# A Four-bit Adder and Gates at the Transistor Level

Abigail Fry
Anusha Datar
Vienna Scheyer

May 9, 2019

## 1 Objective

Our goal for this project was to create a four-bit ripple carry adder in LTSpice. To create this device, we created three individual logic gates (AND, OR, and XOR), a one-bit full adder, and a four-bit adder.

## 2 Logic Gates

### 2.1 AND Gate

We used CMOS components to construct an AND gate.

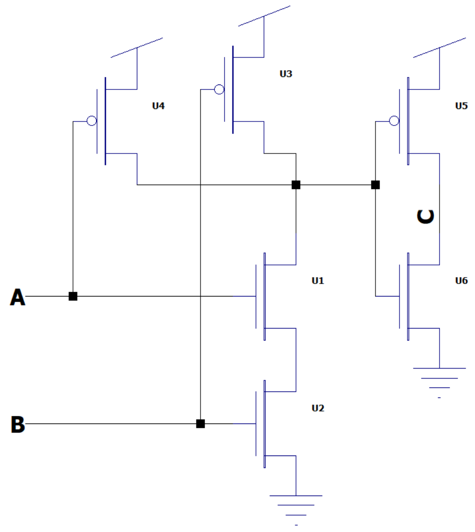

**Figure 1:** AND gate transistor-level schematic.

To make an AND gate, we connected both a NAND gate and and an inverter. The NAND gate, made up of the four leftmost transistors in Figure 1, will return a low output when both inputs are high (because the two NMOS transistors will conduct and the two PMOS transistors will not conduct, so the overall output between them will be low) and return a high output in any other case (because at least one NMOS transistor will not conduct and at least one PMOS transistor will conduct, yielding a high output). Then, we connect the output of the NAND gate to an inverter (made up of the two rightmost transistors) - if the input to the inverter is low, then the PMOS transistor will be high and NMOS transistor will be low, resulting in a high output in between them. Conversely, if the output of the NAND gate is high, the NMOS transistor in the inverter will be high and the PMOS will be low, yielding a low output overall.

Combining the NAND gate and the inverter yields a device that outputs low if any of the inputs are low and outputs high if both inputs are high, which meets the expected functionality of an AND gate.

We constructed the AND gate in LTspice and exhaustively tested it using the following test cases.

| Input | | Expected Output |
|---|---|---|
| A | B | Output |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**Table 1:** Test cases performed on AND gate.

Our AND gate passed all of these tests, as illustrated in the plot below. The output is high if and only if both inputs are high, and otherwise the output is low.

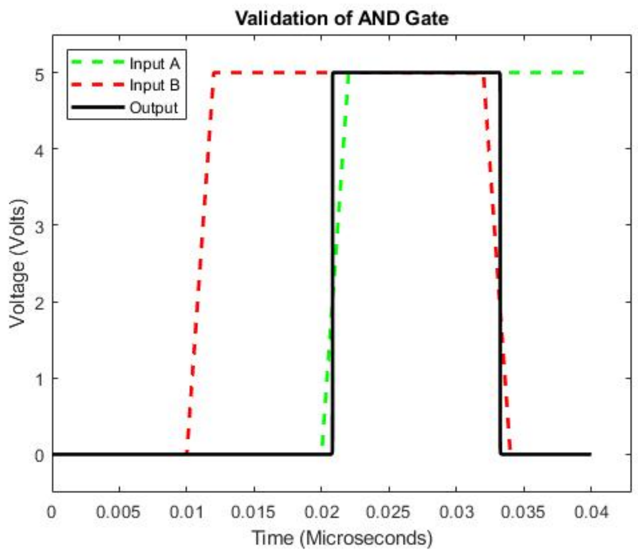

**Figure 2:** AND gate validation plot

## 2.2 OR Gate

We used CMOS components to construct an OR gate as shown in Figure 3.

