# Energy-Efficient VLSI Design ALU Lab Design Project

Nicholas Kim

# March 2024

# Contents

1	Design Objective	<b>2</b>
2	ALU Design	2
3	ALU Verification	4
4	Gate Design and Device Sizing           4.1 Inverter           4.2 NAND           4.3 NOR           4.4 XOR	<b>9</b> 10 11 14 16
5	4-bit Adder Design         5.1       Full Adder         5.1.1       Layout	<b>17</b> 18 19
6	Multi-Bit Gate Design           6.1         4-bit AND           6.2         4-bit OR           6.3         4-bit XOR	20 20 21 21
7	Multiplexer Design         7.1       4:1 Multiplexer	<b>22</b> 24
8	ROM Design	<b>25</b>
9	Challenges and Future Considerations	<b>25</b>

#### 1 Design Objective

The objective of the project is to design a 4-bit ALU in 0.18µm CMOS. The instruction set for the ALU is listed in Table 1.

INSTR<2:0>	OUT<3:0>	Description
000	= A + B	Add
001	= A - B	Subtract
010	= (A AND B)	Bitwise AND
011	= (A OR B)	Bitwise OR
100	= (A XOR B)	Bitwise XOR
101	А	Pass A
110	В	Pass B
111	= 1111	All HIGH

Table 1	1:	ALU	Instructions
---------	----	-----	--------------

This document covers each layer of hierarchy in the ALU design from CMOS devices to the top-level block. Section 2 covers the high-level architecture of the ALU. Section 3 shows the results of the ALU testbench verifying full functionality. Section 4 covers design, device sizing, and layout of all logic gates used in the ALU blocks. Section 5 covers the design of the 4-bit adder with subtraction capability. The full adder design, layout, and layout verification is included in this section. Section 6 covers the 4-bit gate designs. Section 7 covers the multiplexer design. Section 8 covers the ROM block design.

#### 2 ALU Design



Figure 1: ALU Symbol

The ALU high-level architecture is shown in Figure 3. There are 8 unique instructions required of the ALU design. For each instruction, there is a resulting 4-bit output based on the 4-bit inputs A and B. A multiplexer is used to select 1 of 8 channels based on the instruction, each channel containing the 4-bit output of its corresponding operation. The 3-bit instruction input is directly used for the 3-bit channel-select input on the multiplexer. The 4-bit output of the multiplexer is the output of the ALU block.

Instructions 000 and 001 select the output of the 4-bit adder block, maintaining equality between instruction and multiplexer select. Instruction 000 uses the 4-bit adder to perform addition on inputs A and B. The ROM block asserts the subtract and carry-in inputs of the 4-bit adder on Instruction 001 to perform subtraction. The OVERFLOW flag outputs HIGH on sums exceeding 4-bits. For subtraction, the OVERFLOW flag outputs HIGH on negative differences.

Instructions 010, 011, and 100 select the outputs of 4-bit gates performing AND, OR, and XOR bit-wise operations, respectively. The remaining instructions are implemented by directly passing A, B, and VDD (all high) to the 3 remaining channels.



Figure 2: ALU Schematic



Figure 3: ALU Annotated Schematic

# 3 ALU Verification

Figure 4 shows the testbench used for the ALU to verify functionality. Voltage sources assert/deassert the A and B input bits to simulate a variety of different input combinations. Each simulation tests each instruction using periodic voltage sources. The outputs are loaded with 1pF capacitors. The following plots and tables show the results of each simulation demonstrating correct functionality.



Figure 4: ALU Testbench



Figure 5: ALU Testbench Simulation 1

A<3:0>	B<3:0>	INSTR<2:0>	OUT<3:0>	OVERFLOW	Description
0000	0000	000	0000	0	0 + 0 = 0
0000	0000	001	0000	0	0 - 0 = 0
0000	0000	010	0000	0	0000  AND  0000 = 0000
0000	0000	011	0000	0	0000  OR  0000 = 0000
0000	0000	100	0000	0	0000  XOR  0000 = 0000
0000	0000	101	0000	0	Pass A
0000	0000	110	0000	0	Pass B
0000	0000	111	1111	0	All HIGH

Table 2: ALU Testbench Simulation 1 Results



Figure 6: ALU Testbench Simulation 2

A<3:0>	B<3:0>	INSTR<2:0>	OUT<3:0>	OVERFLOW	Description
1001	0011	000	1100	0	9 + 3 = 12
1001	0011	001	0110	0	9 - 3 = 6
1001	0011	010	0001	0	1001  AND  0011 = 0001
1001	0011	011	1011	0	1001  OR  0011 = 1011
1001	0011	100	1010	0	1001  XOR  0011 = 1010
1001	0011	101	1001	0	Pass A
1001	0011	110	0011	0	Pass B
1001	0011	111	1111	0	All HIGH

