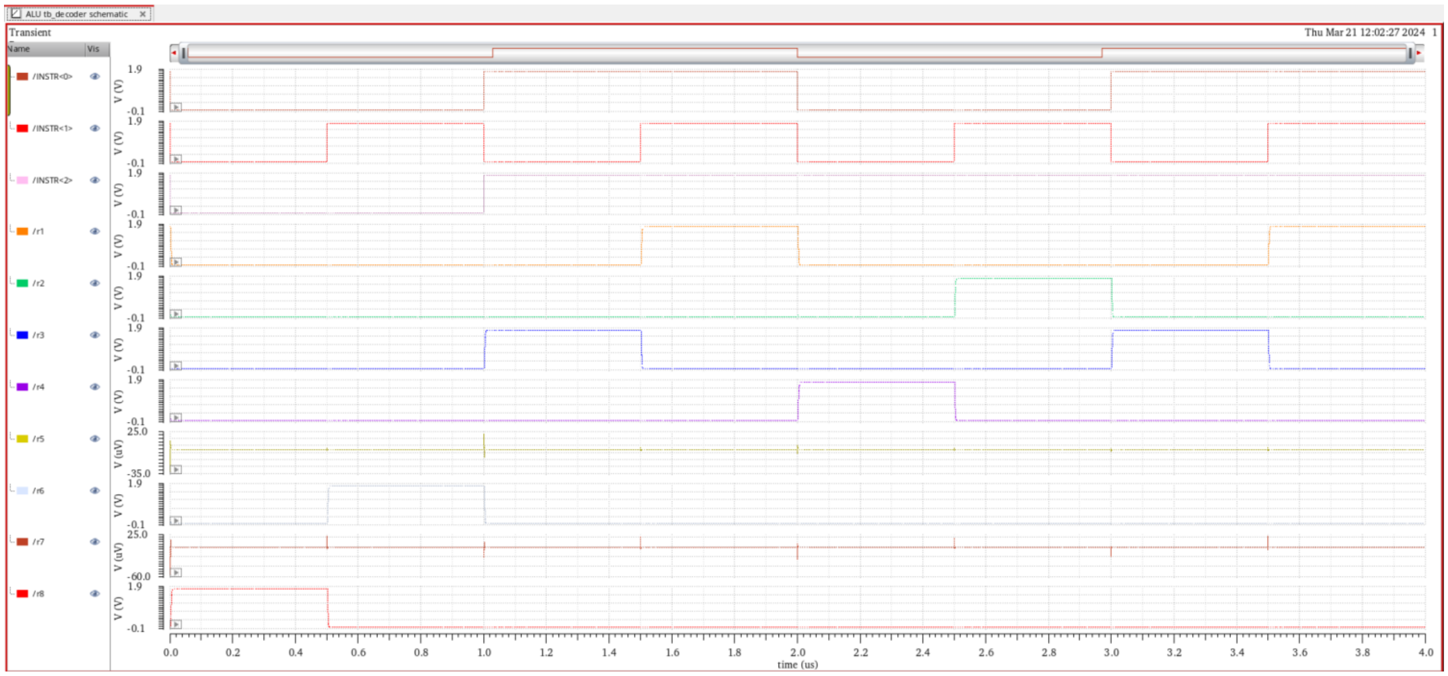


Fig 10. Decoder Testbench Results

In the given results, it's important to understand that R1 corresponds to word line 8, and R8 corresponds to word line 1. When the instruction INSTR<2:0> is 000, we observe that R8 is high while all other R's are low. This indicates that a signal has been sent to word line 1.

4-Bit Adder/Subtractor

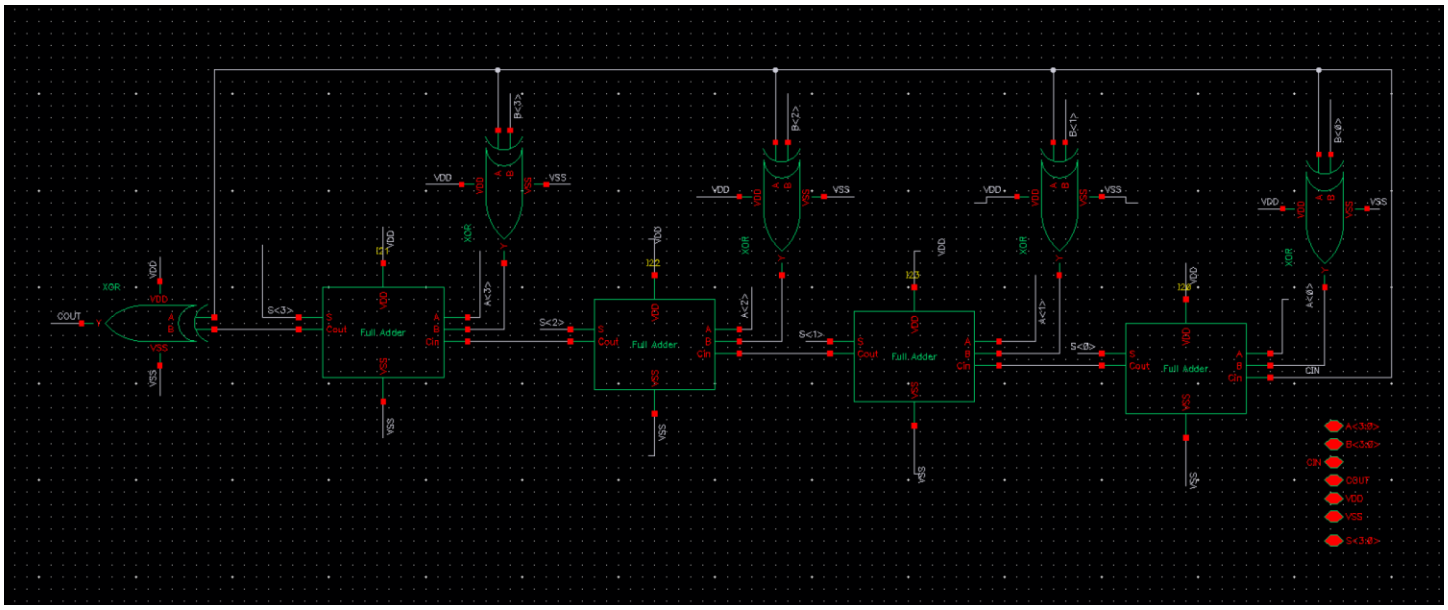


Fig 11. 4-bit adder/subtractor schematic

1-Bit Adder Full Adder

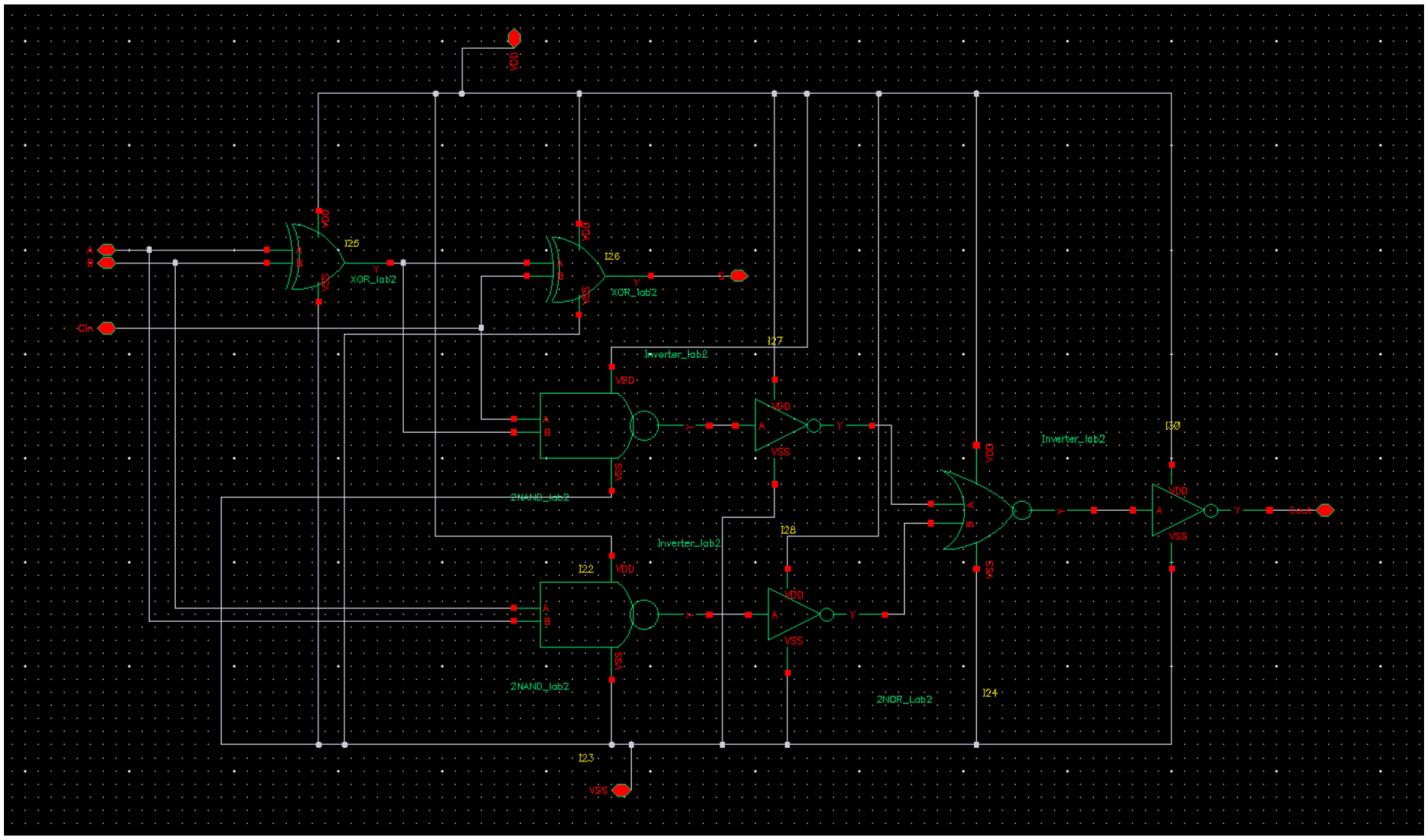


Fig 12. 1-bit adder schematic

Adder/Subtractor explained

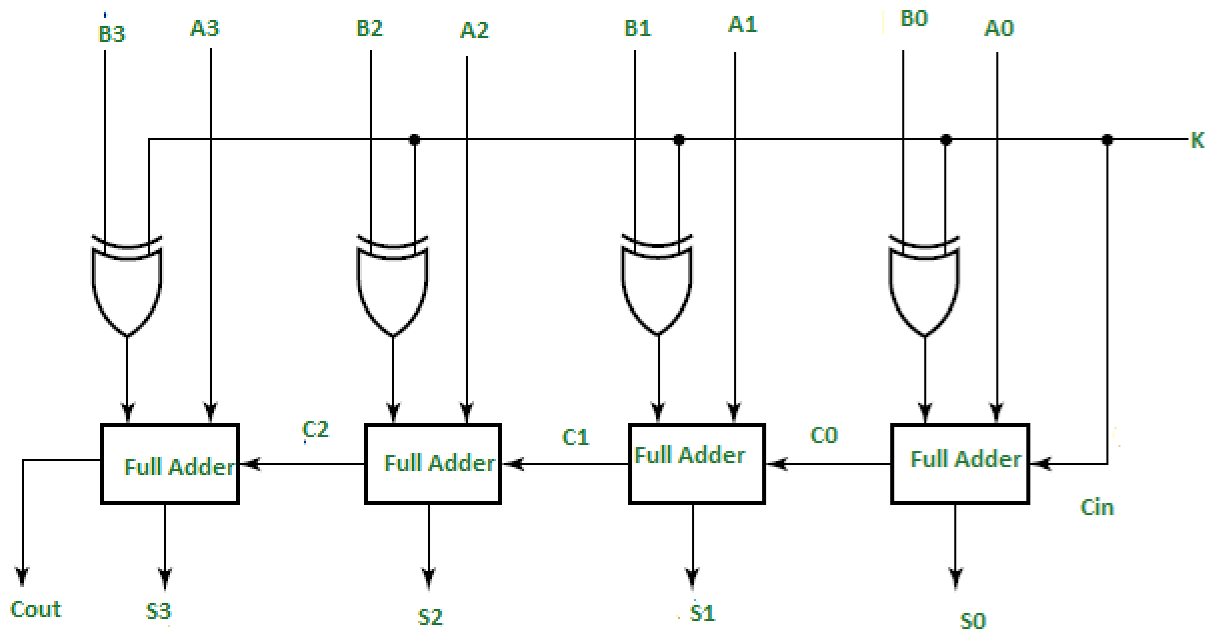


Fig 13. Adder/Subtractor Schematic [1]

From the geeksforgeeks article titled “4-bit binary Adder-Subtractor”, in the setup illustrated in the figure, the first full adder takes in three inputs: the input carry (Cin), the least significant bit of A (A0), and the result of an exclusive OR operation between B0 and K. It produces two outputs: the sum or difference (S0) and the carry out (C0).

If K (the control line) is 1, the output of the exclusive OR operation between B0 and K equals the complement of B0, meaning we're performing A + (B0'), indicating subtraction. For 2’s complement subtraction of A and B, the formula is A + B’ + Cin.

Conversely, if K equals 0, the output of the exclusive OR operation is simply B0. This implies we're performing A + B, which is binary addition.

