Design of a 2-Bit Carry Ripple Adder Using Half Adders and Full Adders

Ziya Kadir TOKLUOGLU
Introduction of Digital Integrated Circuit

November 22, 2024

Contents

1 Introduction

In digital electronics, arithmetic operations such as addition are fundamental. A Carry Ripple Adder (CRA) is a simple and widely used circuit for adding binary numbers. This report details the design of a 2-bit Carry Ripple Adder using Half Adders and Full Adders, implemented solely with NAND, NOR, and Inverter gates to ensure gate efficiency.

2 Logic Formula Explanation

2.1 Exclusive-OR (XOR) Formula

The Exclusive-OR (XOR) operation is essential for calculating the sum in both Half Adders and Full Adders. The XOR function can be implemented using only NAND, NOR, and Inverter gates as follows:

$$Y = ((A \cdot B)'' + (A + B)')' \tag{1}$$

Explanation: This formulation of XOR is chosen for its efficiency, requiring only 2 NOR gates, 1 NAND gate, and 1 Inverter, totaling 4 gates. This minimizes the gate count compared to alternative implementations, thereby enhancing the overall efficiency of the adder circuit.

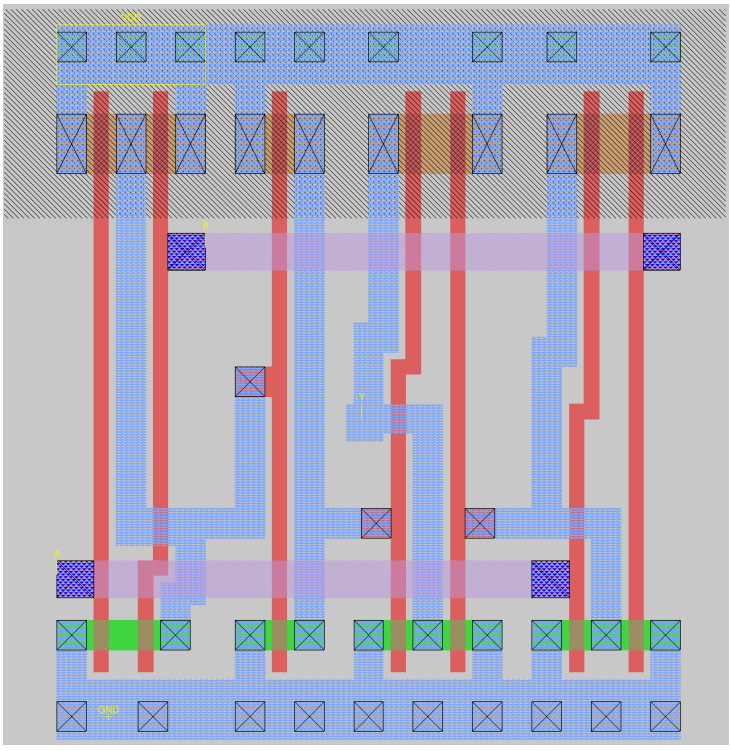

Figure 1: Layout Design of the XOR Gate

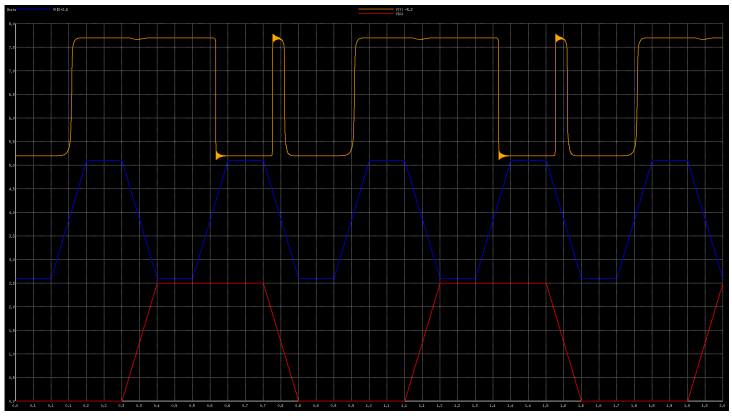

Figure 2: SPICE Simulation of the XOR Gate

2.2 Half Adder Formulas

A Half Adder is a fundamental building block for constructing a Carry Ripple Adder. It adds two single-bit binary numbers and produces a sum and a carry output.

2.2.1 Sum Calculation

The sum output of the Half Adder is calculated using the XOR formula derived above:

$$\text{Sum} = Y = ((A \cdot B)'' + (A + B)')' \tag{2}$$

Explanation: By utilizing the previously defined XOR implementation, the sum can be efficiently calculated using only 4 gates (2 NOR, 1 NAND, and 1 Inverter).

