An Analog Four-Bit Multiplier Using CMOS Transistors

Pranavi Boyalakuntla
Skye Ozga

Circuits Spring 2020

Final Paper

Introduction
Logic Gates
AND Gates
XOR Gates
OR Gates
One-Bit Full Adder
Four-Bit Adder
Four-Bit Multiplier for Unsigned Values
External Image References

Introduction

Your computer uses ALU's to compute mathematical operations. One of these operations is multiplication. In order to multiply two numbers, your computer uses a process called binary multiplication. In this paper, we will walk through the process of creating and validating a 4-bit multiplier for unsigned numbers.

Logic Gates

A logic gate is an electronic component that can computer simple boolean expressions. They are a really powerful tool! With combining different types of logic gates in different orientations, you can compute some pretty complicated boolean expressions as well.

We used a multiplicity of 3 for the pMOS transistors and a multiplicity of 1 for the nMOS transistors used. This was determined experimentally to leave the cross-over voltage near 22.5 V.

For all of the logic gates described, we used pMOS transistors in a pull-up network to achieve a strong high and nMOS transistors in the complementary pull-down network to achieve a strong low.

AND Gates

The most basic and fundamental logic gate is the AND gate. There are two inputs to this logic gate. The output of an AND gate will only be high if both inputs are high. It will be low for any other combinations of inputs. The truth table is shown below.

Table 1.1

A	B	Out
0	0	0
1	0	0
0	1	0
1	1	1

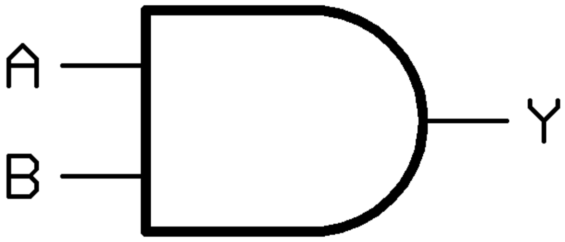

Figure 0: An AND gate

The design of this and gate is based off a NAND gate and an inverter. The NAND gate essentially has the opposite outputs as shown in table 1.1. Therefore, if we use an inverter, then the output signal will follow the truth table of the and gate.

We can divide the circuit shown in figure 1 up further. The left side of the schematic is the NAND gate and the right side of the schematic is an inverter. The NAND gate works in such a way that if both inputs are high, then both of the pMOS transistors do not conduct while both of the nMOS transistors do conduct. This leaves the output voltage of the NAND gate to be a logical low. If one of the input signals is high and the other is low, then the pMOS transistor with the low gate voltage will conduct and the nMOS transistor with the high gate voltage will conduct. This results in a output voltage that is high. Lastly, if both input signals are low, then both of the pMOS transistors will conduct current and neither of the nMOS transistors will conduct current. This causes the output voltage to be high.

