

Abstract

In this poster, the design, simulation, and physical layout implementation of a full-custom 8-bit Arithmetic Logic Unit (ALU) is presented. The ALU is designed to operate for 6 modes of functions. Different logic families including static CMOS, transmission-gate based design, MUX-based design are incorporated in this custom design to improve its performance. Carry-lookahead adder (CLA) is used for adder implementation for fast computations. One 8x1 multiplexer (MUX) was designed using 7 2x1 multiplexers and each of these 2x1 MUX were designed using a transmission gate logic. Each instruction was successfully tested by using a custom test circuit with various input patterns. The complete ALU was carried out using the recently open-sourced GlobalFoundries 180 nanometer 7-track standard cell library. The 8-bit ALU is designed, and it is verified to perform the correct functions as designed.

Keywords: Arithmetic Logic Unit (ALU), Custom CMOS Design, Layout Design, PSPICE Simulation, Open-Source Hardware.

Introduction

The last few decades since the invention of modern-age computers, scientists and engineers have sought to increase the processing power of the general central processing units (CPUs) at the cost of an increased power-usage. However, the recent trend of acquiring an optimal performance out of a CPU at a minimal power usage has taken a great leap. In the field of deep-learning where the past assumptions that increased multi-bit capacity to record the results to provide an optimal performance out of the system has been shown to be false. Many researchers have shown that a reduced bit-usage can, in fact, allow a system to get as much as 90% of the original performance, which, in many cases, is more than sufficient to produce optimal result. Therefore, we present the design and simulation of an 8-bit ALU that can be potentially used in a CPU for performing quantized operations.

Design Considerations

A full-custom implementation of ALU was done and four-bit opcodes were used to implement seven arithmetic and logical instructions: NOT, NOP, AND, OR, ADD, SUB, XOR. These instructions have been designed using a combination of static CMOS and transmission-gate families [1]. Each opcode operates on an input of 8-bit operands and the final output is generated using 9-bit output, including the carry bit when ADD and SUB are done. The implementation of NOT, NOP, AND, OR, and XOR used straightforward CMOS design process, and this can be seen in (1) to (5).

$$INV_i = \sim A_i \quad (1)$$

$$NOP = 0 \quad (2)$$

$$AND_i = A_i \cdot B_i \quad (3)$$

$$OR_i = A_i + B_i \quad (4)$$

$$XOR_i = A_i \oplus B_i \quad (5)$$

ADD and SUB were implemented using CLA (Carry Look-Ahead) technique. Equations (6) and (7), were used for the 1-bit propagation (P) and generate (G) terms.

$$G_i = A_i \cdot B_i \quad (6)$$

$$P_i = A_i \cdot \bar{B}_i + \bar{A}_i \cdot B_i = A_i \oplus B_i \quad (7)$$

Then, the 1-bit sum (S), subtraction (SB) and carry (C) bits were generated using the (8) to (10), where A and B are 1-bit inputs.

$$S_i = A_i \oplus B_i \oplus C_{i-1} \quad (8)$$

$$SB_i = \sim A_i \oplus B_i \oplus C_{i-1}, \{C_{i-1} = 1 \text{ for } i = 0\} \quad (9)$$

$$C_i = G_i + P_i \cdot C_{i-1} \quad (10)$$

As the complexity of the design grows higher, it becomes increasingly difficult to implement a flat CMOS design for a complete system, therefore, a hierarchical design method was followed, where all the core gates were implemented as the root design, and all derived structures were based on the root. For example, figure 1 shows a top-level module and nested structure of ADD instruction.