2.2.2 Carry Calculation

The carry output is straightforward:

$$\text{Carry} = A \cdot B \tag{3}$$

Explanation: The carry is directly obtained by the AND operation of inputs A and B. Since the AND operation is already implicitly present in the sum formula's layout, no additional gates are required for the carry output, further enhancing the efficiency of the Half Adder design.

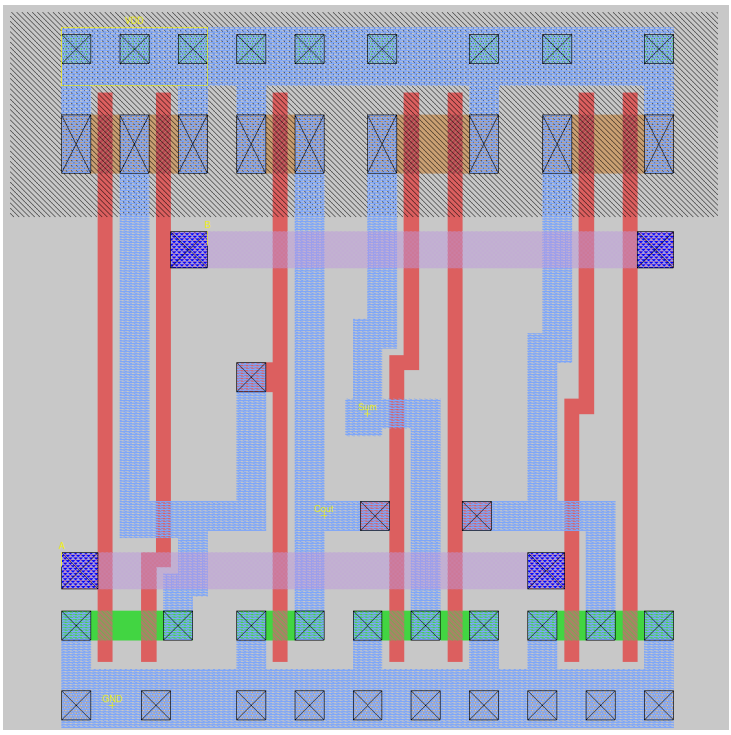

Figure 3: XOR Implementation in Half Adder Layout

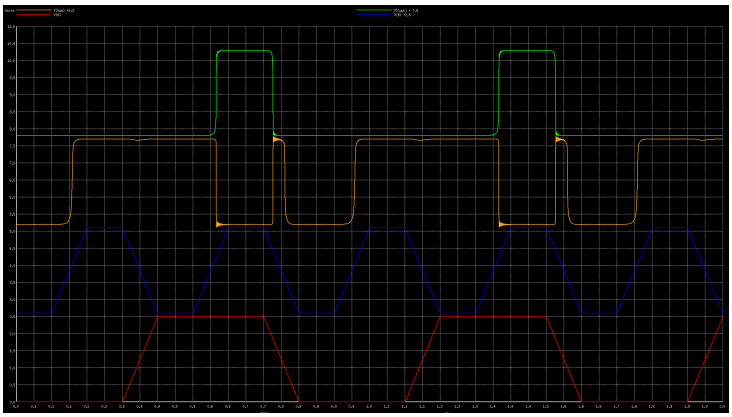

Figure 4: SPICE Simulation of Half Adder XOR

2.3 Full Adder Formulas

A Full Adder extends the functionality of a Half Adder by including an additional input for carry-in, enabling the addition of three binary digits. The Full Adder provides both a sum and a carry-out, which are essential for constructing multi-bit adders like the Carry Ripple Adder.

2.3.1 Sum Calculation

Let us define X as the XOR operation between inputs A and B:

$$X = ((A \cdot B)'' + (A + B)')' \tag{4}$$

Using the previously defined XOR implementation, the sum output S of the Full Adder is calculated as:

$$\text{Sum} = ((X \cdot C)'' + (X + C)')' \tag{5}$$

Explanation: By reusing the XOR implementation for A XOR B (denoted as X), the sum S is efficiently computed using an additional 4 gates (2 NOR, 1 NAND, and 1 Inverter) for the XOR operation between X and carry-in C. This approach leverages the existing XOR gate design, thereby minimizing the total gate count.