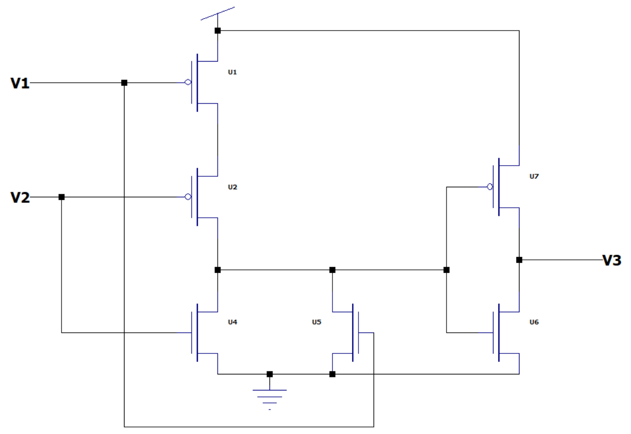

**Figure 3:** OR gate transistor-level schematic.

At the transistor level, the circuit consists of a few pieces that we can analyze: U1 and U2 make a PMOS NOR gate, U4 and U5 make an NMOS current mirror, and U6 and U7 make an inverter to turn the NOR gate output into OR gate logic.

In the PMOS NOR gate, when the gates of U1 and U2 are both held low, both PMOS transistors are on and current is allowed to flow. This makes the output of the NOR gate high. In the three other cases (U1 high and U2 low, U1 low and U2 high, or both gates high), at least one transistor is off which obstructs the flow of current and causes the output of the NOR gate to be low.

When U1 and U2 are both low, the current mirror NMOS transistors are off. The current that flows through the NOR gate flows into the gates of U6 and U7. The inverter NMOS is on and the inverter PMOS is off, so Vout is low.

When one of the inputs or both of the inputs is high, there is no current in the current mirror and the gates of the inverter transistors are pulled low. This means the inverter PMOS is on and the inverter NMOS is off, so the output is high in these three cases.

We constructed the OR gate in LTspice and exhaustively tested it using the following test cases.

| Input | | Expected Output |
|---|---|---|
| A | B | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**Table 2:** Test cases performed on OR Gate.

Our OR gate passed all of these tests, as illustrated in the plot below. The output is low if and only if both inputs are low, and otherwise the output is high.

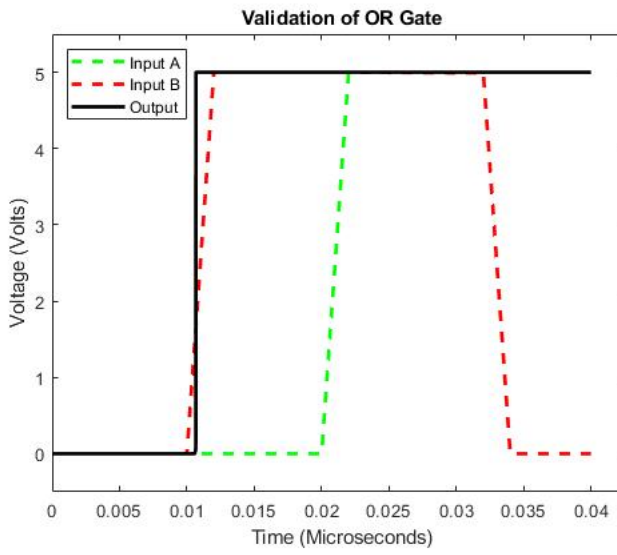

**Figure 4:** OR gate validation plot.

## 2.3 XOR Gate

We used CMOS components to construct an XOR gate.