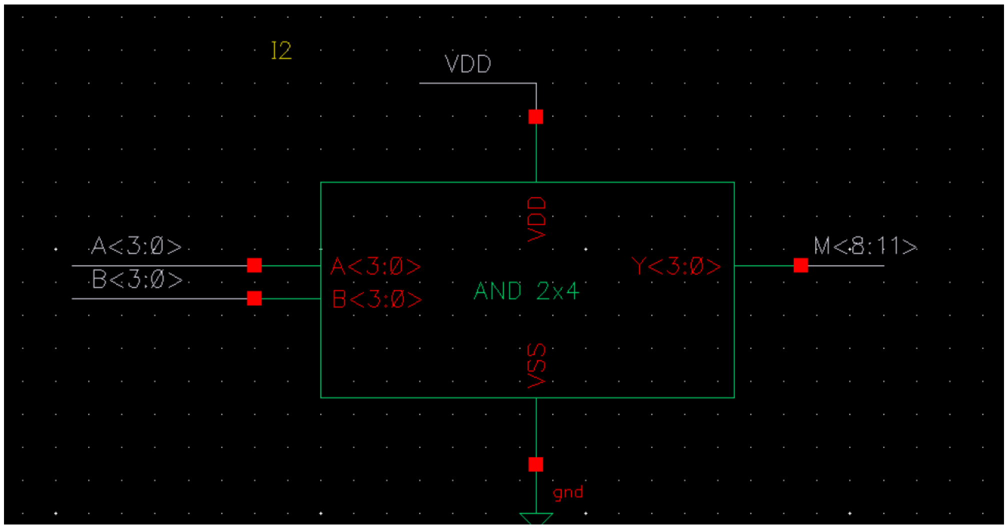
Then, C0 is sequentially fed into the second full adder as one of its inputs. The sum or difference (S0) is recorded as the least significant bit of the result. A1, A2, and A3 are directly inputted into the second, third, and fourth full adders, respectively. The third inputs are the exclusive OR of B1, B2, and B3 with K for the second, third, and fourth full adders, respectively.

The carries C1 and C2 are passed on to the successive full adders as inputs. C3 becomes the total carry for the sum or difference. S1, S2, and S3 are recorded to complete the result along with S0.

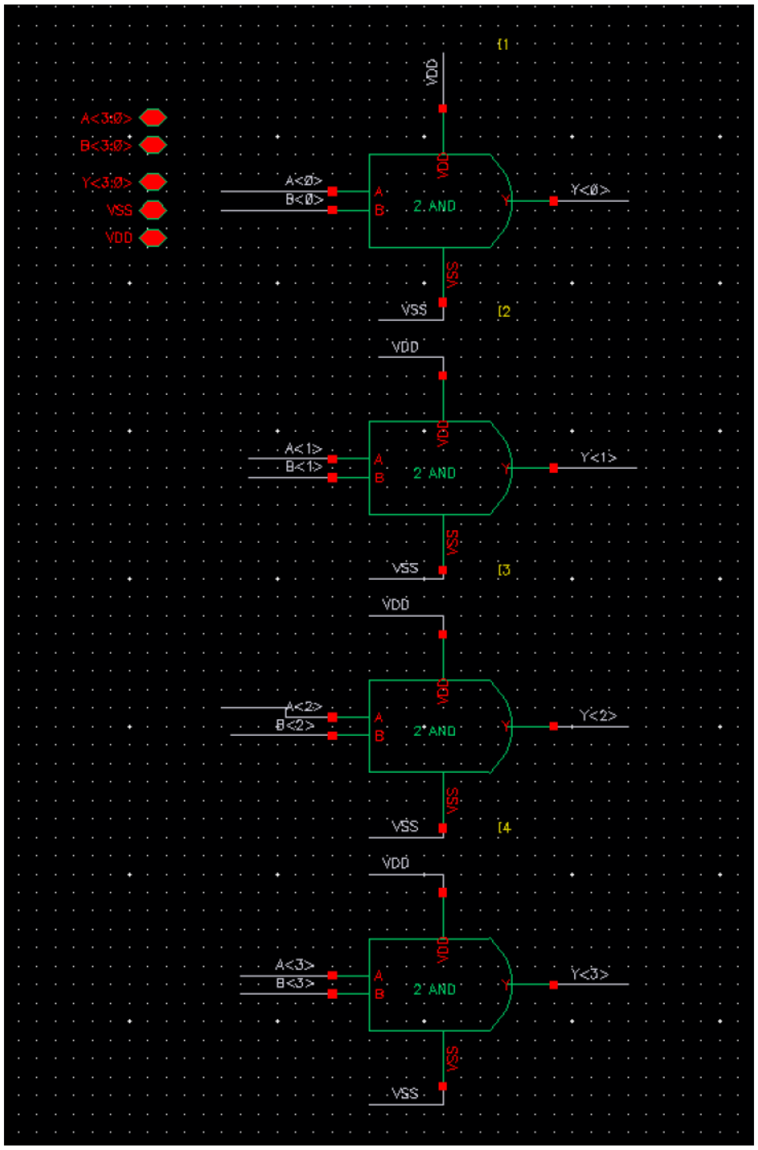
For an n-bit binary adder-subtractor, the setup employs n full adders. [1]

The **ALU uses a 1-bit ripple carry adder**, The delay in obtaining the output is significant because the process of calculating the final result takes a considerable amount of time. This delay is due to several design issues inherent in the ripple carry adder system. The complexity arises from the extensive calculation involved in determining the end output, making the overall process intricate and time-consuming. **A faster solution for an improvement would be swapping this out with a mirror adder**.

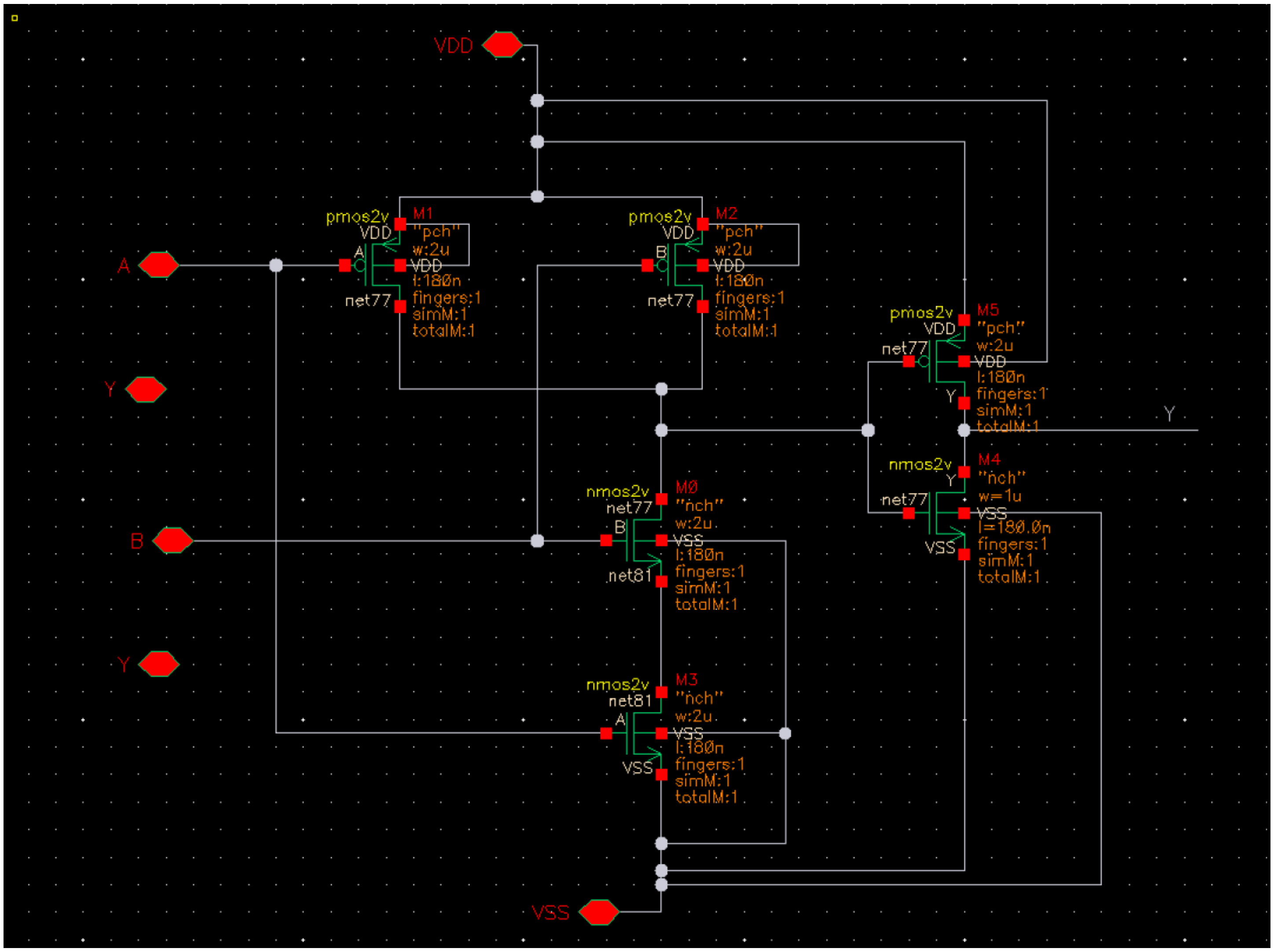
2:4 AND Symbol



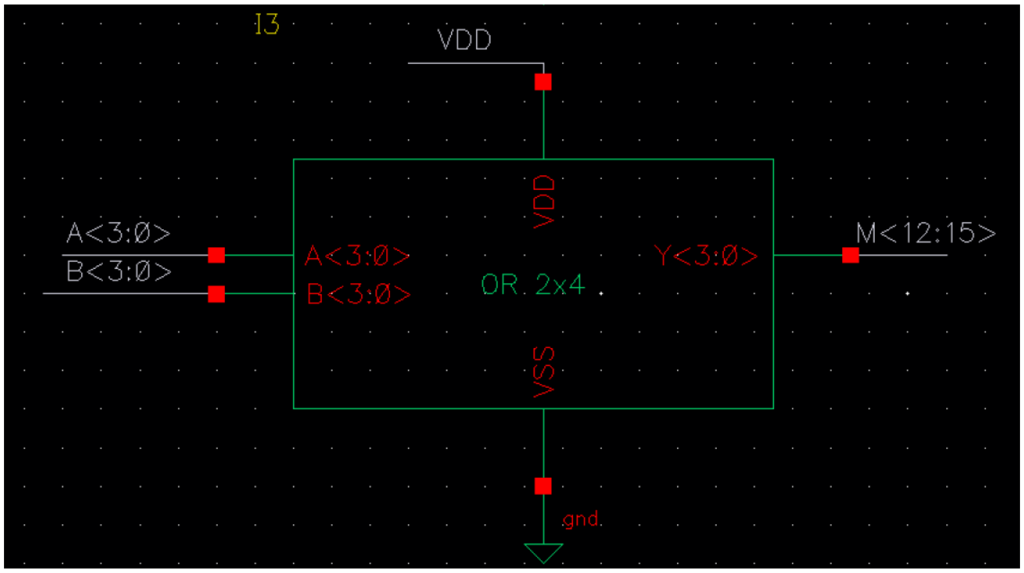
2:4 AND Schematic



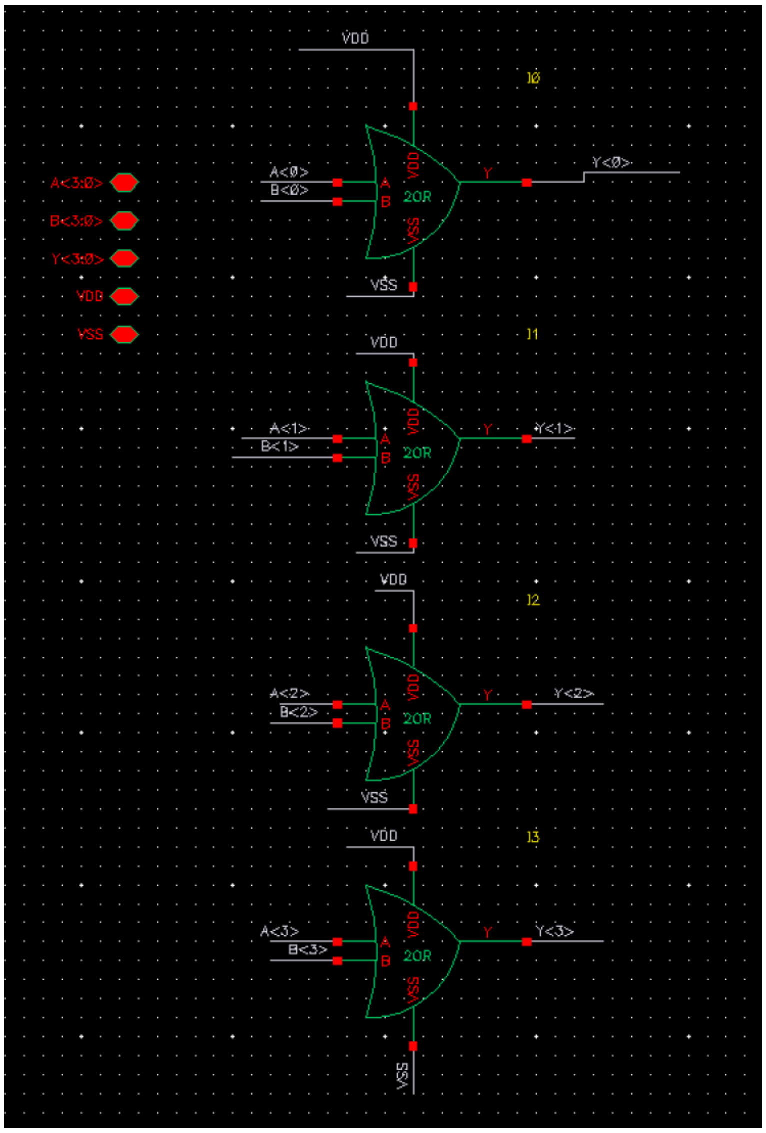
2:1 AND Schematic Transistor Level


VDD
A
Y
B
Y
VSS
pmos2v M1 "pch" w:2u l:180n fingers:1 simM:1 totalM:1
pmos2v M2 "pch" w:2u l:180n fingers:1 simM:1 totalM:1
pmos2v M5 "pch" w:2u l:180n fingers:1 simM:1 totalM:1
nmos2v M0 "nch" w:2u l:180n fingers:1 simM:1 totalM:1
nmos2v M3 "nch" w:2u l:180n fingers:1 simM:1 totalM:1
nmos2v M4 "nch" w=1u l=180.0n fingers:1 simM:1 totalM:1
net77
net81