Table 3: ALU Testbench Simulation 2 Results



Figure 7: ALU Testbench Simulation 3

					-
A<3:0>	B<3:0>	INSTR<2:0>	OUT<3:0>	OVERFLOW	Description
0110	0111	000	1101	0	6 + 7 = 13
0110	0111	001	1111	1	6 - 7 = -1 (OVERFLOW)
0110	0111	010	0110	0	0110  AND  0111 = 0110
0110	0111	011	0111	0	0110  OR  0111 = 0111
0110	0111	100	0001	0	0110  OR  0111 = 0001
0110	0111	101	0110	0	Pass A
0110	0111	110	0111	0	Pass B
0110	0111	111	1111	0	All HIGH

Table 4: ALU Testbench Simulation 3 Results



Figure 8: ALU Testbench Simulation 4

A<3:0>	B<3:0>	INSTR<2:0>	OUT<3:0>	OVERFLOW	Description
1110	0101	000	0011	1	14 + 5 = 19 (OVERFLOW)
1110	0101	001	1001	0	14 - 5 = 9
1110	0101	010	0100	1	1110  AND  0101 = 0100
1110	0101	011	1111	1	1110  OR  0101 = 1111
1110	0101	100	1011	1	1110  XOR  0101 = 1110
1110	0101	101	1110	1	Pass A
1110	0101	110	0101	1	Pass B
1110	0101	111	1111	1	All HIGH

 Table 5: ALU Testbench Simulation 4 Results



Figure 9: ALU Testbench Simulation 5

A<3:0>	B<3:0>	INSTR<2:0>	OUT<3:0>	OVERFLOW	Description
1010	0101	000	1111	0	10 + 5 = 15
1010	0101	001	0101	0	10 - 5 = 5
1010	0101	010	0000	0	1010  AND  0101 = 0000
1010	0101	011	1111	0	1010  OR  0101 = 1111
1010	0101	100	1111	0	1010  XOR  0101 = 1111
1010	0101	101	1010	0	Pass A
1010	0101	110	0101	0	Pass B
1010	0101	111	1111	0	All HIGH

Table 6: ALU Testbench Simulation 5 Results



Figure 10: ALU Testbench Simulation 6

					-
A<3:0>	B<3:0>	INSTR<2:0>	OUT<3:0>	OVERFLOW	Description
0111	1000	000	1111	0	7 + 8 = 15
0111	1000	001	1111	1	7 - 8 = -1 (OVERFLOW)
0111	1000	010	0000	0	0111  AND  1000 = 0000
0111	1000	011	1111	0	0111  OR  1000 = 1111
0111	1000	100	1111	0	0111  XOR  1000 = 1111
0111	1000	101	0111	0	Pass A
0111	1000	110	1000	0	Pass B
0111	1000	111	1111	0	All HIGH

Table 7: ALU Testbench Simulation 6 Results



Figure 11: ALU Testbench Simulation 7

A<3:0>	B<3:0>	INSTR<2:0>	OUT<3:0>	OVERFLOW	Description
1111	1111	000	1110	1	15 + 15 = 30 (OVERFLOW)
1111	1111	001	1111	0	15 - 15 = 0
1111	1111	010	1111	1	1111  AND  1111 = 1111
1111	1111	011	1111	1	1111  OR  1111 = 1111
1111	1111	100	0000	1	1111  XOR  1111 = 0000
1111	1111	101	1111	1	Pass A
1111	1111	110	1111	1	Pass B
1111	1111	111	1111	1	All HIGH

Table 8: ALU Testbench Simulation 7 Results

# 4 Gate Design and Device Sizing

The logic gates used in the ALU design include:

- Inverter (NOT)
- NAND (2-input, 3-input, and 4-input)
- NOR
- XOR

A PMOS:NMOS width ratio of 2:1 is used in this design. The device widths in each logic gate are designed to match the drive strength of a  $2\mu$ m: $1\mu$ m inverter. All device lengths are 0.18 $\mu$ m.

#### 4.1 Inverter



Figure 12: Inverter Symbol



Figure 13: Inverter Schematic (PMOS 2 $\mu$ m, NMOS 1 $\mu$ m). Other gate devices are sized to match the equivalent drive strength of a 2 $\mu$ m:1 $\mu$ m inverter.