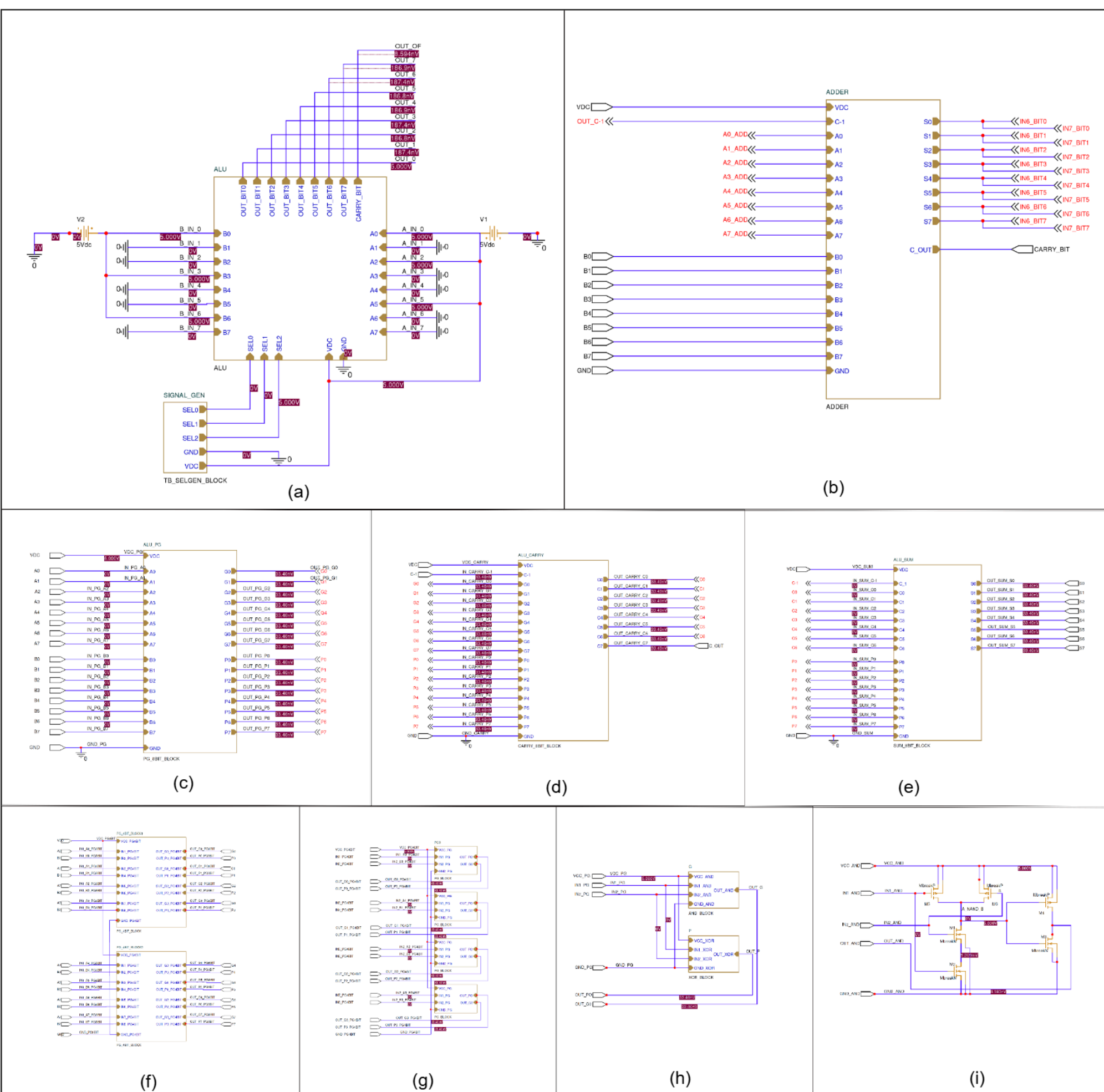


Figure 1. (a) ALU testbench (b) 8-bit ADD instruction top module (c) 8-bit Propagate-Generate (PG) module (d) 8-bit Carry generate module (e) 8-bit Sum (ADD) module (f) Two 4-bit PG modules (g) Four 2-bit PG modules (h) 1-bit XOR/AND blocks (i) CMOS 1-bit AND gate

The individual output bit from each instruction computation is then fed into an 8x1 MUX that uses a 3-bit control lines to select one instruction output. A total of 8, 8x1 MUX are used to generate a total of 9 output bits. The 2x1 MUX is designed using a transmission-gate logic and the structure for the a 2x1 MUX and the complete 8x1 MUX is shown in Figure 2.