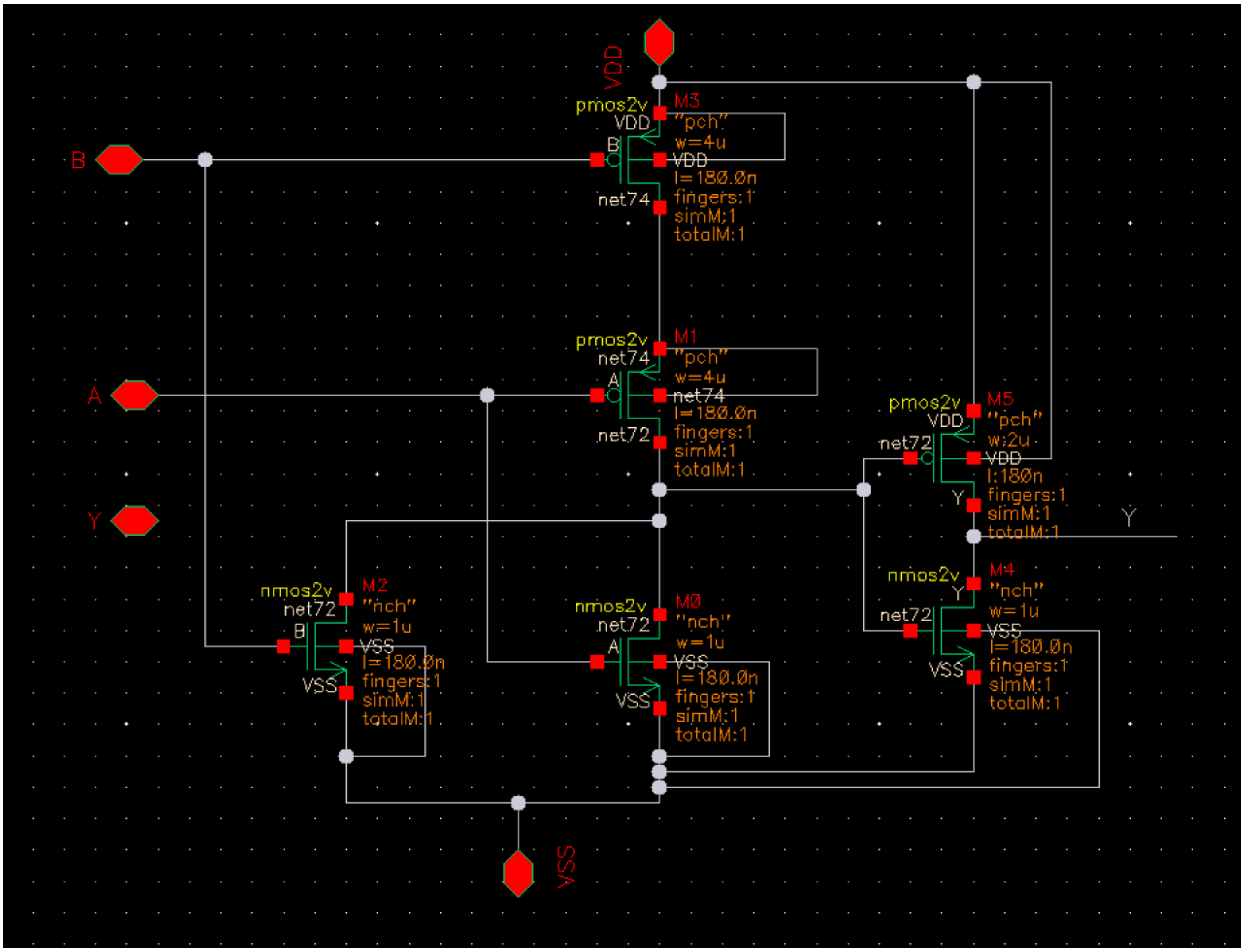
2:4 OR Symbol



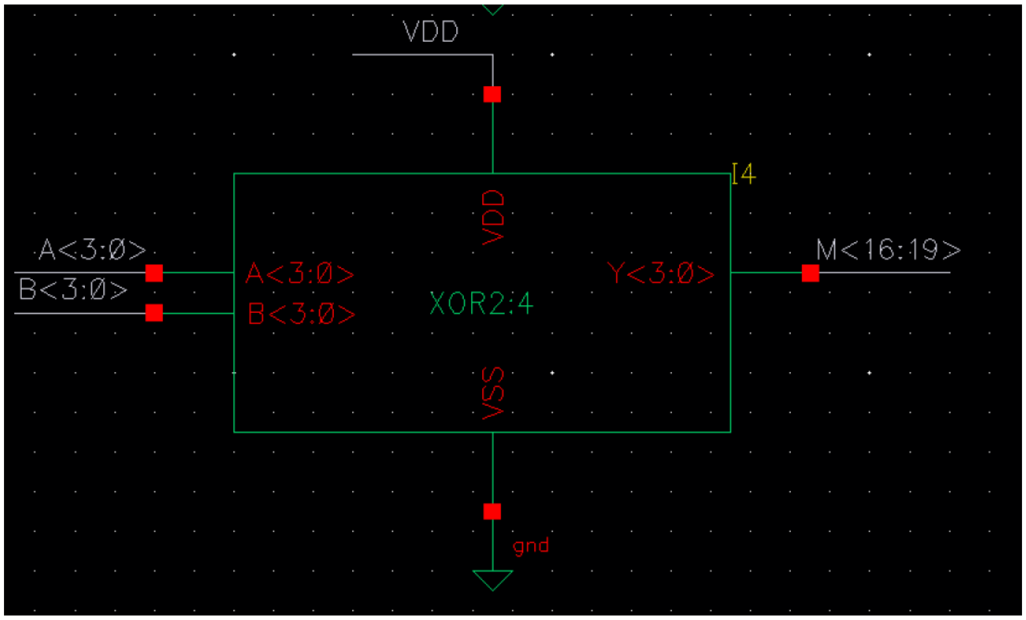
2:4 OR Schematic



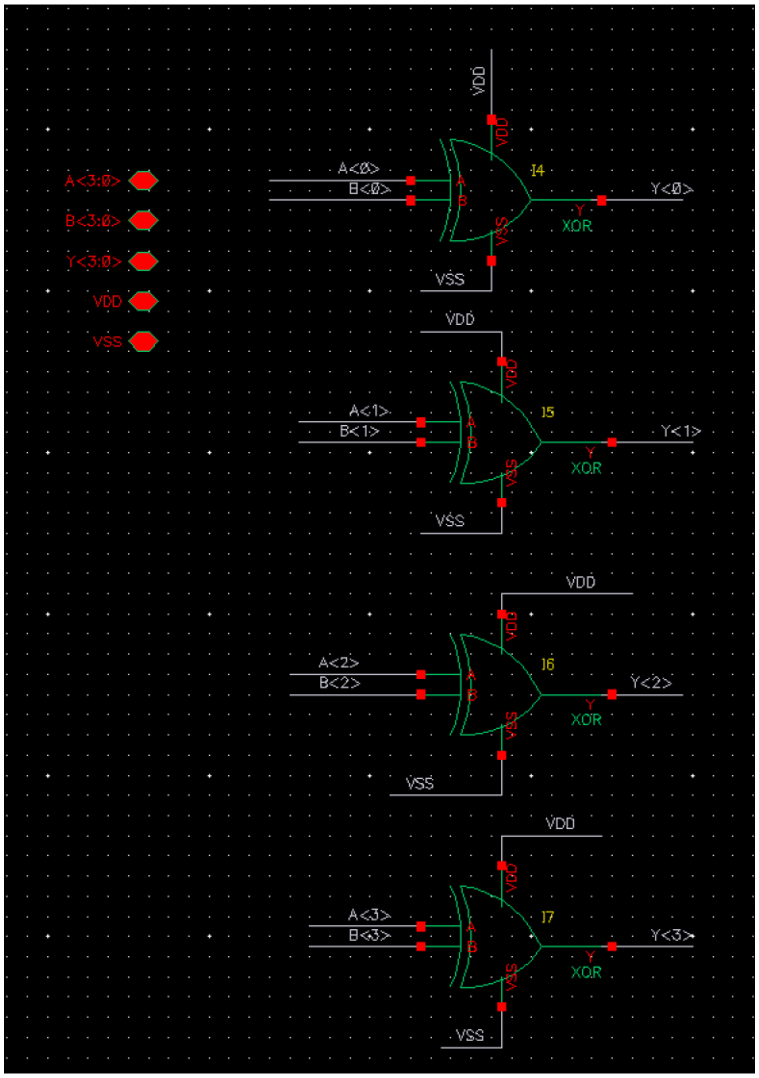
2:1 OR Schematic Transistor Level



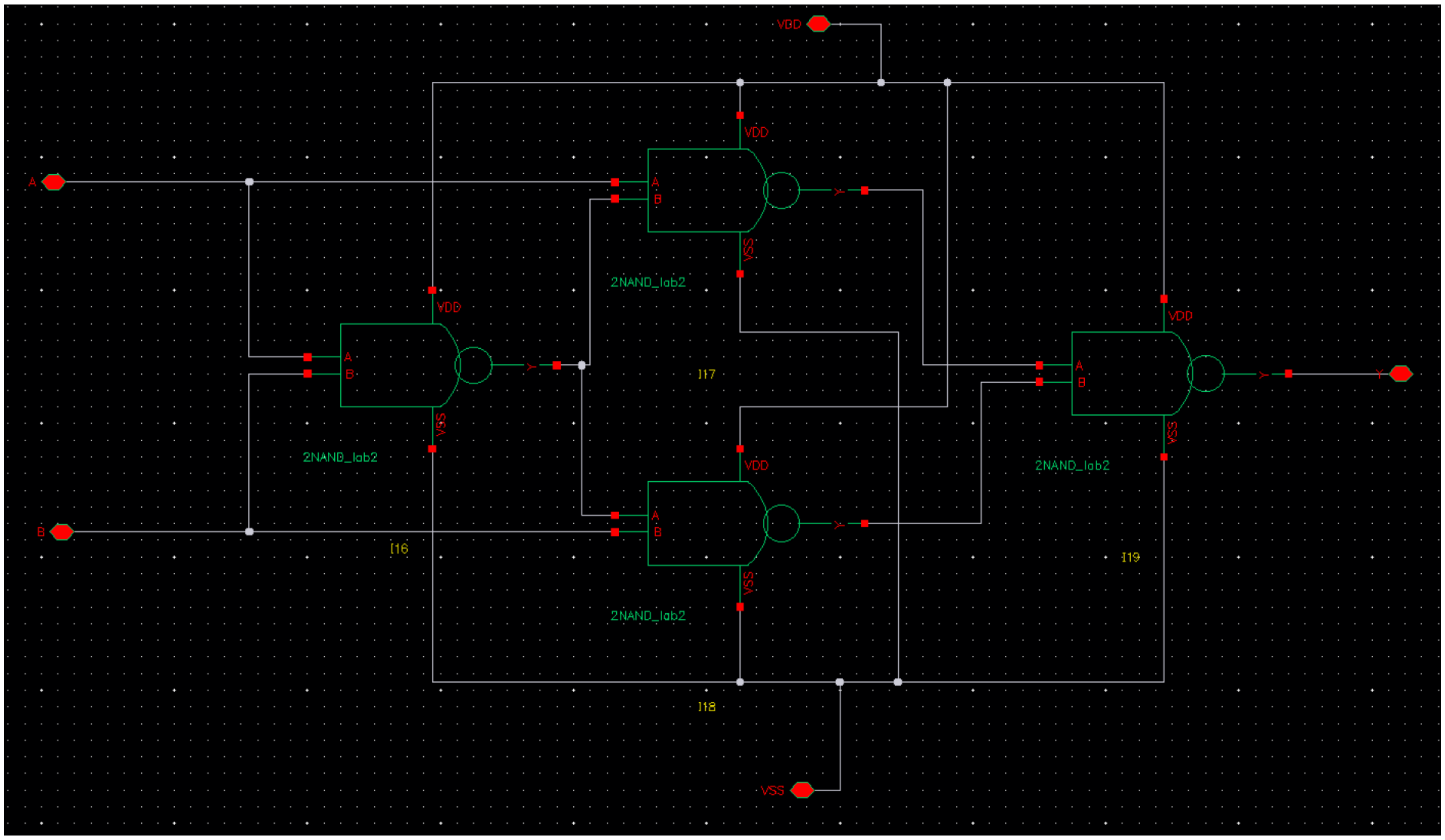
2:4 XOR Symbol



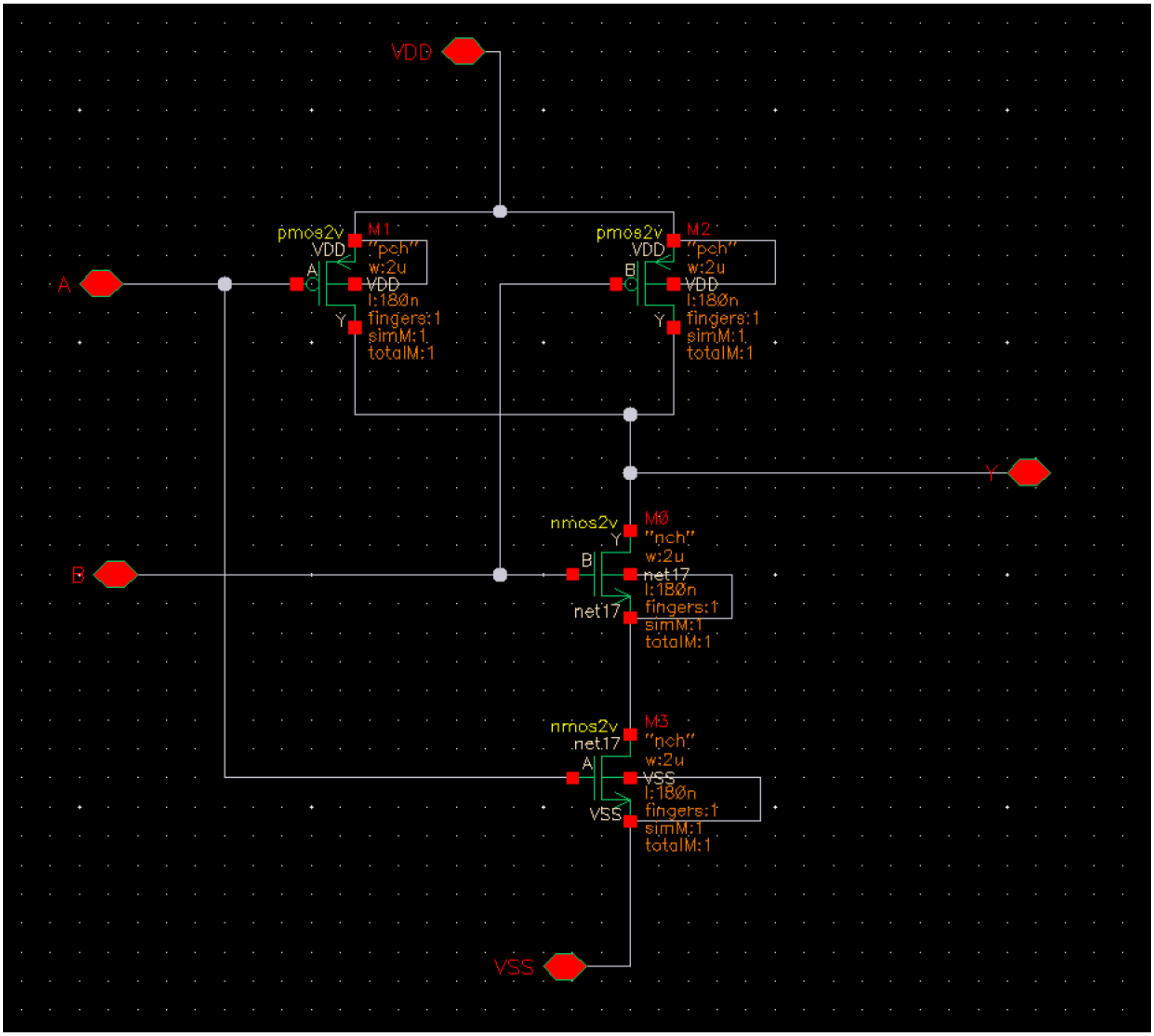
2:4 XOR Schematic



2:1 XOR Schematic



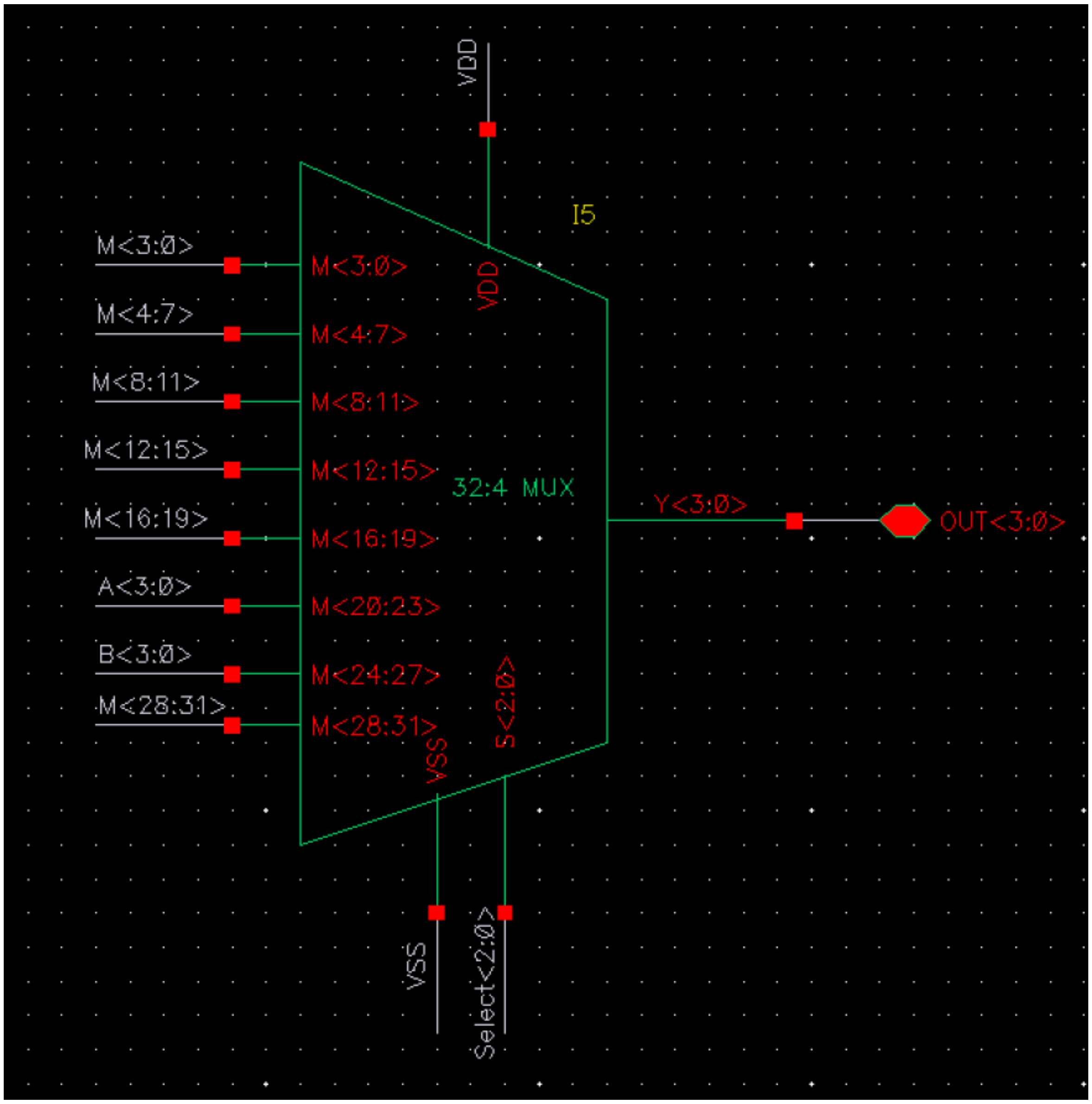
2:1 NAND Transistor Level