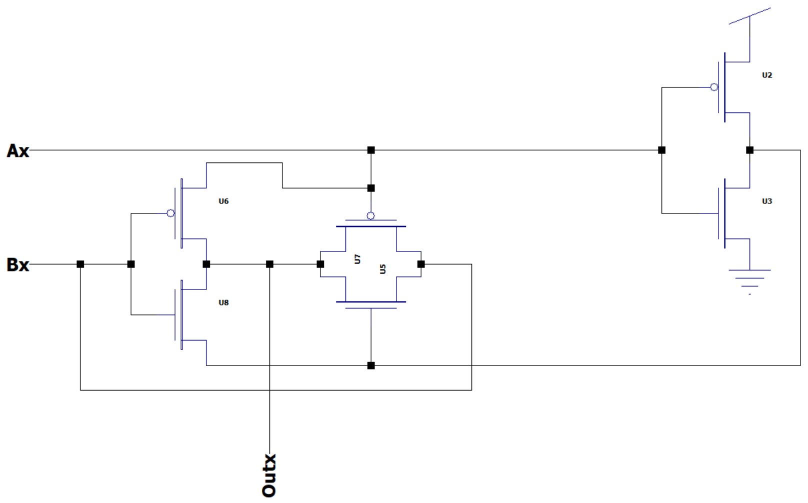

**Figure 5:** XOR gate transistor-level schematic.

The XOR gate is made up of several parts: two inverters and an analog switch. The inverted and non-inverted forms of the Ax input are wired to the gates of the NMOS and PMOS analog switch transistors, respectively. The inverted and non-inverted forms of the Bx input are wired to the output of the whole circuit and the input to the analog switch, respectively.

When both Ax and Bx are low, the NMOS in the analog switch is on, and the PMOS in the switch is off. The switch passes the Bx input and drives the output low.

When Ax is high and Bx is low, the NMOS and PMOS transistors in the analog switch are off. The Ax input powers the Bx inverter, which drives the output high.

When Ax is low and Bx is high, the NMOS and PMOS transistors in the analog switch are on. The Bx inverter is not powered, and the analog switch passes the Bx input to the output so the output is high.

When both Ax and Bx are high, the NMOS and PMOS transistors in the analog switch are off. The Bx inverter is powered and Bx gets inverted, so the output is low.

We constructed the XOR gate in LTSpice and exhaustively tested it using the following test cases.

| Input | | Expected Output |
|---|---|---|
| A | B | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**Table 3:** Test cases beformed on XOR gate.

Our XOR gate passed all of these tests, as illustrated on the plot below. The output is high if and only if both inputs are distinct, and otherwise the output is low.

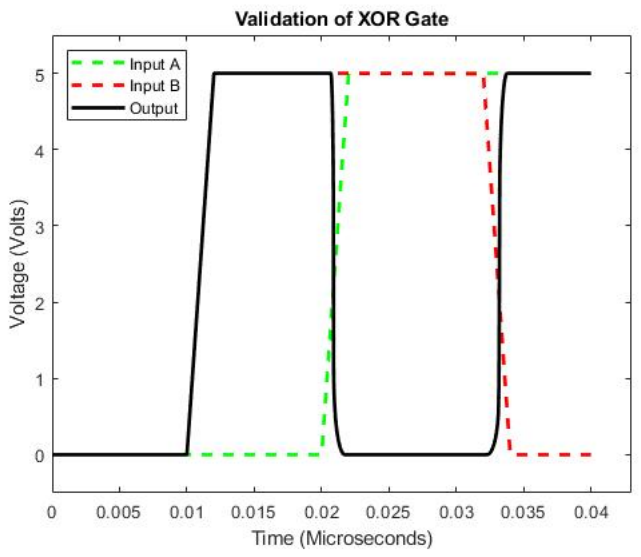

**Figure 6:** XOR gate validation plot

## 3 One-Bit Adder

Using combinations of the logic gates we developed, we created a one-bit adder.