2.3.2 Carry Calculation

The carry-out C_{out} is determined using the following formula:

$$\text{Carry} = ((A \cdot B)' \cdot ((A \text{ XOR } B)' \cdot C))' \tag{6}$$

Explanation: The carry-out is derived by combining the AND operation of A and B with the AND operation of the inverted XOR result and carry-in C. This implementation requires an additional NOR gate specifically for generating the carry bit, resulting in a total of 9 gates for the Full Adder (4 gates for A XOR B, 4 gates for X XOR C, and 1 NOR gate for the carry bit). This design ensures gate efficiency by reusing existing XOR structures without redundant gate usage.

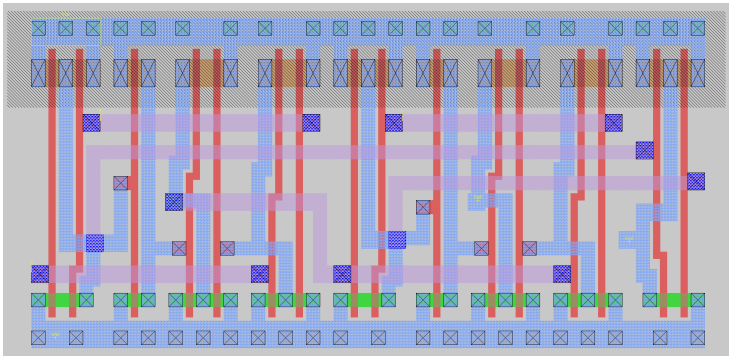

Figure 5: Layout Design of the Full Adder

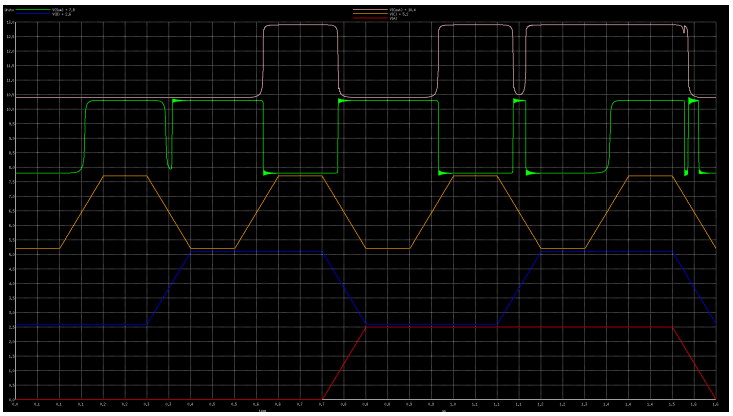

Figure 6: SPICE Simulation of the Full Adder

2.4 Carry Ripple Adder

The Carry Ripple Adder (CRA) combines multiple Half Adders and Full Adders to perform multi-bit binary addition. In this design, a 2-bit CRA is implemented using one Half Adder and one Full Adder, enabling the addition of two 2-bit numbers along with an initial carry-in.

The following truth table illustrates the functionality of the 2-bit Carry Ripple Adder. It shows the binary inputs a_1, a_0, b_1, b_0, the resulting sum outputs $(\text{Sum}_1, \text{Sum}_0)$, and the final carry (Carry):

Table 1: Truth Table for the 2-Bit Carry Ripple Adder

a_1	a_0	b_1	b_0	Sum_1	Sum_0	Carry
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	1	0	0
0	0	1	1	1	1	0
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	1	0
1	0	1	0	0	0	1
1	0	1	1	0	1	1
1	1	0	0	1	1	0
1	1	0	1	0	0	1
1	1	1	0	0	1	1
1	1	1	1	1	0	1

2.4.1 Design Implementation

The 2-bit Carry Ripple Adder is constructed as follows:

- Least Significant Bit (LSB): Uses a Half Adder to add the two LSBs of the input numbers.
- Most Significant Bit (MSB): Utilizes a Full Adder to add the two MSBs along with the carry-out from the Half Adder.

Gate Count Analysis:

- Half Adder: Requires 4 gates (as previously designed).
- Full Adder: Requires 9 gates.
- Total Gates: 4 (Half Adder) + 9 (Full Adder) = 13 gates

Explanation: The design leverages the efficiency of the Half Adder and Full Adder by reusing the XOR implementations, thereby minimizing the total number of gates required. This modular approach not only simplifies the design but also enhances scalability for adding more bits if needed.