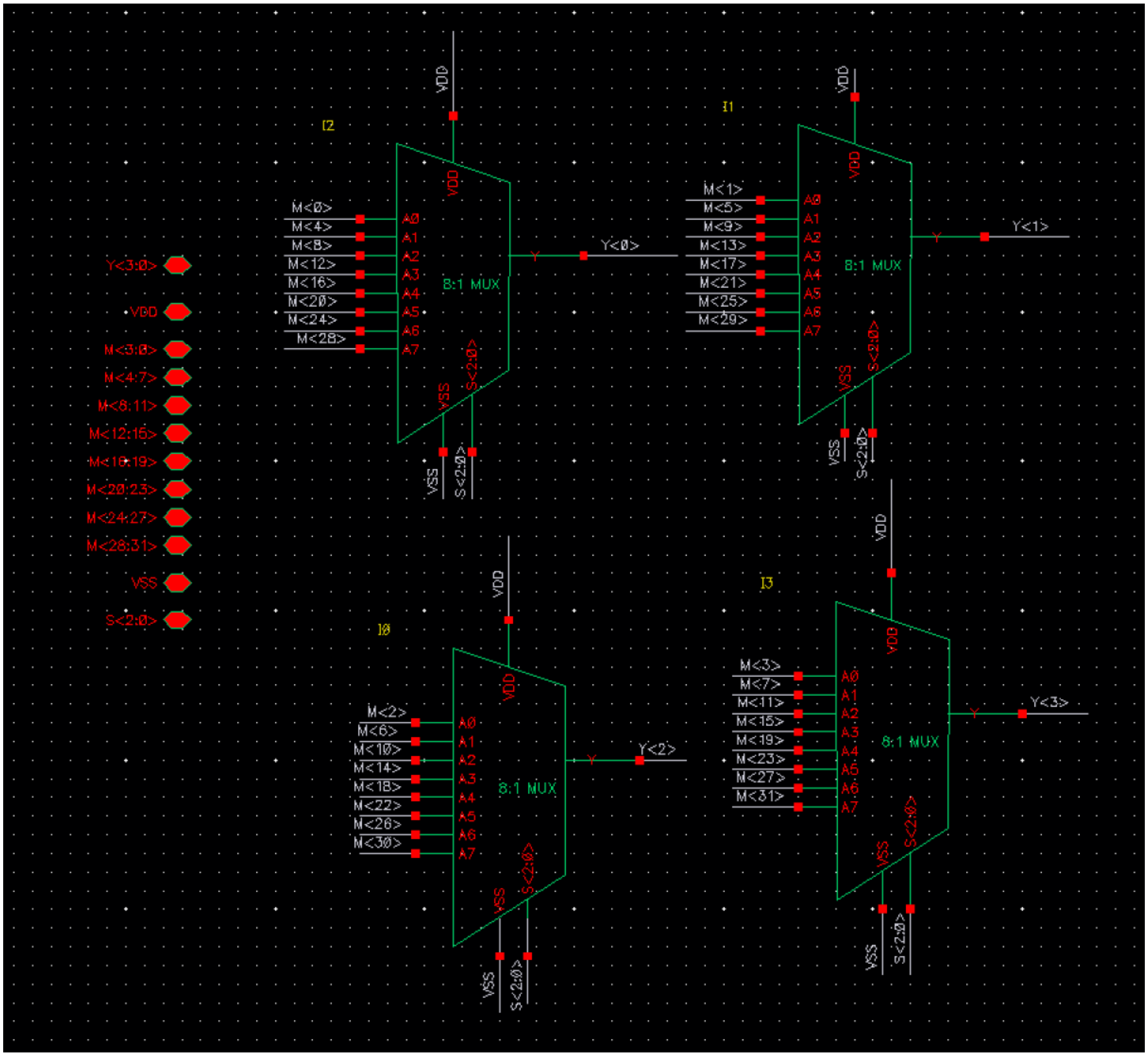
AND, OR, XOR Explained

These images clearly layout AND, OR and XOR gates in CMOS technology. These sizes for the transistors are already optimized. To make the gates faster, smaller transistors are required. Other ways to make the gates faster is to move away from planar transistor architectures i.e. use 3D devices, such as gate-all-around (GAA) architectures.