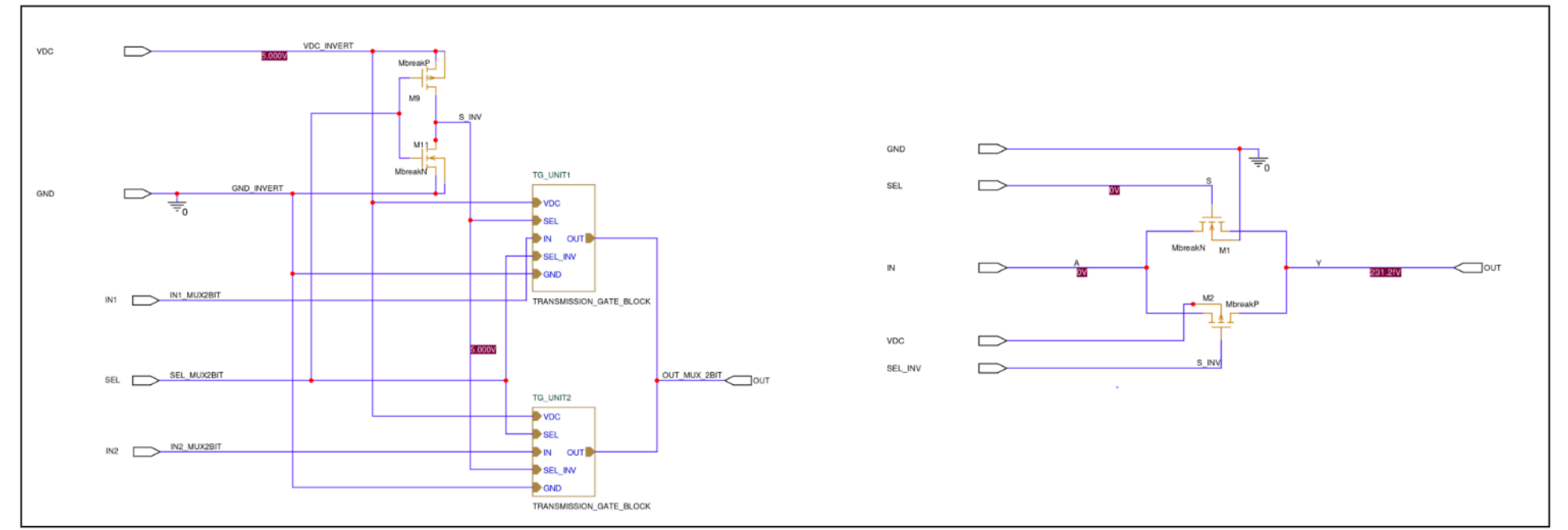


Fig. 2. Transmission-gate based MUX design

For 1-bit output from each instruction block, the output is fed into the MUX as shown in the block diagram below.

Layout Design

We used the GlobalFoundries 180nm Process Design Kit (PDK) to create the complete layout for the ALU. The PDK provides a set of standard cell libraries (7-track and 9-track), DRC and LVS decks to design and verify the functionality implemented. As shown in Figure 3, we used a hierarchical implementation methodology in KLayout as it provided an easier route to layout implementation. The complete design was completed in a double strip.