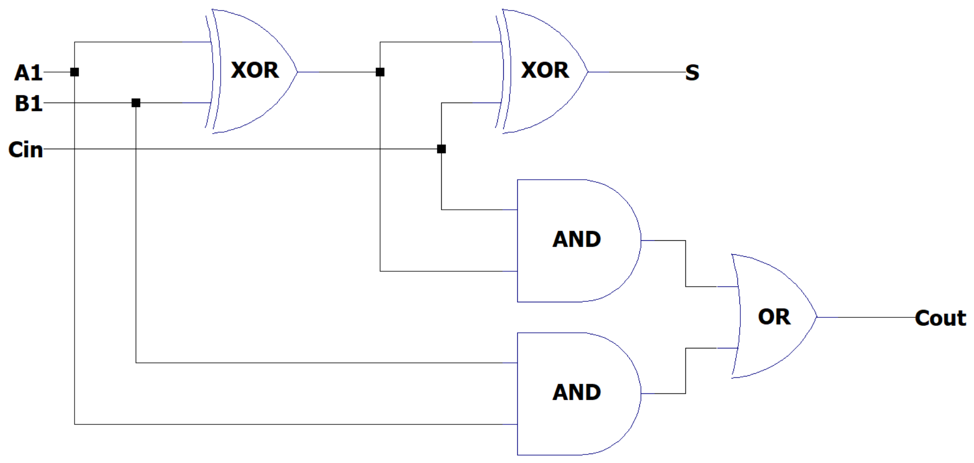

**Figure 7:** One-bit adder gate-level schematic.

To calculate the sum bit, we use two cascaded XOR Gates. The first XOR gate determines the initial sum bit of the output based only on the inputs: if the inputs are both 0 or both 1, the initial sum will be 0, and if one is 0 and the other one is 1, the initial sum will be 1 (their sum). Then, if the carryin is equal to the initial sum, the output will be 0 (either reaffirming or adding to the original sum), and if the carryin is not equal to other original sum, it will effectively add 1 to the output.

To calculate the carryout, we AND each input together and we AND both the ouput and the carryin and then OR the results from those operations. This way, we can identify whether or not there is an overflow on the initial sum or after factoring in the carryin bit.

After constructing the adder using hierarchical schematics and our original logic gates, we exhaustively tested it to ensure it functioned in all possible cases.

| Input | | | Expected Output | |
|---|---|---|---|---|
| A | B | Carryin | Sum | Carryout |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Table 4:** Test cases performed on full one-bit adder.

Our one-bit adder cell works with each possible set of inputs, as illustrated on the plot below.

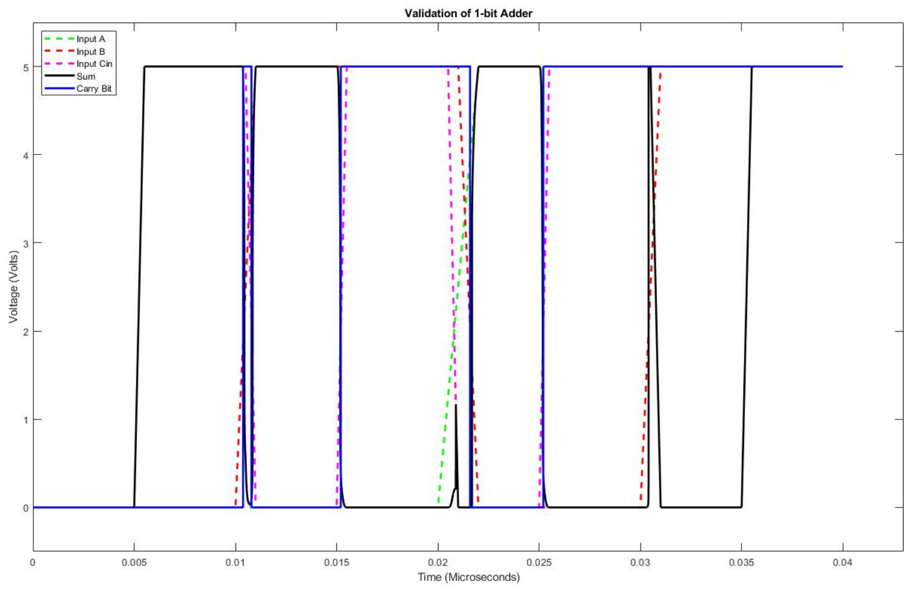

**Figure 8:** One-bit adder validation plot

## 4 Four-Bit Adder

By connecting four of the one-bit adders that we developed, we created a four-bit adder.