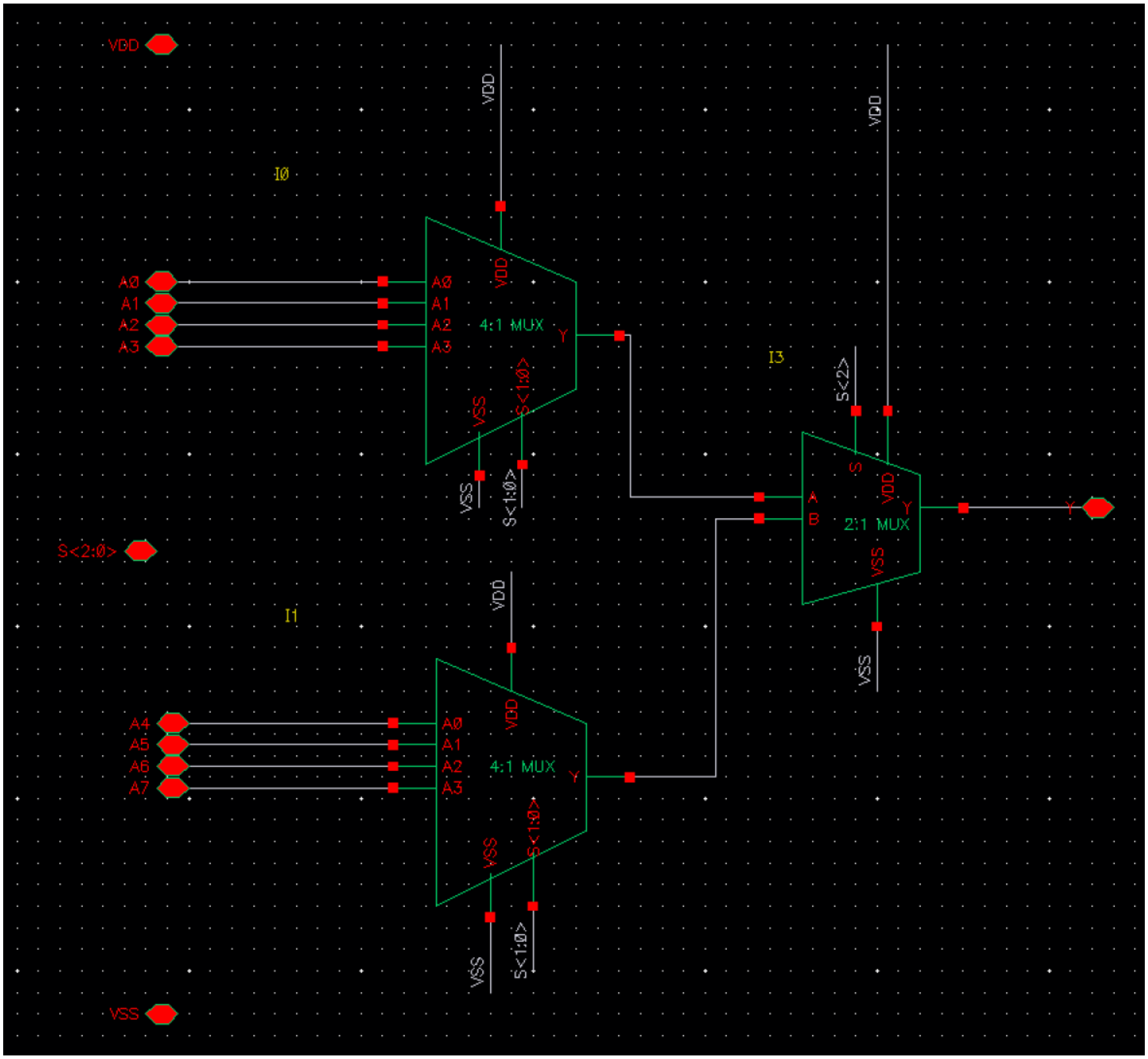
32:4 Mux


VDD
I5
M<3:0>
M<4:7>
M<8:11>
M<12:15>
M<16:19>
A<3:0>
B<3:0>
M<28:31>
M<3:0>
VDD
M<4:7>
M<8:11>
M<12:15>
32:4 MUX
M<16:19>
M<20:23>
M<24:27>
M<28:31>
S<2:0>
VSS
Y<3:0>
OUT<3:0>
VSS
Select<2:0>

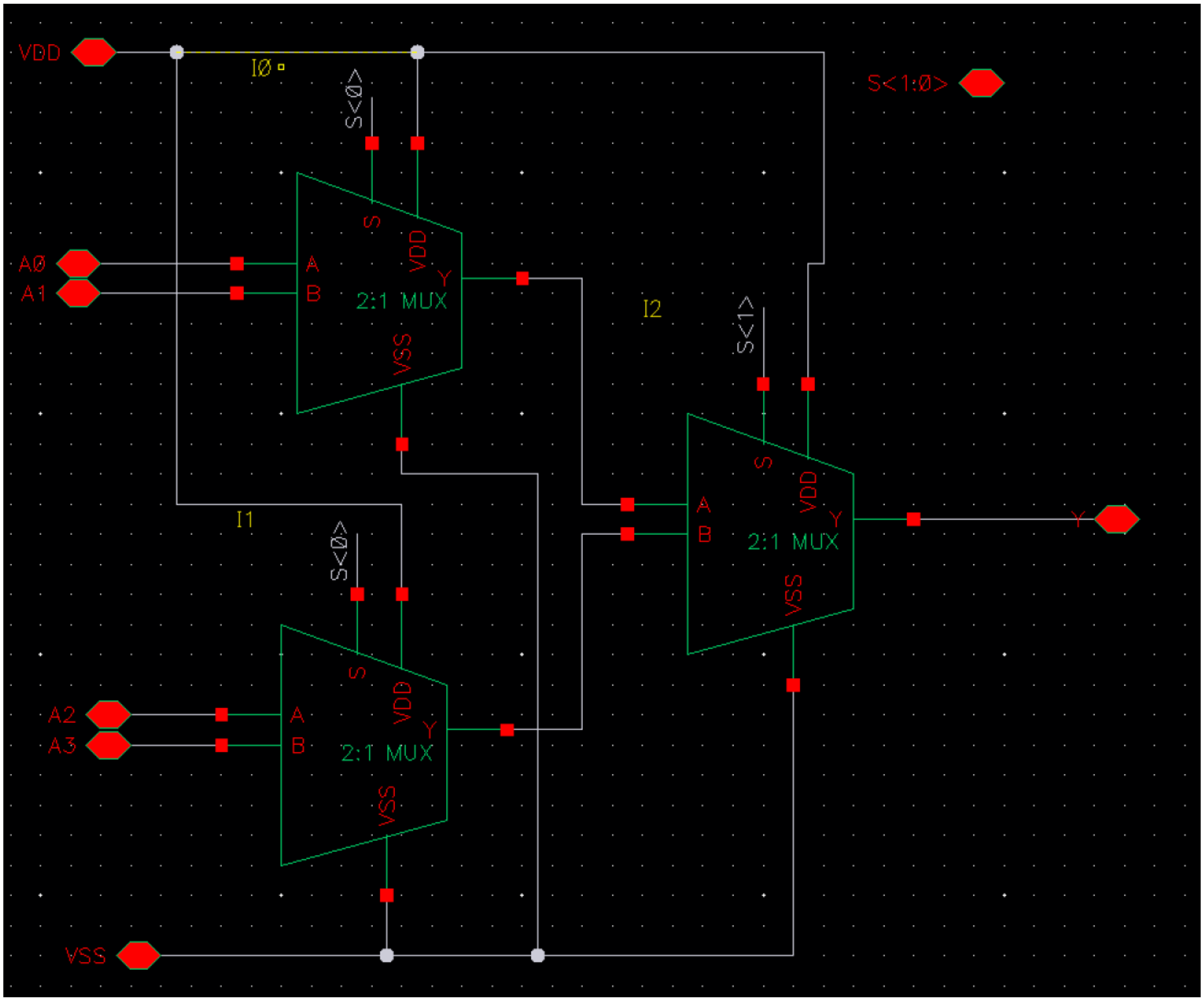
32:4 Mux Schematic



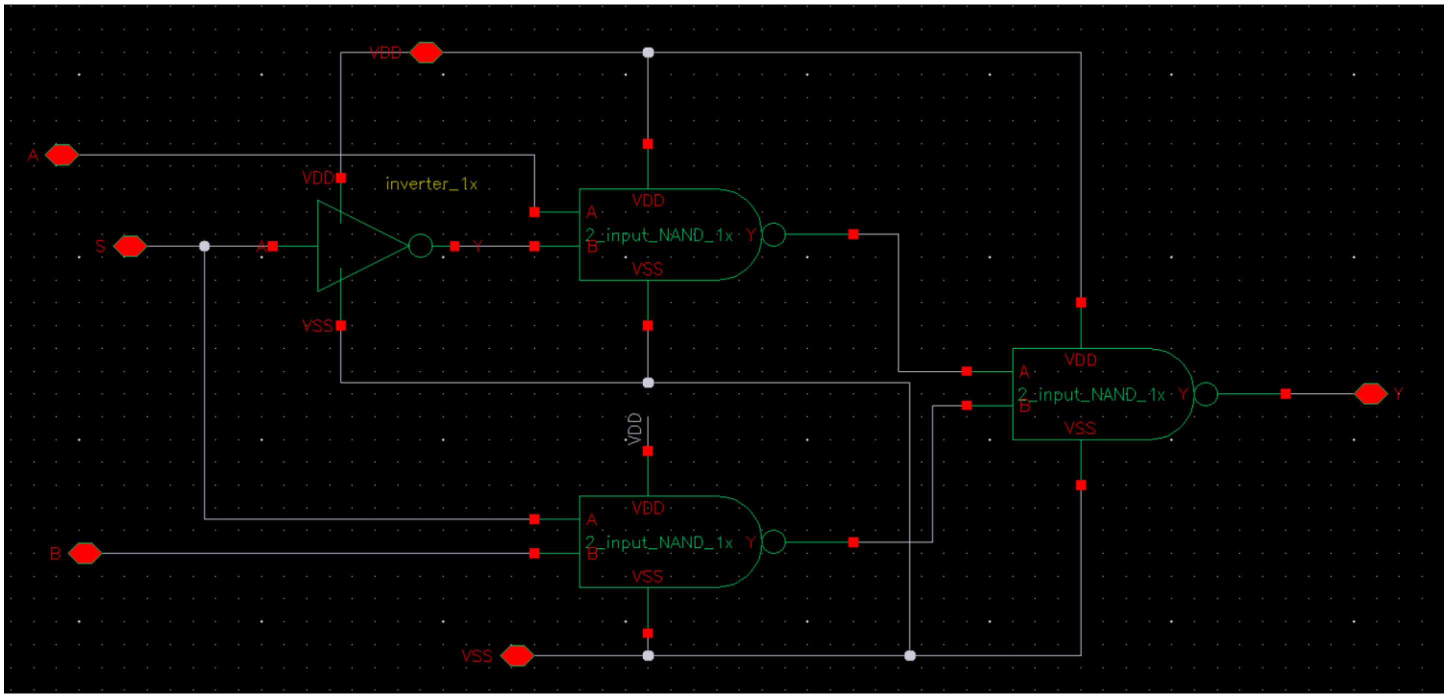
8:1 Mux


VDD
I0
A0
A1
A2
A3
4:1 MUX
Y
VSS
S<1:0>
S<2:0>
I3
S<2>
S
A
B
2:1 MUX
I1
A4
A5
A6
A7
VSS

4:1 Mux



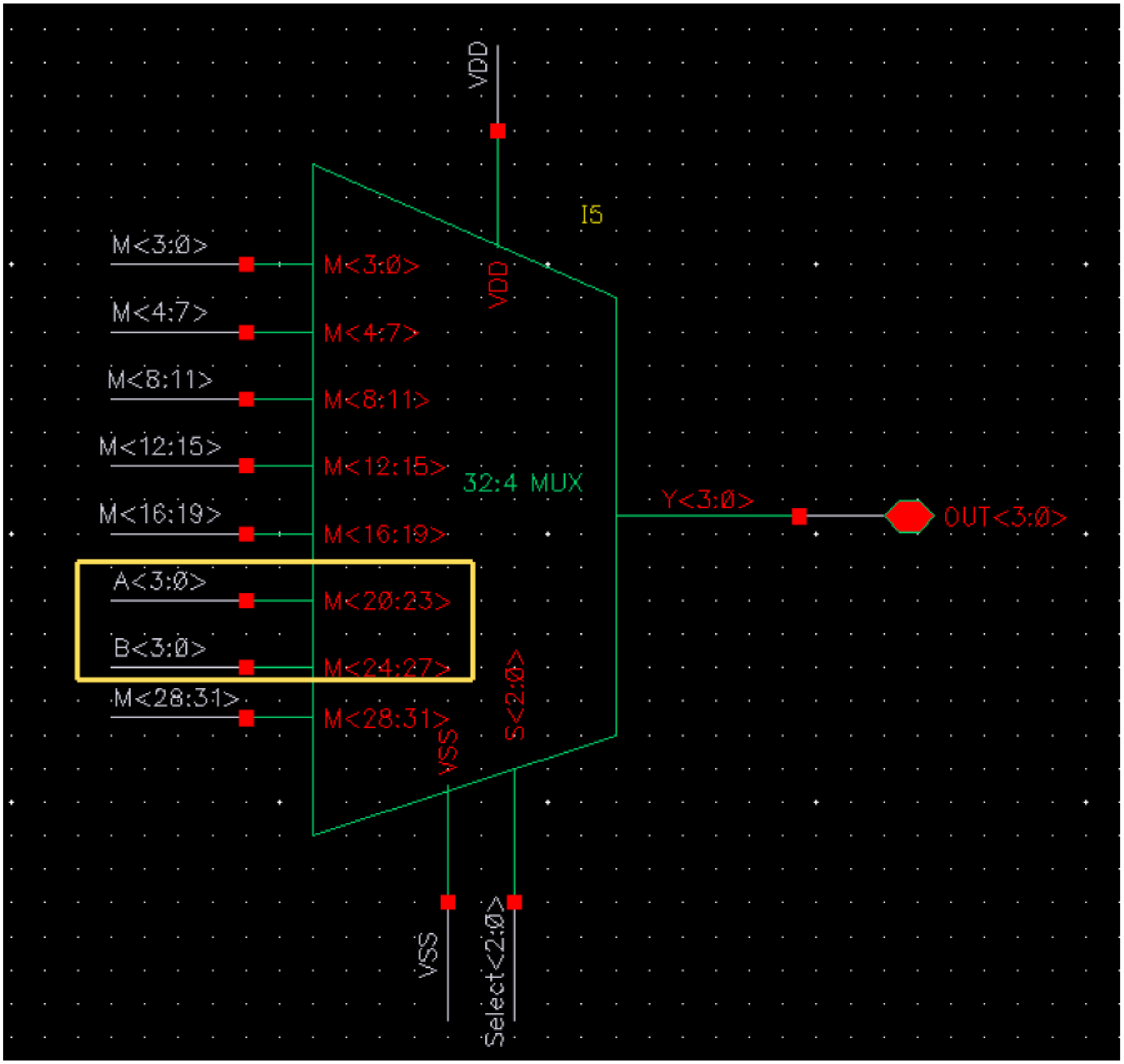
2:1 Mux



Mux Explained

A 32:4 multiplexer (mux) is a digital circuit that selects one out of 32 input lines based on 4 select lines and forwards the selected input to a single output line. It operates by using binary encoding on the select lines to determine which input line to pass to the output. The multiplexer's schematic includes input lines (32), select lines (4), and one output line. The internal structure involves cascading smaller multiplexers to handle the large number of inputs.