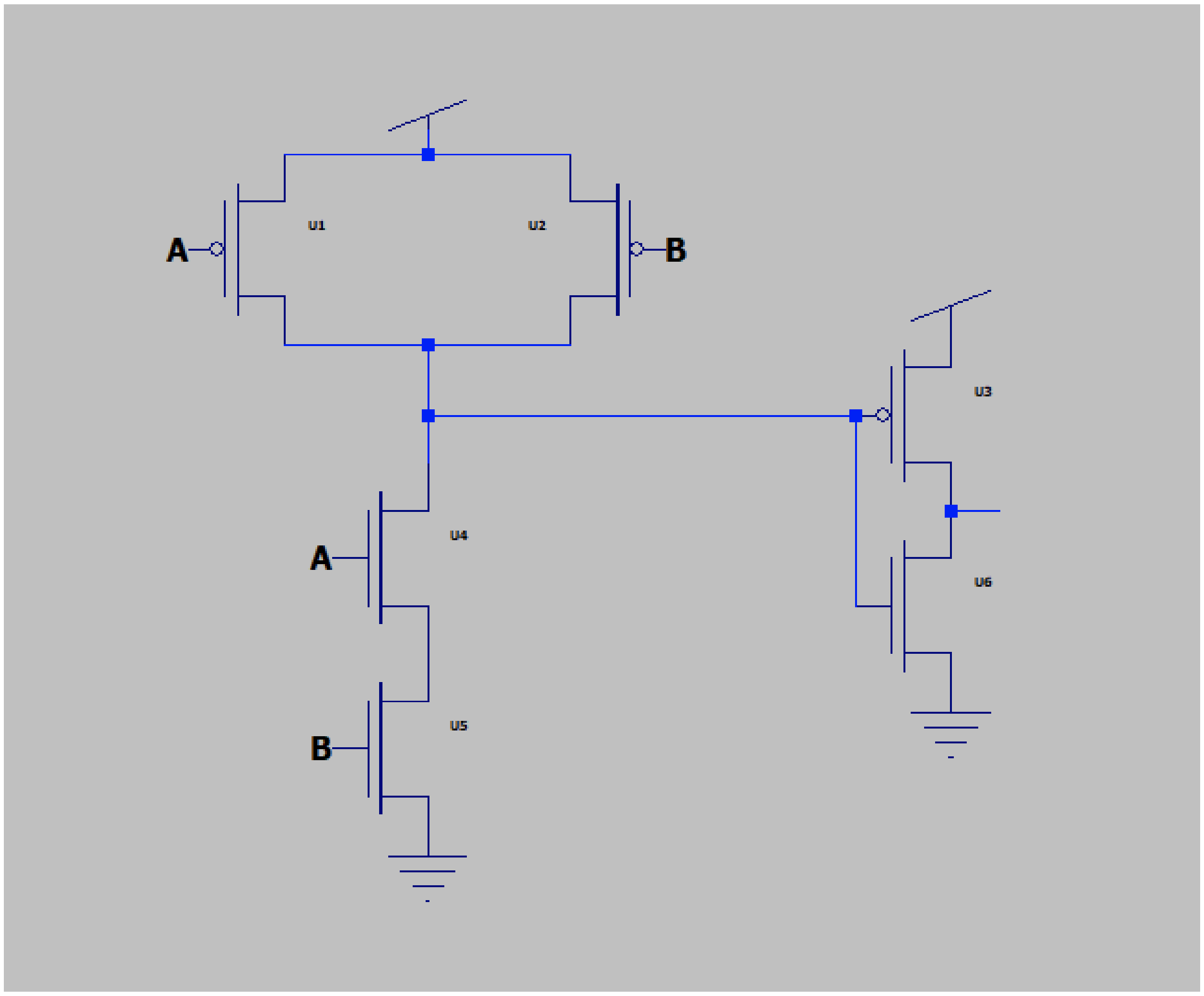

Figure 1: AND Gate Schematic

On the right hand side of figure 1, we constructed an inverter with an nMOS and pMOS transistor to invert the signal out of the NAND gate. The inverter is used in other gates as well, but we will cover how it works here. Referring to Figure 2, There is only one input to the inverter. If the input is high, then the pMOS will not conduct and the nMOS will conduct which leaves the output to be a logical low. If the input is low, then the pMOS will conduct and the nMOS will not conduct which leave the output to be a logical high. As you may notice, the output of the inverter is the opposite of what the input signal is. This is also why it is called an inverter!