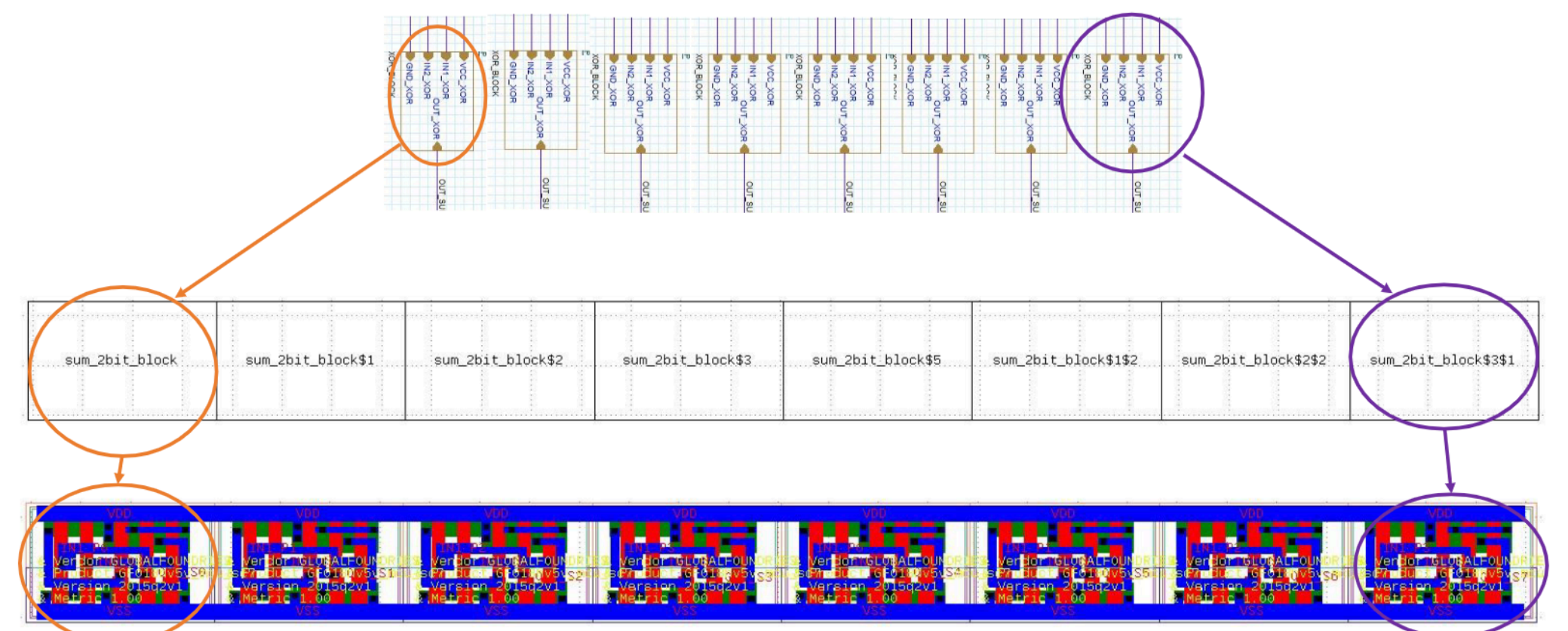


Fig. 3. Physical layout implementation of the SUM block

Results and Discussions

We simulated the design using SPICE. For pre-layout simulation, we used Cadence OrCAD Capture and PSPICE, and for post-layout simulation, we extracted the netlist from the layout using Magic. Figure 4 provides the execution waveform of four instructions after the layout were extracted.