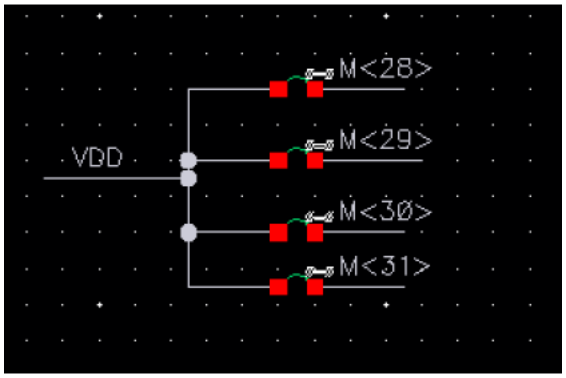
PASS A & PASS B



PASS A & PASS B Explained

The input to the mux that are called when the subsequent instruction is called is tied to either A or B.

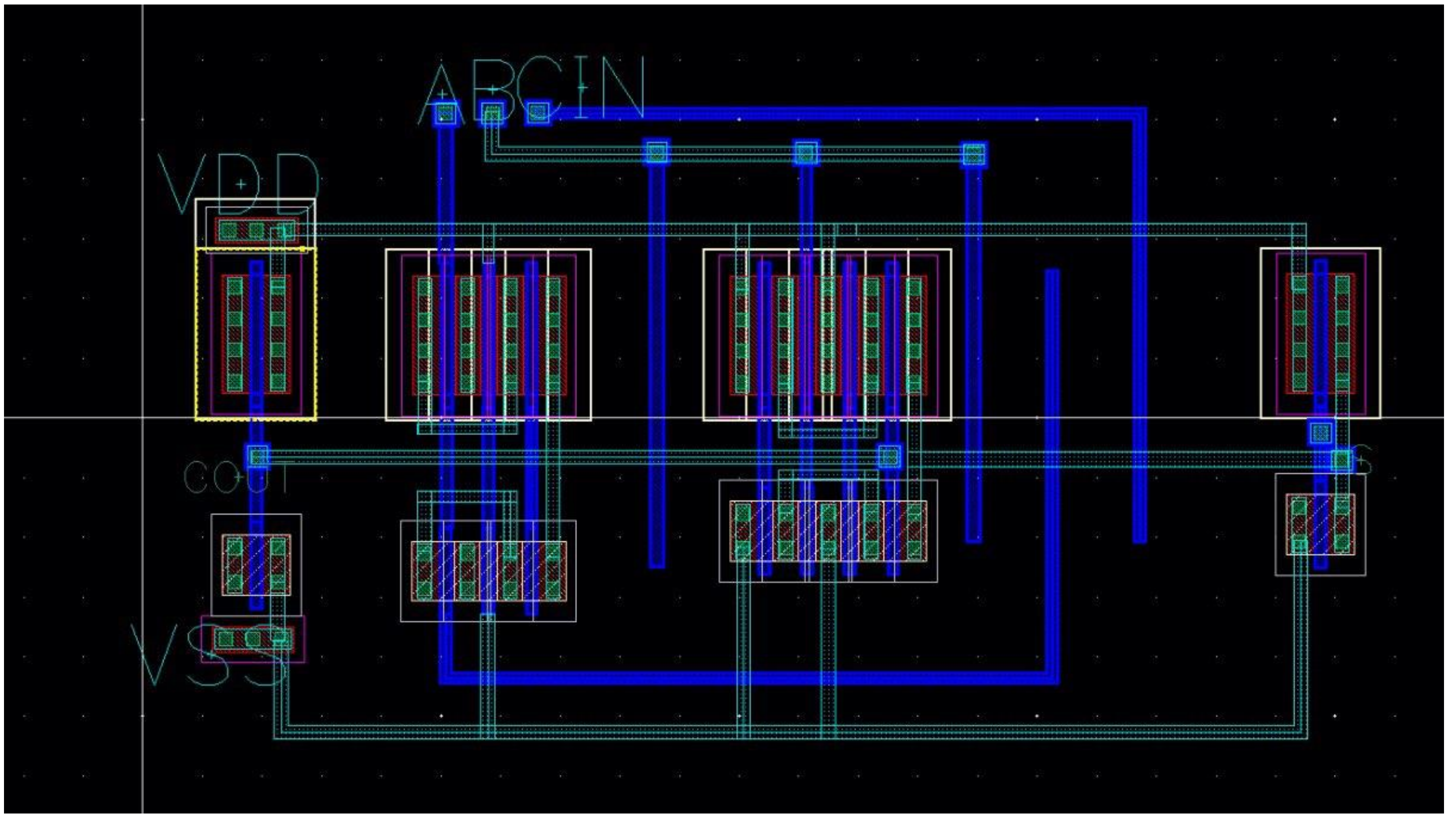
ALL HIGH



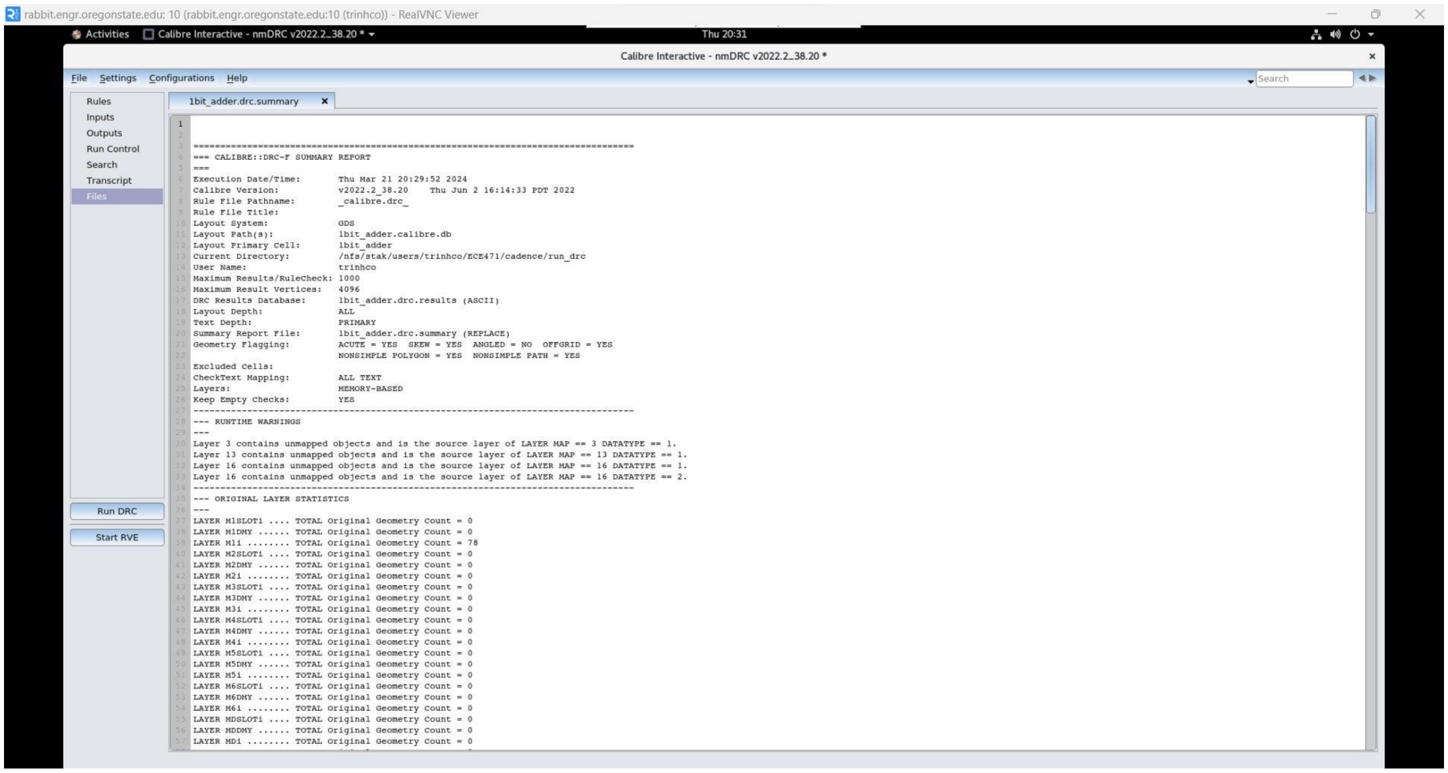
PASS A & PASS B Explained

The input to the mux that are called when the subsequent instruction is called is tied to either A or B.