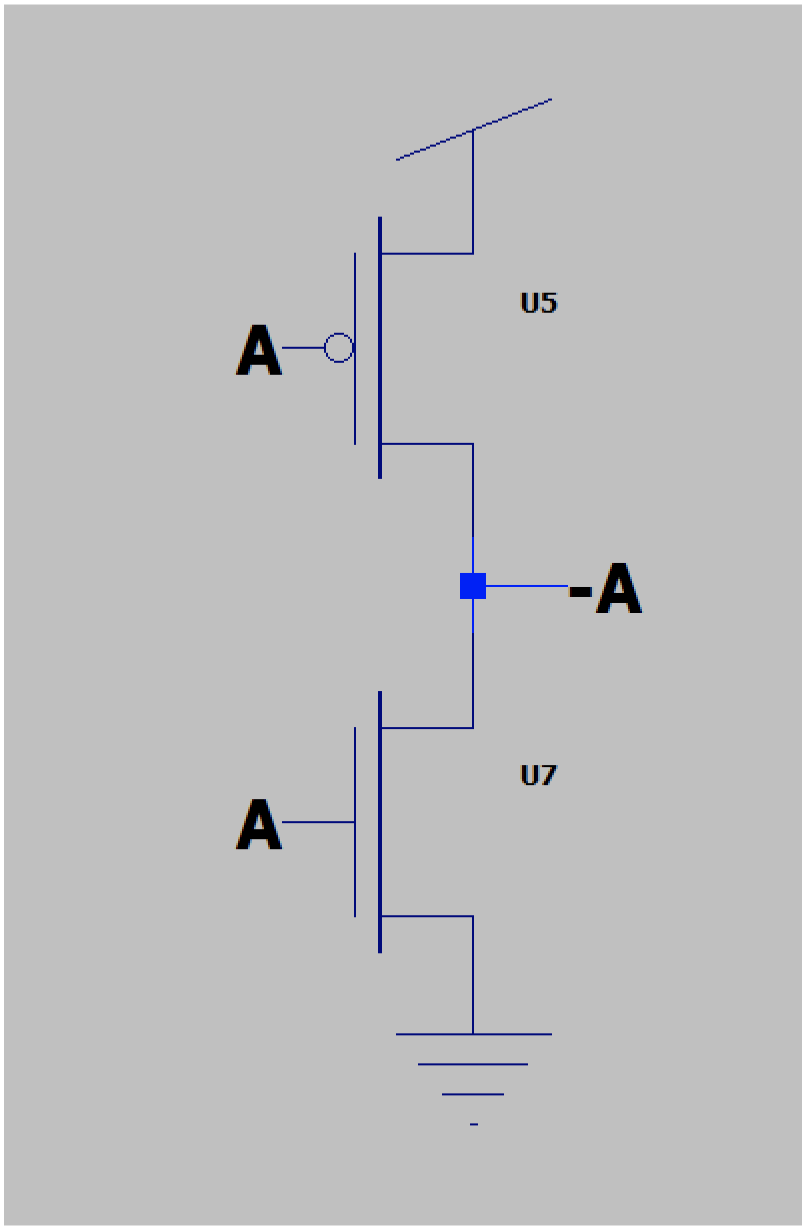

Figure 2: Inverter Schematic

We measured the voltage transfer characteristic for this AND gate through an LTSpice simulation. In order to do this, we held one input at a logical high or Vdd and swept the other input. It doesn't matter which input as the behavior doesn't change. Since this is an AND gate, we should expect that when the other input is also a logical high, that the output voltage quickly switches to high as well. This is what we see in figure 3. Just as the input we are sweeping crosses the 2.5V threshold, the AND gate output voltage starts moving towards a logical high as well. We are also able to get strong logical lows and strong logical highs from this AND gate which is seen by the fact that the voltage transfer characteristic gets indistinguishably close to ground and Vdd.

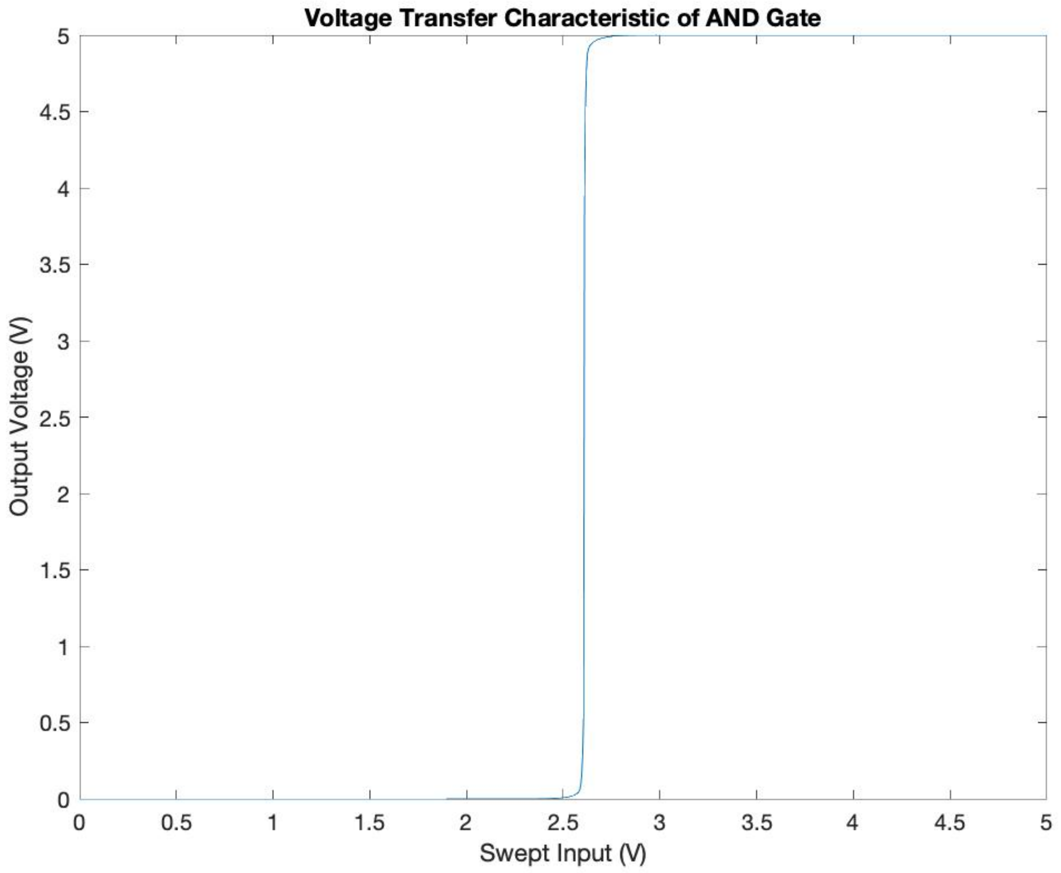

Figure 3: Voltage transfer characteristic of the AND gate.