Figure 14: Inverter Layout

## 4.2 NAND



Figure 15: NAND2 Symbol



Figure 16: NAND2 Schematic (PMOS 2µm, NMOS 2µm). Parallel PMOS devices in pull-up network match drive strength at 2µm. Series NMOS devices in pull-down network must be 2µm each to match the equivalent output-to-ground resistance of a 1µm NMOS device in a 2µm:1µm inverter pull-down network.



Figure 17: NAND2 Layout



Figure 18: NAND3 Symbol



Figure 19: NAND3 Schematic (PMOS 2µm, NMOS 3µm). Parallel PMOS devices in pull-up network match drive strength at 2µm. Series NMOS devices in pull-down network must be 3µm each to match the equivalent output-to-ground resistance of a 1µm NMOS device in a 2µm:1µm inverter pull-down network.



Figure 20: NAND4 Symbol



Figure 21: NAND4 Schematic (PMOS 2µm, NMOS 4µm). Parallel PMOS devices in pull-up network match drive strength at 2µm. Series NMOS devices in pull-down network must be 4µm each to match the equivalent output-to-ground resistance of a 1µm NMOS device in a 2µm:1µm inverter pull-down network.

#### 4.3 NOR



Figure 22: NOR2 Symbol



Figure 23: NOR2 Schematic (PMOS 4µm, NMOS 1µm). Parallel NMOS devices in pull-down network match drive strength at 1µm. Series PMOS devices in pull-up network must be 4µm each to match the equivalent source-to-output resistance of a 2µm PMOS device in a 2µm:1µm inverter pull-up network.



Figure 24: NOR2 Layout



Figure 25: XOR Symbol



Figure 26: XOR Schematic



Figure 27: XOR Layout

# 5 4-bit Adder Design



Figure 28: 4-Bit Adder Symbol

The 4-bit adder block is used to perform addition or subtraction on two 4-bit inputs. It is implemented as a ripple-carry adder in which the carry-out of the least-significant full adder propagates to the carry-in of the next-most-significant. Two's complement subtraction is implemented by toggling all B input bits with XOR gates and relying on the upper-level design to assert the carry-in bit. The OVERFLOW flag described in Section 2 is implementing by outputting the carry-out bit. The carry-out bit is toggled in subtraction mode such that signed binary outputs are flagged.



Figure 29: 4-Bit Adder Schematic

#### 5.1 Full Adder



Figure 30: Full Adder Symbol

The full adder adds two one-bit inputs with a carry-in and carry-out. The design for the full adder is derived from the sum and carry-out as logical functions of bits A, B and carry-in.

$$S = A \oplus B \oplus CIN$$

 $COUT = A \cdot B + CIN \cdot (A \oplus B)$ 



Figure 31: Full Adder Schematic

#### 5.1.1 Layout



Figure 32: Full Adder Layout

The full adder layout utilizes the gate layouts shown in Figures 14, 17, 24, and 27. Figures 33 and 34 show the Calibre DRC and LVS passing results.

Cali	bre Interactive - nmDRC v2022.2_38.20 : /nfs/stak/users/kimnice/run_drc/drc_runset *	Calibre - RVE v2022.2_38.20 : fullAdder.drc.results ×
<u>File</u> <u>Settings</u> <u>Co</u>	nfigurations Help Search	≤ Eile ⊻iew Highlight Tools Window Setup Help
Rules	fullAdder.drc.summary X	
Outputs		Search
Run Control	3	= Tritter: Show Not Waived T ⊞No Results Found
Search	4 === CALIBRE::DRC-F SUMMARY REPORT	🖾 🍕 Check / Cell   Results
Transcript		
Files	7 Calibre Version: v2022.2 38.20 Thu Jun 2 16:14:33 PDT 2022	
	8 Rule File Pathname: calibre.drc	
	9 Rule File Title:	
	10 Layout System: GDS	
	11 Layout Path(s): fullAdder.calibre.db	
	12 Layout Primary Cell: fullAdder	52
Run DRC	Current Directory: /nfs/stak/users/kimnich/eevlsi/cadence/run_drc	×
	User Name: kimnich	
Show RVE	15 Maximum Results/RuleCheck: 1000	
	10 Maximum Result Vertices: 4096	
	The Result's Database: InitAdder.drc.result's (ASCII)	
	10 Day Depth. All	
	20 Summary Report File: fullAdder.drc.summary (REPLACE)	
	Geometry Flagging: ACUTE = YES SKEW = YES ANGLED = NO OFFGRID = YES	
	22 NONSIMPLE POLYGON = YES NONSIMPLE PATH = YES	
	23 Excluded Cells:	Calibre Run Completed Successfully Results are Valid
		0

Figure 33: Full Adder Layout Design Rules Checking (DRC)