Layout

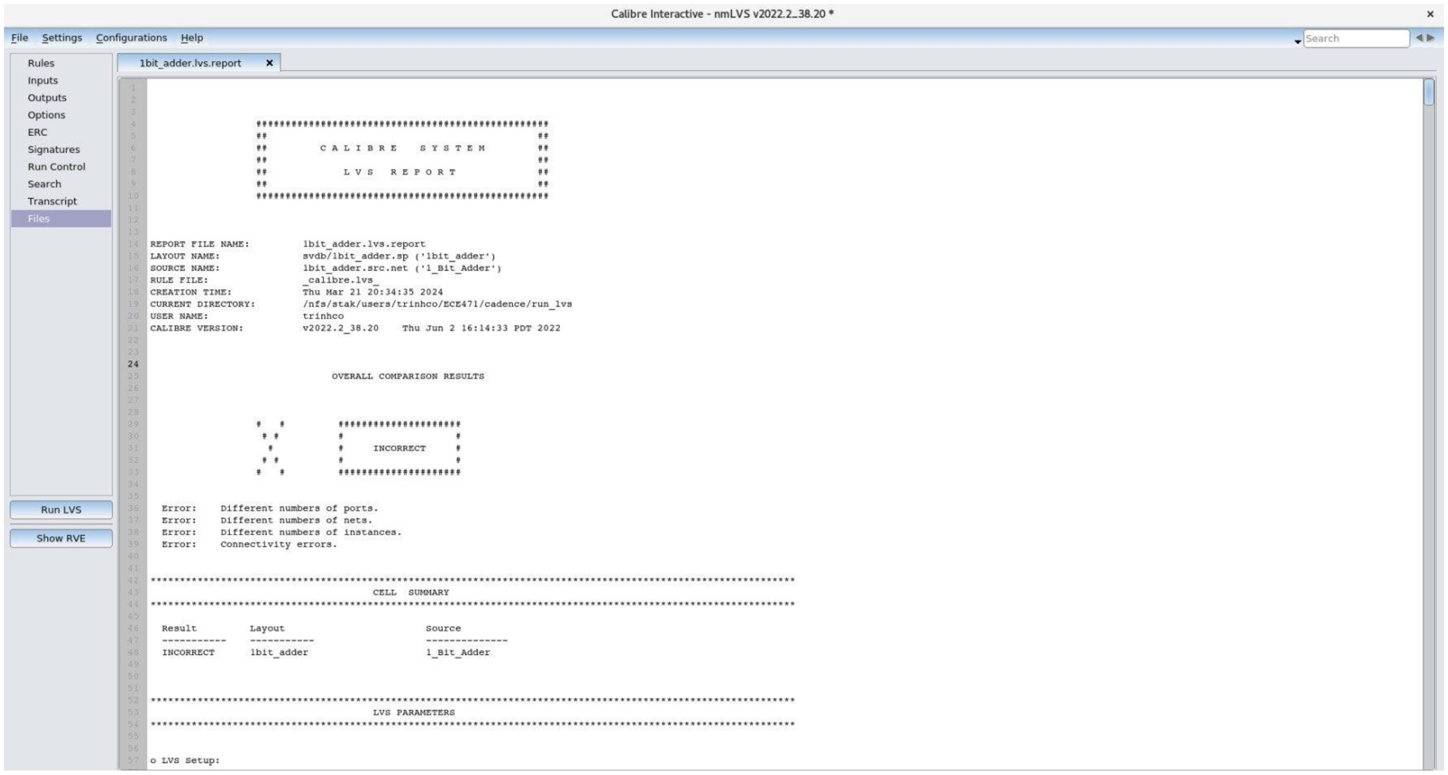


DRC Layout


rabbit.engr.oregonstate.edu: 10 (rabbit.engr.oregonstate.edu:10 (trinhco)) - RealVNC Viewer
Activities
Calibre Interactive - nmDRC v2022.2_38.20 *
Thu 20:31
Calibre Interactive - nmDRC v2022.2_38.20 *
File
Settings
Configurations
Help
Search
Rules
Inputs
Outputs
Run Control
Search
Transcript
Files
Run DRC
Start RVE
1bit_adder.drc.summary

```

==================================================================================
=== CALIBRE::DRC-F SUMMARY REPORT
===
Execution Date/Time:       Thu Mar 21 20:29:52 2024
Calibre Version:           v2022.2_38.20    Thu Jun 2 16:14:33 PDT 2022
Rule File Pathname:        _calibre.drc_
Rule File Title:
Layout System:             GDS
Layout Path(s):            1bit_adder.calibre.db
Layout Primary Cell:       1bit_adder
Current Directory:         /nfs/stak/users/trinhco/ECE471/cadence/run_drc
User Name:                 trinhco
Maximum Results/RuleCheck: 1000
Maximum Result Vertices:   4096
DRC Results Database:      1bit_adder.drc.results (ASCII)
Layout Depth:              ALL
Text Depth:                PRIMARY
Summary Report File:       1bit_adder.drc.summary (REPLACE)
Geometry Flagging:         ACUTE = YES  SKEW = YES  ANGLED = NO  OFFGRID = YES
                           NONSIMPLE POLYGON = YES  NONSIMPLE PATH = YES
Excluded Cells:
CheckText Mapping:         ALL TEXT
Layers:                    MEMORY-BASED
Keep Empty Checks:         YES
----------------------------------------------------------------------------------
--- RUNTIME WARNINGS
---
Layer 3 contains unmapped objects and is the source layer of LAYER MAP == 3 DATATYPE == 1.
Layer 13 contains unmapped objects and is the source layer of LAYER MAP == 13 DATATYPE == 1.
Layer 16 contains unmapped objects and is the source layer of LAYER MAP == 16 DATATYPE == 1.
Layer 16 contains unmapped objects and is the source layer of LAYER MAP == 16 DATATYPE == 2.
----------------------------------------------------------------------------------
--- ORIGINAL LAYER STATISTICS
---
LAYER M1SLOTi .... TOTAL Original Geometry Count = 0
LAYER M1DMY ...... TOTAL Original Geometry Count = 0
LAYER M1i ........ TOTAL Original Geometry Count = 78
LAYER M2SLOTi .... TOTAL Original Geometry Count = 0
LAYER M2DMY ...... TOTAL Original Geometry Count = 0
LAYER M2i ........ TOTAL Original Geometry Count = 0
LAYER M3SLOTi .... TOTAL Original Geometry Count = 0
LAYER M3DMY ...... TOTAL Original Geometry Count = 0
LAYER M3i ........ TOTAL Original Geometry Count = 0
LAYER M4SLOTi .... TOTAL Original Geometry Count = 0
LAYER M4DMY ...... TOTAL Original Geometry Count = 0
LAYER M4i ........ TOTAL Original Geometry Count = 0
LAYER M5SLOTi .... TOTAL Original Geometry Count = 0
LAYER M5DMY ...... TOTAL Original Geometry Count = 0
LAYER M5i ........ TOTAL Original Geometry Count = 0
LAYER M6SLOTi .... TOTAL Original Geometry Count = 0
LAYER M6DMY ...... TOTAL Original Geometry Count = 0
LAYER M6i ........ TOTAL Original Geometry Count = 0
LAYER MDSLOTi .... TOTAL Original Geometry Count = 0
LAYER MDDMY ...... TOTAL Original Geometry Count = 0
LAYER MDi ........ TOTAL Original Geometry Count = 0
```

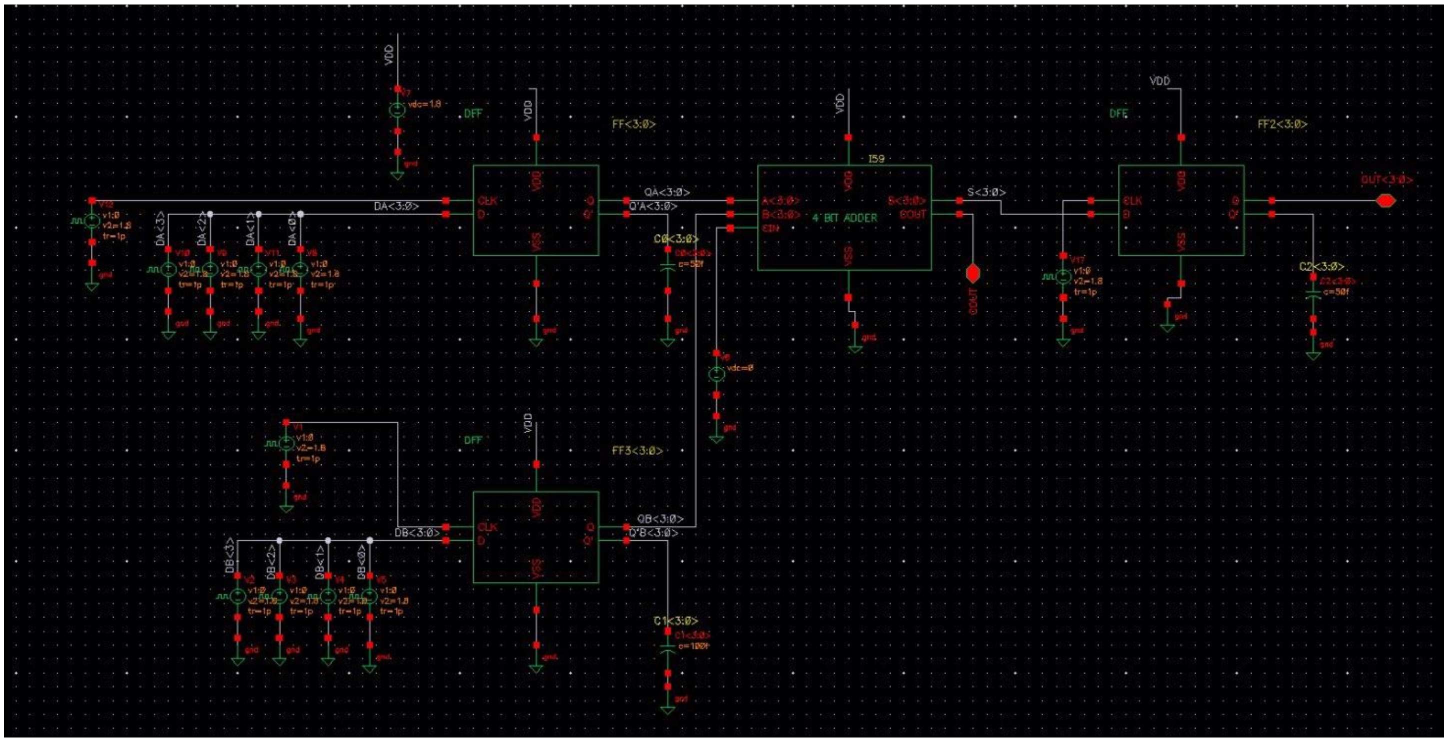
LVS Layout



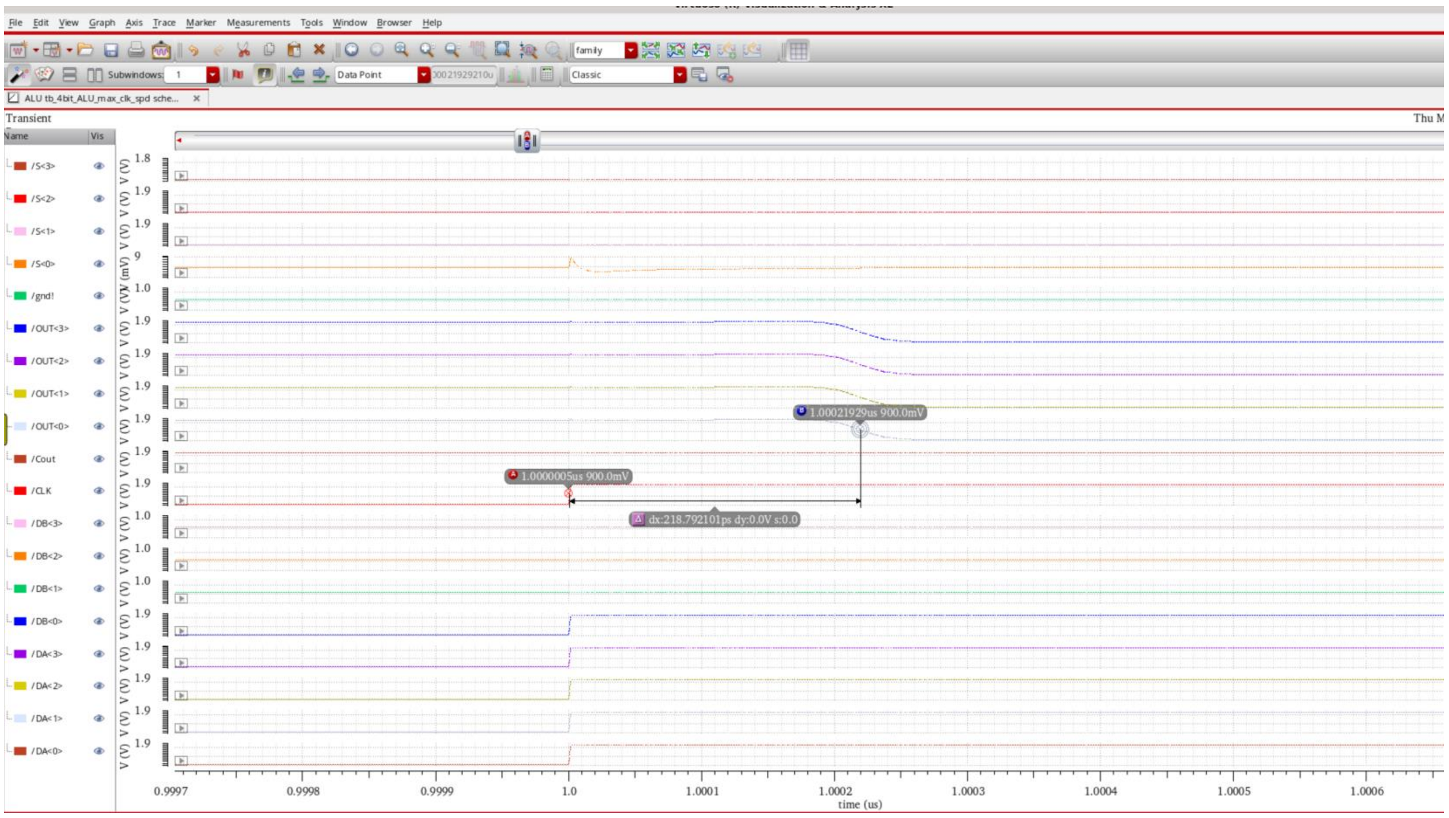
```
                  ##################################################
                  ##                                              ##
                  ##          C A L I B R E    S Y S T E M        ##
                  ##                                              ##
                  ##              L V S   R E P O R T             ##
                  ##                                              ##
                  ##################################################



REPORT FILE NAME:         1bit_adder.lvs.report
LAYOUT NAME:              svdb/1bit_adder.sp ('1bit_adder')
SOURCE NAME:              1bit_adder.src.net ('1_Bit_Adder')
RULE FILE:                _calibre.lvs_
CREATION TIME:            Thu Mar 21 20:34:35 2024
CURRENT DIRECTORY:        /nfs/stak/users/trinhco/ECE471/cadence/run_lvs
USER NAME:                trinhco
CALIBRE VERSION:          v2022.2_38.20    Thu Jun 2 16:14:33 PDT 2022



                               OVERALL COMPARISON RESULTS



                  #   #         #####################
                   # #          #                   #
                    #           #     INCORRECT     #
                   # #          #                   #
                  #   #         #####################


  Error:    Different numbers of ports.
  Error:    Different numbers of nets.
  Error:    Different numbers of instances.
  Error:    Connectivity errors.


**************************************************************************************************************
                                      CELL  SUMMARY
**************************************************************************************************************

  Result        Layout                        Source
  -----------   -----------                   --------------
  INCORRECT     1bit_adder                    1_Bit_Adder



**************************************************************************************************************
                                      LVS PARAMETERS
**************************************************************************************************************


o LVS Setup:
```

ADDITIONAL PROJECT REQUIREMENTS (571)

Requirement: Create a testbench for your 4-bit adder that includes flip-flops at the inputs (for A<3:0> and B<3:0>) and output (OUT<3:0>), so that your adder forms the combinational logic of a sequential design path between two clock edges. Load the output flip-flops with 50fF capacitance each to emulate the next logic stage this would feed into.



Requirement: Determine the input (A, B) configuration that leads to the worst-case propagation delay from input to output.