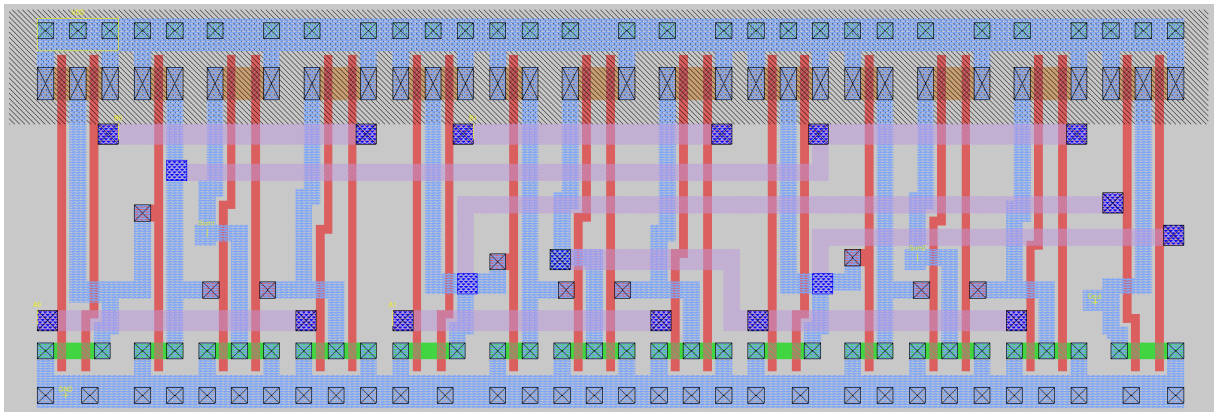

Figure 7: Layout Design of the 2-Bit Carry Ripple Adder

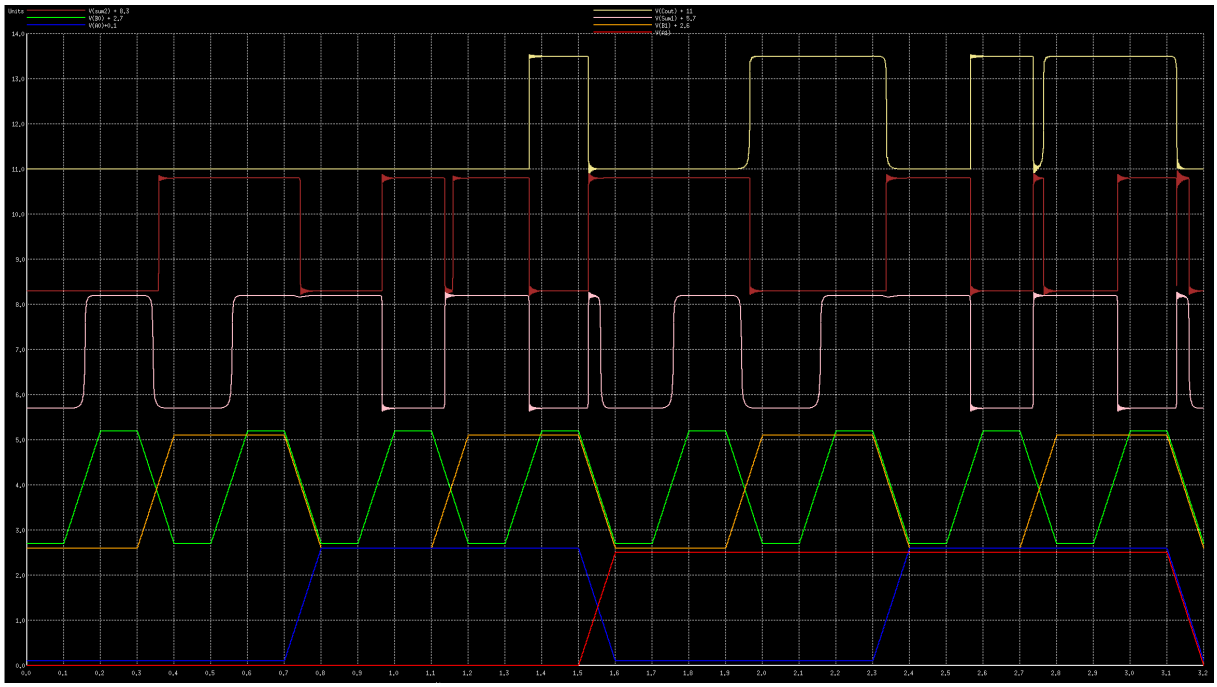

Figure 8: SPICE Simulation of the 2-Bit Carry Ripple Adder

3 Conclusion

In this report, we have successfully designed a 2-bit Carry Ripple Adder using Half Adders and Full Adders implemented exclusively with NAND, NOR, and Inverter gates. The chosen logic formulas for XOR and carry operations prioritize gate efficiency, minimizing the total gate count to 13 gates for the complete adder. The layout designs and SPICE simulations validate the functional correctness and efficiency of the proposed design. This modular approach lays a strong foundation for scaling up to larger bit-width adders, maintaining efficiency and reliability in digital arithmetic operations.