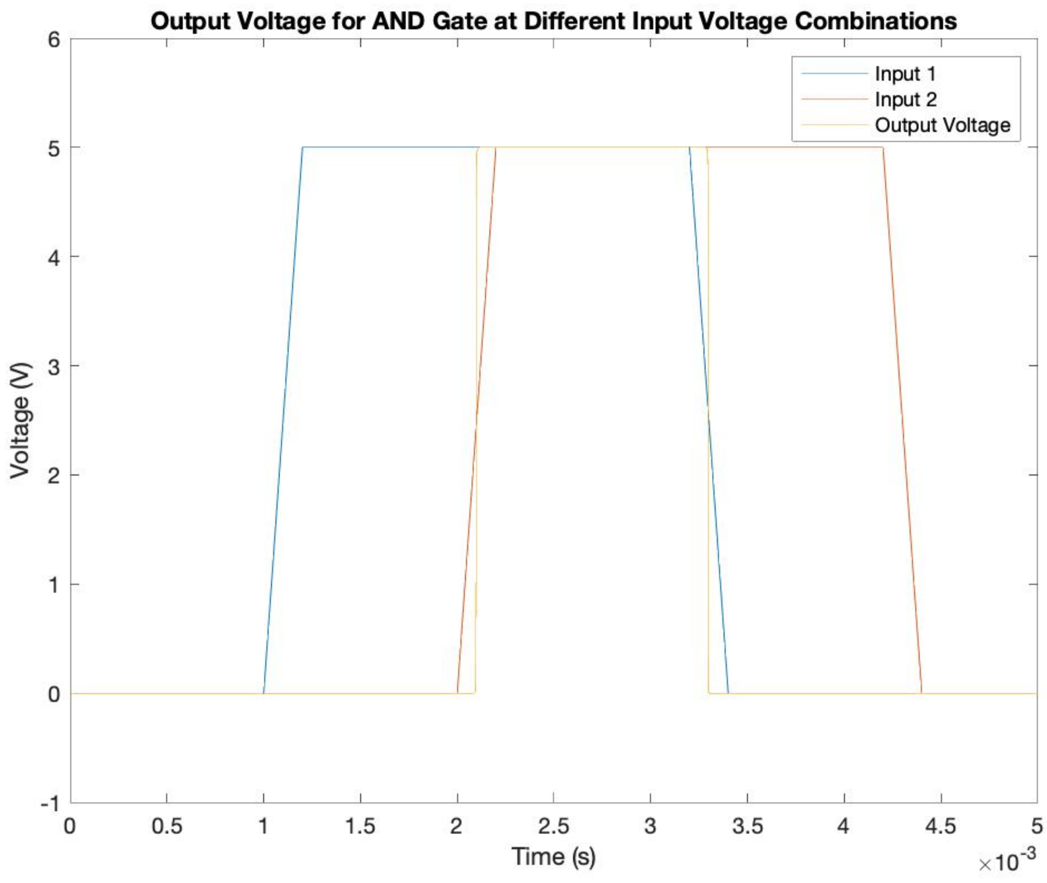

Figure 4: Output voltage of the AND gate at different input voltage combinations.

In order to find the theoretical clock speed for this multiplier, we wanted to look at the propagation delays of individual components to the final multiplier.

From sweeping the two inputs with a phase difference between them, we were able to find the propagation delay of the AND gate. We measure the propagation delay at 2.5V because that is when it switches from a logical low to a logical high. The calculated propagation delay for this AND gate was 1.42e-9 seconds.