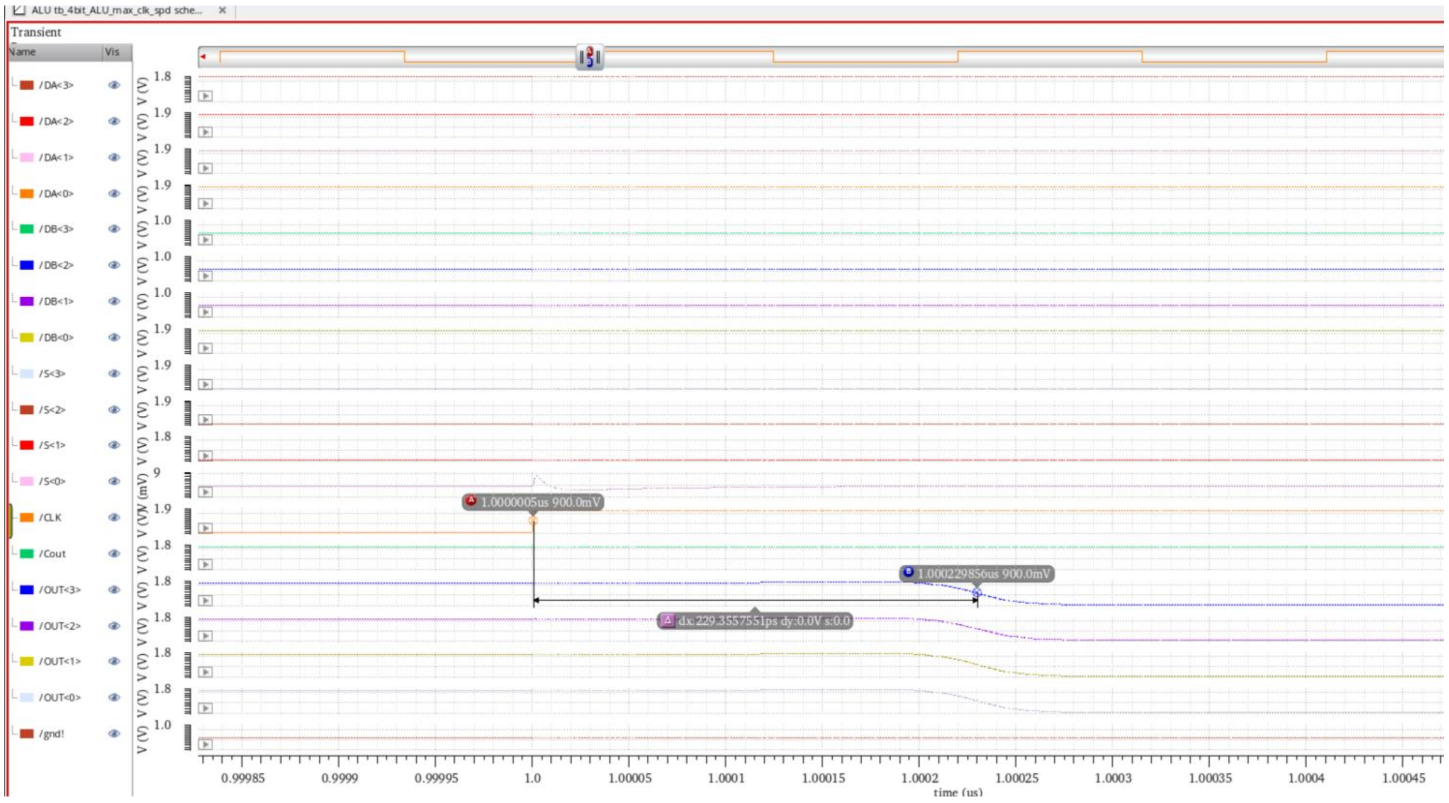
Propagation delay is 218.79ps from positive edge to output

Requirement: Simulate your logic path to find and demonstrate the maximum possible clock speed (or close to it) for VDD = {1.7 V, 1.8 V, 1.9 V}. Determine which of these supply levels is fastest (for worst-case computational path), and which is most efficient (i.e. least energy required for this worst-case computation).

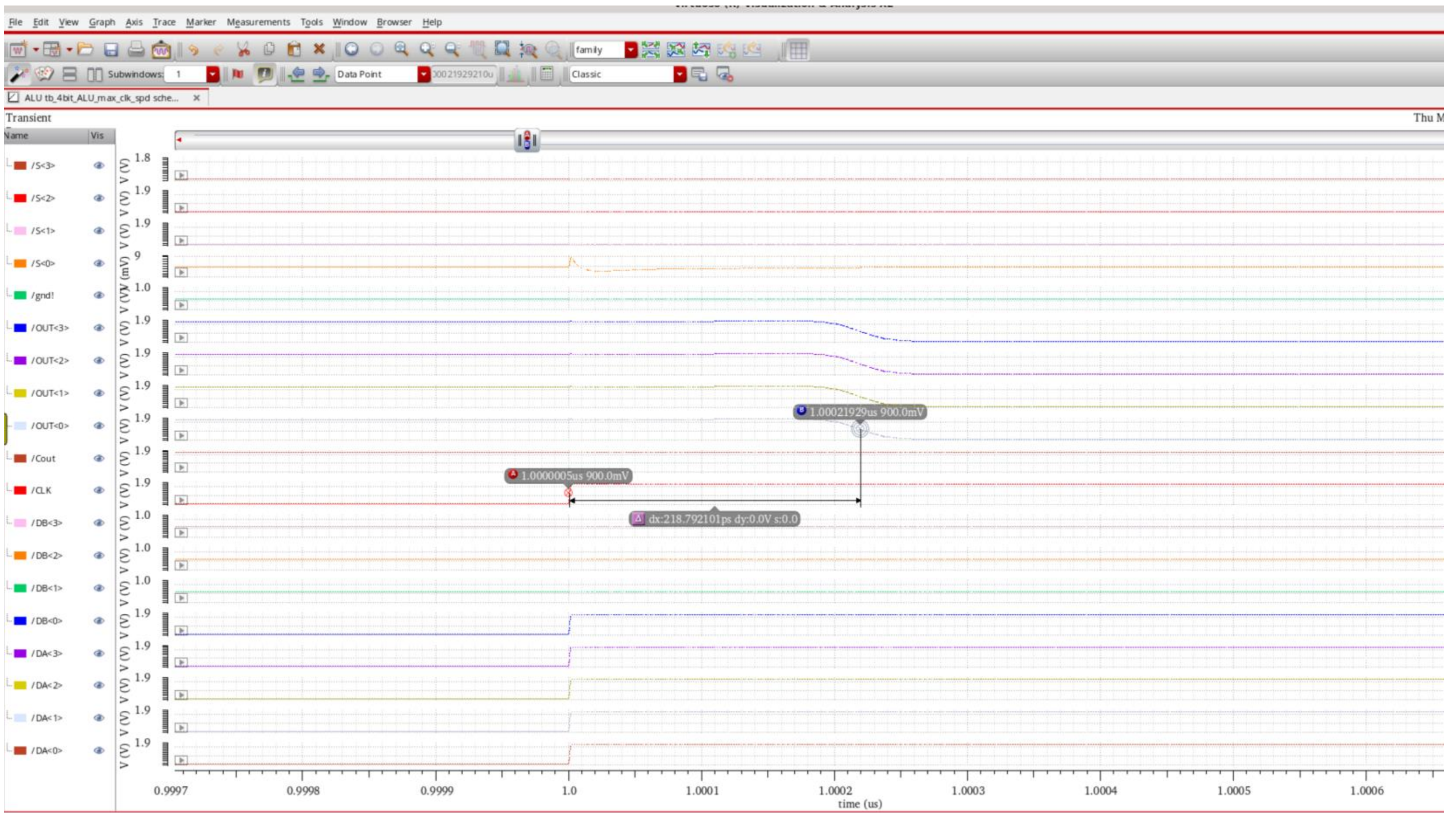
1111	1111
+0000	+0001
--------	-------
1111	0000
FAST	SLOW

We want to test for WORST case scenario

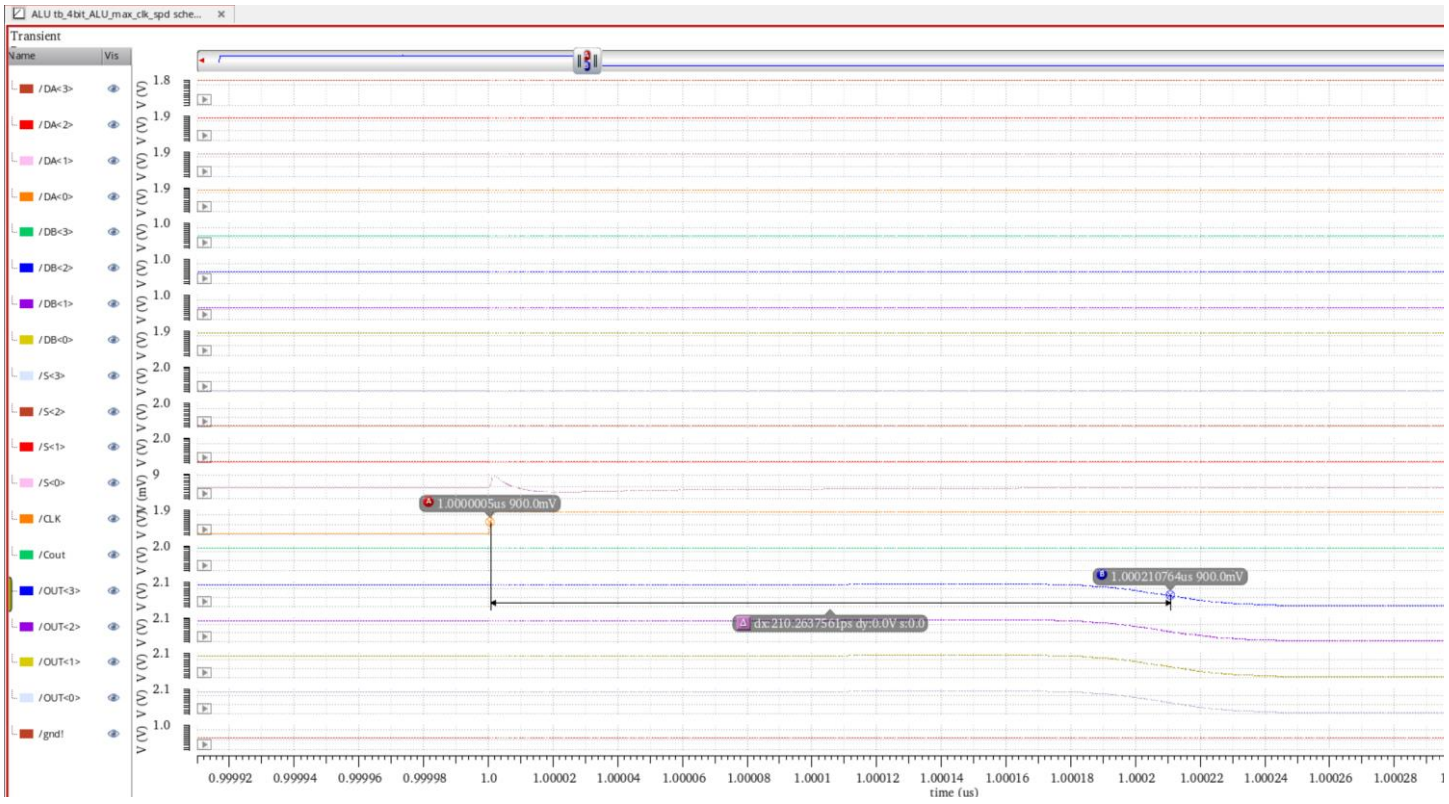
VDD = 1.7V, Propagation delay = 229.36ps



VDD = 1.8V, Propagation delay = 218.79ps



VDD = 1.9V, Propagation delay = 210.26ps



Results

VDD = 1.9V is the fastest (for worst-case computational path)

$$P_{\text{switching}} = \alpha C V_{DD}^2 f$$

The equation above shows what is needed to calculate the power when data goes through the input to the output on the positive clock edge.

$$\alpha = 1$$

For VDD = 1.7

$$(50fF)(1.7V_{DD})^2\left(\frac{1}{229.36ps}\right) = 0.63mW$$

For VDD = 1.8

$$(50fF)(1.8V_{DD})^2\left(\frac{1}{218.79ps}\right) = 0.73mW$$

For VDD = 1.9

$$(50fF)(1.9V_{DD})^2\left(\frac{1}{218.79ps}\right) = 0.86mW$$

Depending on our requirements, we can see that the higher VDD is, the more power the Adder will consume, therefore it is up to the designer to choose which VDD will fit the design constraints.

Requirement: With flip-flops at the input and output, we now have an accumulator. Using your architecture, implement and simulate the program A = A + 2 iteratively using your ALU.

Conclusion

In conclusion, the ALU design project has been successfully executed, resulting in the creation of a fully functional 4-bit Arithmetic Logic Unit (ALU). The project began with a comprehensive introduction outlining the objectives, design methodology, and testing requirements. The hierarchical design approach, starting from the block level down to the transistor level, facilitated a structured development process.

The ALU design incorporates various arithmetic and logical operations, including addition, subtraction, bitwise AND, OR, XOR, as well as input passing and setting outputs to high. Simulation-based testing was extensively performed to validate the functionality of the ALU, covering different instruction sets and scenarios.

Key components of the ALU, such as the ROM, decoder, and adder/subtractor, were thoroughly explained, providing insight into their operation and functionality. The ROM, in particular, was detailed to illustrate its role in decoding instructions and selecting operations.

Additionally, the project addressed additional requirements, including the implementation of flip-flops for sequential logic design, determination of worst-case propagation delay, simulation for maximum clock speed under different supply voltages, and iterative program execution for an accumulator.

Overall, the ALU design project demonstrated a solid understanding of CMOS logic design principles, combinational and sequential logic, microprocessor building blocks, and simulation-based testing techniques. The successful completion of the project affirms the efficacy of the design approach and highlights the achievement of the project objectives.

References

[1] GfG, “4-bit binary Adder-Subtractor,” GeeksforGeeks, Aug. 27, 2019. https://www.geeksforgeeks.org/4-bit-binary-adder-subtractor/ (accessed Mar. 21, 2024).