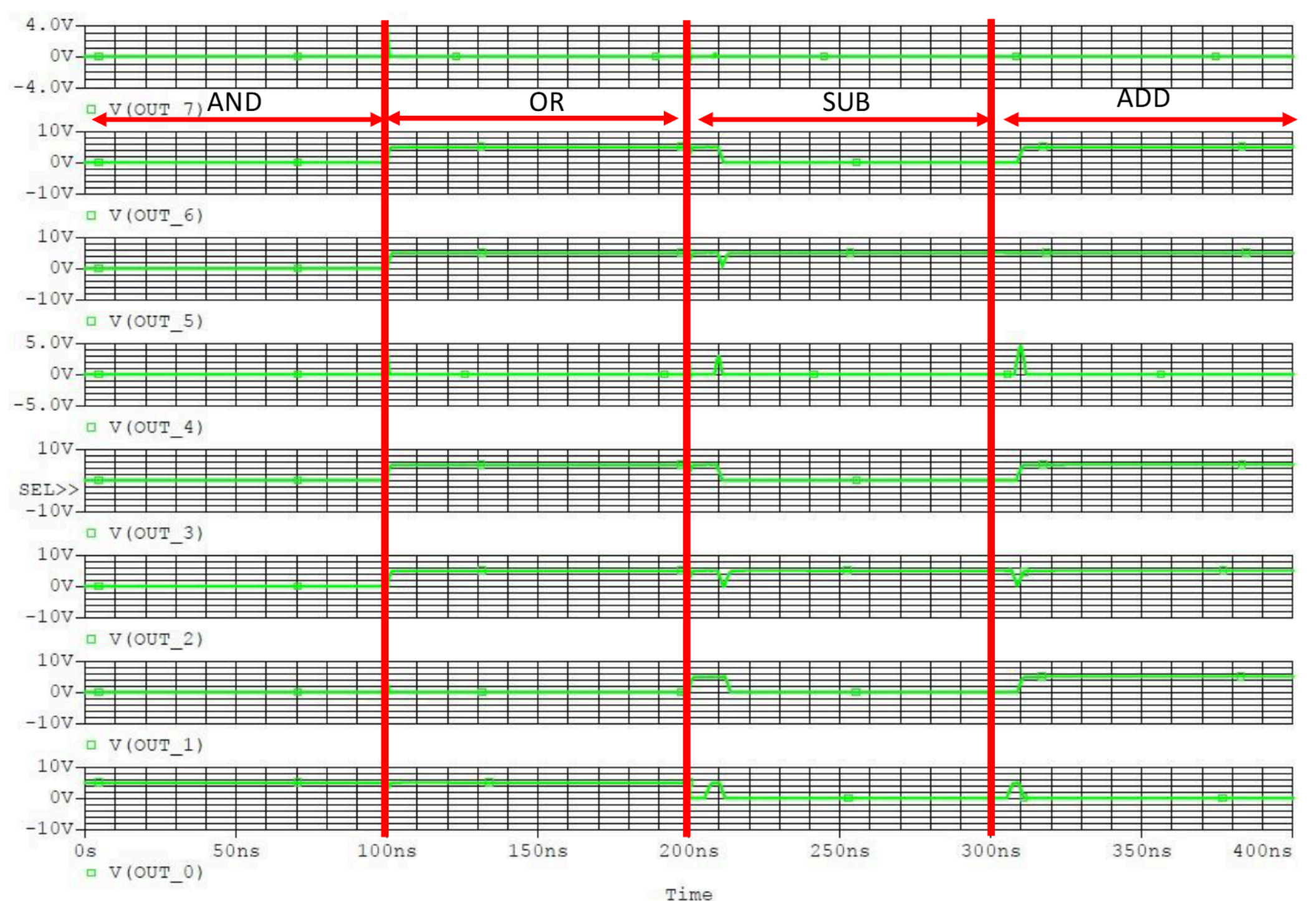


Fig. 4. Post-layout simulation of the extracted netlist

Comparing the average and the worst-case delays, we get the following results and we also observed that the worst-case delay was obtained in the subtraction instruction.

Table 1. Comparison of pre-layout and post-layout simulation results

Timing	Pre-layout	Post-layout	Difference
Average Delay	5.14ns	7.3ns	42%
Worst Case Delay	9.7ns	15.1ns	55%

Conclusions and Future Work

In this work, we presented the design, analysis and physical implementation and verification of an 8-bit ALU. The ALU is designed to perform 7 functions with 3 select lines. Pre-layout and post-layout simulations were done and correct working of all the functions were observed with a difference of 42% in the average delay and 55% worst-case delay.

References

[1] N. Weste and D. Harris, "CMOS VLSI Design: A Circuits and systems perspective". New Jersey: Addison-Wesley Educational Publishers Inc, 2010.