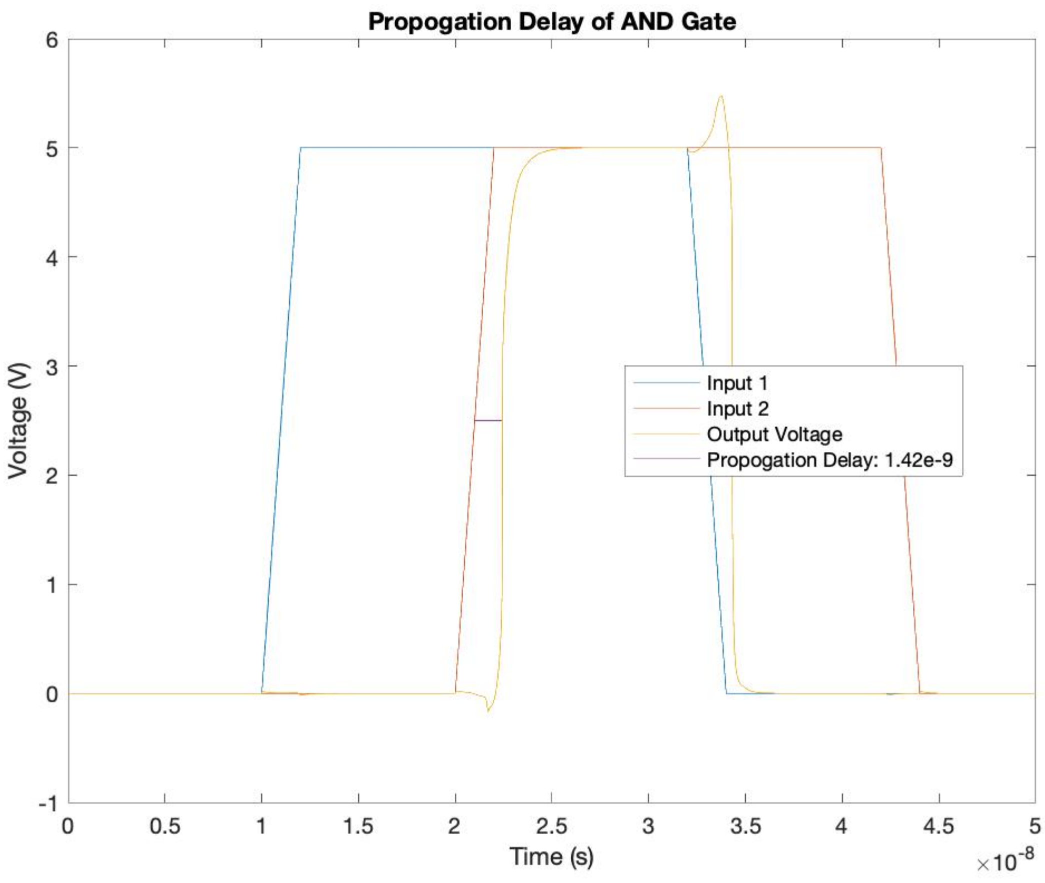

Figure 5: Propagation delay of an AND gate

XOR Gates

An XOR gate is only high if and only if one of the two inputs is high. If both inputs are high, then the output will be low. It follows the truth table shown in table 1.2.

Table 1.2

A	B	Out
0	0	0
1	0	1
0	1	1
1	1	0

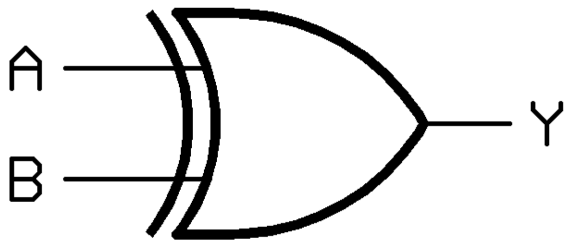

Figure 6: An XOR Gate

In order to construct an XOR Gate, we used a few subcomponents. One of those was an inverter again as shown in the top left. The functionality of an inverter was explained in the previous AND Gate section.

The right hand side of figure 7 shows the schematic for an XOR gate.

If all of the inputs are low, then the inverted inputs will be high. In the pull up network this means that U1 and U4 will not conduct while U3 and U2 will because they are pMOS transistors. U1 and U2 are in series, however, and so is U3 and U4. This means that if one of U1 or U2 is not conducting, that branch will not pass current and if one of U3 or U4 is not conducting, that branch will also not pass current. On the pull down network's side, the branch with two inverted signal inputs will pass current. Since this is the case, the final output voltage will be a logical low.

If one of the inputs is high (assume A) and the other input is low (assume B), then the right branch in the pull up network will conduct. This is because B is low so the first pMOS transistor in the right branch will conduct and the inverted input signal of A which controls the gate voltage of the pMOS transistor below it will also allow it to conduct. In looking at the pull down network, neither of the branches will allow current to flow which will keep the output voltage of the XOR gate to be a logical high. This logic stays true if B is high and A is low.

Lastly, if all of the inputs are high, a similar situation as with when all of the inputs were low occurs and the final output voltage is a logical low.