		- Dearent
Rules	fullAdder.lvs.report X	
Inputs	13	
Outputs	14 REPORT FILE NAME:	fullAdder.lvs.report
Ontions	15 LAYOUT NAME:	<pre>svdb/fullAdder.sp ('fullAdder')</pre>
space	16 SOURCE NAME:	fullAdder.src.net ('fullAdder')
ERC	17 RULE FILE:	Callbre.lvs Fri Mar 22 12:09:24 2024
Signatures	19 CURRENT DIRECTORY:	/nfs/stak/users/kimnich/eevlsi/cadence/
Run Control	run_lvs	
Search	20 USER NAME:	kimnich
Transcript	21 CALIBRE VERSION:	v2022.2_38.20 Thu Jun 2 16:14:33 PDT
Filos	2022	
Files	23	
Run LVS	24	
	25	OVERALL COMPARISON RESULTS
Show RVE	26	
	28	
	29	
	30	* * * * *
	31 *	# # CORRECT #
	32 # #	* * \_/

Figure 34: Full Adder Layout Versus Schematic (LVS)

# 6 Multi-Bit Gate Design



Figure 35: Multi-Gate Symbol

The multi-bit gate blocks perform bit-wise operations on two 4-bit inputs. Figure 35 shows the common symbol used by all multi-bit gates. The design for each multi-bit gate involves logic for each bit pair between the two inputs.

#### 6.1 4-bit AND

An AND operation is performed on 2 bits by inverting the output of a NAND2 gate. The 4-bit AND gate block implements 4 AND functions on each bit pair.



Figure 36: 4-Bit AND Schematic

#### 6.2 4-bit OR

An OR operation is performed on 2 bits by inverting the output of a NOR2 gate. The 4-bit OR gate block implements 4 OR functions on each bit pair.



Figure 37: 4-Bit OR Schematic

#### 6.3 4-bit XOR

The 4-bit XOR gate block implements 4 XOR functions on each bit pair.



Figure 38: 4-Bit XOR Schematic

# 7 Multiplexer Design



Figure 39: 32:4 Multiplexer Symbol

The 32:4 mux (multiplexer) outputs one 4-bit channel out of 8 4-bit input channels based on a 3-bit channel-select. It consists of 4 identical 8:1 muxes. Each 8:1 mux takes in one bit from each channel. All 8:1 muxes share the same channel-select such that all four bits of the selected channel are sent to the output.



Figure 40: 32:4 Multiplexer Schematic

The 8:1 mux is implemented as two 4:1 muxes, the two outputs of which being selected by a third select bit.



Figure 41: 8:1 Multiplexer Section

#### 7.1 4:1 Multiplexer



Figure 42: 4:1 Multiplexer Symbol

The 4:1 mux outputs 1 bit out of 4 input bits based on 2 select bits. It is implemented using 4 NAND3 gates, each taking one combination of the four select configurations (00, 01, 10, 11). The three NAND3 gates with input selects that are not true outputs HIGH. The gate with the true select passes and inverts its corresponding bit input, after which it is inverted again by the NAND4 gate at the block output.



Figure 43: 4:1 Multiplexer Schematic

### 8 ROM Design



Figure 44: ROM Symbol

As described in Section 2, the only function of the ROM block is to assert the subtraction flag for the 4-bit adder on Instruction 001; the remaining instructions directly control the 32:4 multiplexer. The subtraction flag is asserted when neither of the two most-significant instruction bits are HIGH and the least significant instruction bit is HIGH.



Figure 45: ROM Schematic

#### 9 Challenges and Future Considerations

The most significant challenge in this project was passing LVS for the full adder layout. The error was indicating 4 net discrepancies on the NAND gate inputs that suggested that the two XOR gates needed to be swapped. However, selecting each XOR in the schematic viewer suggested that the XOR gates and NAND gates were connected correctly with respect to the net discrepancies. The actual error was due to the outputs of the NAND gates needing to be swapped. The LVS error was misleading as it indicated errors on the NAND inputs as opposed to the NAND outputs. LVS recognized the order of the NAND gates adhering to their respective output nets, whereas the schematic viewer and highlighting tools recognized the order of the NAND gates adhering to their respective input nets.

For future designs, these are a few considerations regarding areas for improvement:

- Standard layout practices should be implemented to make compact and efficient gates.
- 4-bit adder may be implemented using a more efficient topology.
- 32:4 multiplexer may be reduced to a 28:4 multiplexer since the 4-bit adder input covers two instructions. However, this would disrupt the instruction-to-select continuity, so further design would be needed to determine the most efficient method.