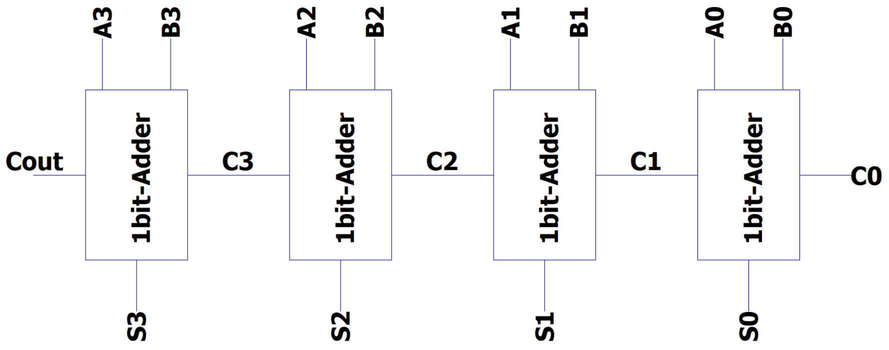

**Figure 9:** Four-bit ripple carry adder adder-level schematic

To create the four-bit adder, we created a new one-bit adder for each bit of the binary representation of the two inputs. We then connected the carryout of each individual adder to the carryin of the adder for the following bit. This leaves a single input carryin and a single output carryout value associated with the final adder and a sum bit at each adder cell.

After assembling the four-bit adder in LTSpice, we created a test suite with the following cases to validate the functionality of carryin, carryout, and addition on each bit. By choosing these tests, we ensured we had high test coverage without having to exhaustively test all two hundred fifty-six possible cases.

| Input | | | Expected Output | |
|---|---|---|---|---|
| A | B | Carryin | Sum | Carryout |
| 0000 | 0000 | 0 | 0000 | 0 |
| 0000 | 0000 | 1 | 0001 | 0 |
| 1111 | 1111 | 0 | 1110 | 1 |
| 1111 | 1111 | 1 | 1111 | 1 |
| 1111 | 0000 | 0 | 1111 | 0 |
| 1111 | 0000 | 1 | 0000 | 1 |
| 0000 | 1111 | 0 | 1111 | 0 |
| 0000 | 1111 | 1 | 0000 | 0 |

**Table 5:** Test cases performed on full four-bit adder.

Our four-bit adder cell works with each set of inputs we tested from the cases above, as illustrated on the plot below. Because these tests validate each component of the adder, we feel confident that our device functions.

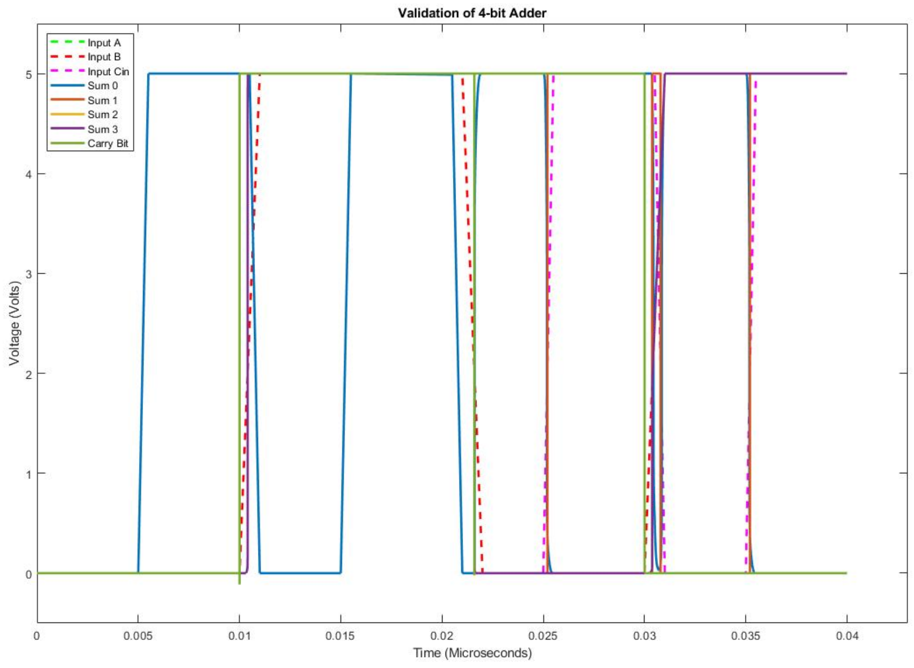

**Figure 10:** Four-bit adder validation plot