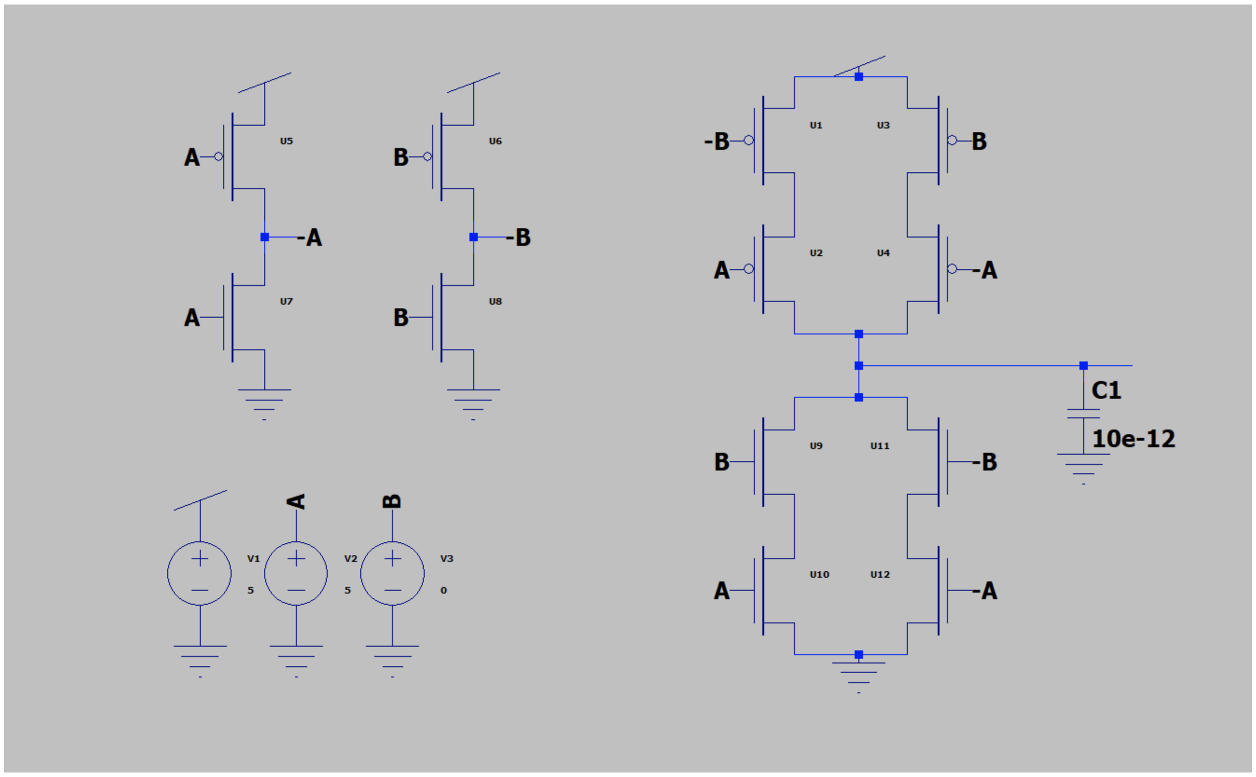

Figure 7: XOR Gate Schematic

We measured the voltage transfer characteristic of this XOR Gate through the LTSpice simulation. In order to do this, we held one input at a logical low while we swept the other one. This is so that once the voltage we are sweeping reaches the threshold between logical low and logical high, then, as there would only be one input that is a logical high, the output voltage should start increasing towards Vdd. This is what we see in the VTC plot in figure 8. The XOR gate visibly starts increasing towards Vdd a little before 2.5V. This means that for this gate, the actual threshold between a logical low and logical high is a little bit before 2.5V.

We also see that we are able to reach strong logical lows and strong logical highs from the VTC plot because the output voltage can get very close to ground and very close to Vdd.

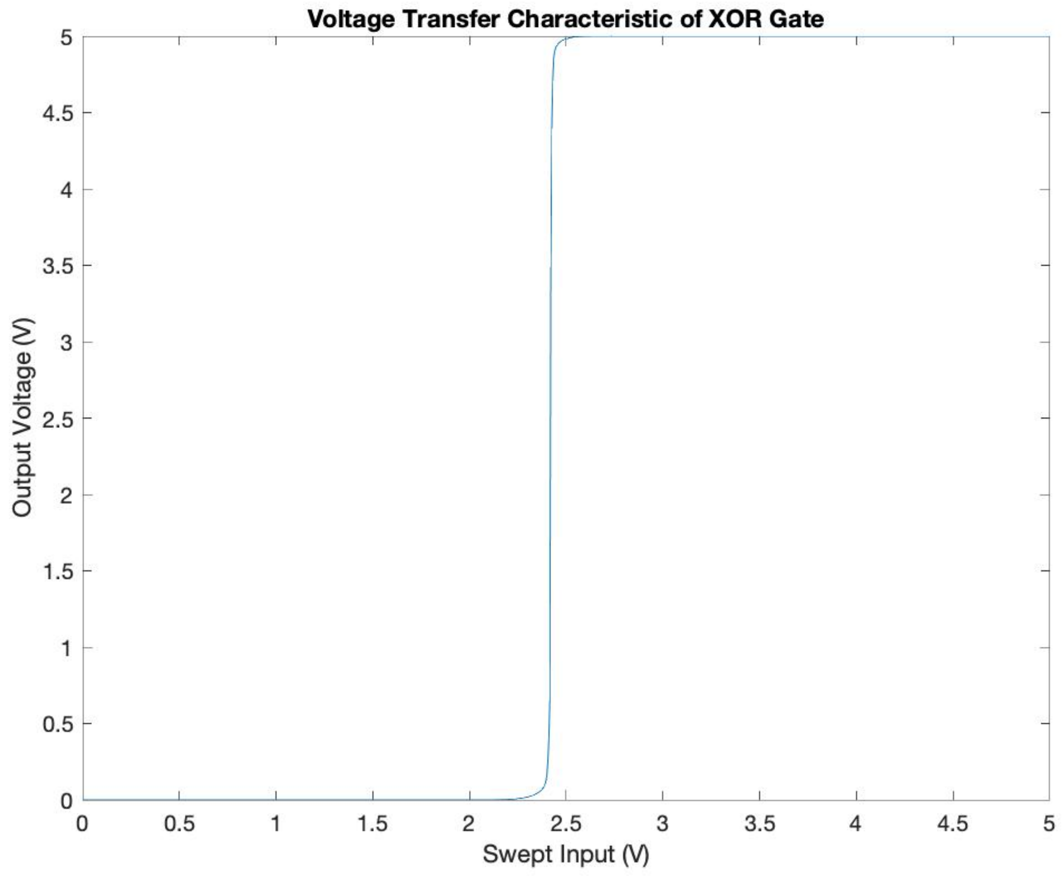

Figure 8: Voltage transfer characteristic of an XOR gate.

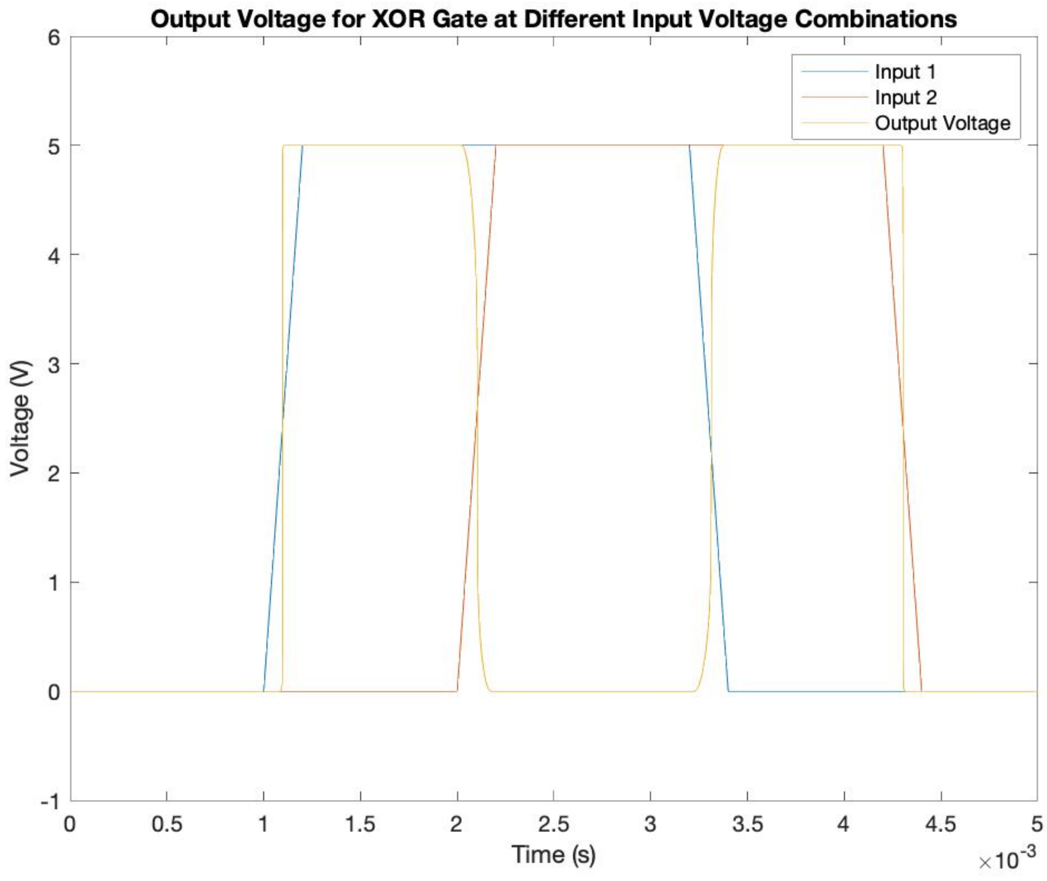

Figure 9: Output voltage for the XOR gate at different input voltage combinations.

From sweeping the two inputs with a phase difference between them, we were able to find the propagation delay of the XOR gate. The calculated propagation delay for this XOR gate was 1.52e-9 seconds. It is slightly larger than the propagation delay for the AND gate because there are more components to this logic gate.

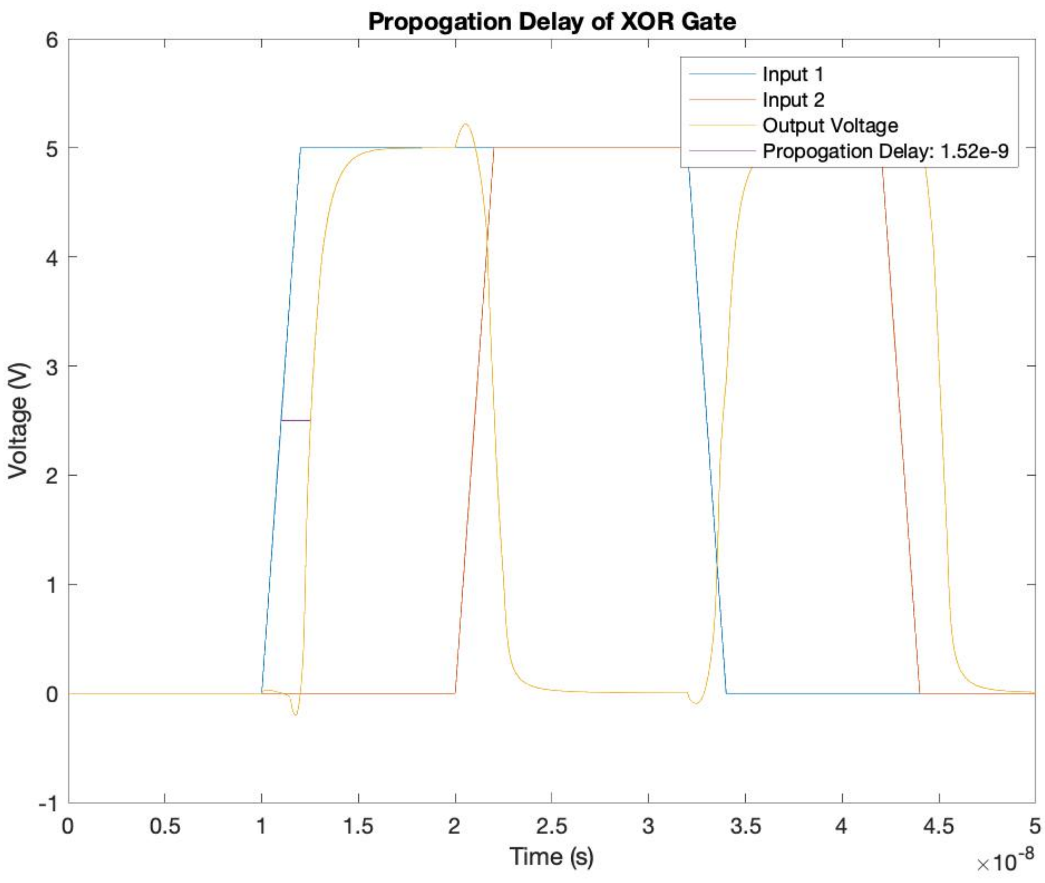

Figure 10: Propagation delay for the XOR Gate.

OR Gates

An OR gate is high if either of the inputs are high. It is also high if both of the inputs are high. It follows the truth table shown in table 1.3.

Table 1.3

A	B	Out
0	0	0
1	0	1
0	1	1
1	1	1

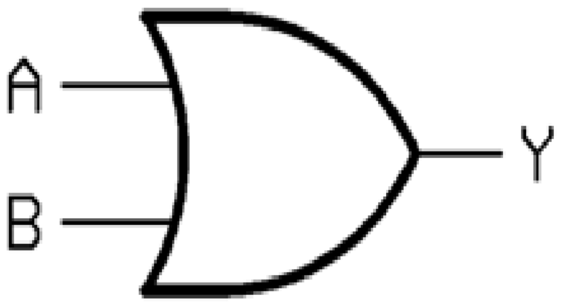

Figure 11: An OR Gate

An OR gate is actually constructed of a NOR gate and an inverter. A NOR gate has the exact opposite response to input signals in comparison with an OR gate. The circuit and behavior of an inverter was discussed in the AND gate explanation.

Figure 12 shows the schematic of an OR gate. It takes two inputs and produces one output.

If both of the inputs are logical lows, then both of the pMOS transistors in the pull up network will conduct. In addition to that, both of the nMOS transistors in the pull down network will not conduct. This leaves the output voltage of the NOR gate to be a logical high.

If one of the inputs is high (assume A) and the other is low (assume B), then one of the pMOS transistors in the pull up network will conduct while the other one does not. This will not allow current to flow through the pull up network due to the fact that the two pMOS transistors are in series. In addition, in the pull down network, the left branch will conduct while the right one does not. This means that the output voltage in this scenario will be a logical low for the NOR gate.

Lastly, if both of the inputs are high, then neither of the transistors in the pMOS pull up network will conduct. In addition, both of the nMOS transistors in the pull down network will conduct. Therefore, that will leave the output voltage to be a logical low.

Next, the output from the NOR gate is fed into the inverter which will flip the output so that it follows the truth table shown in table 1.3.

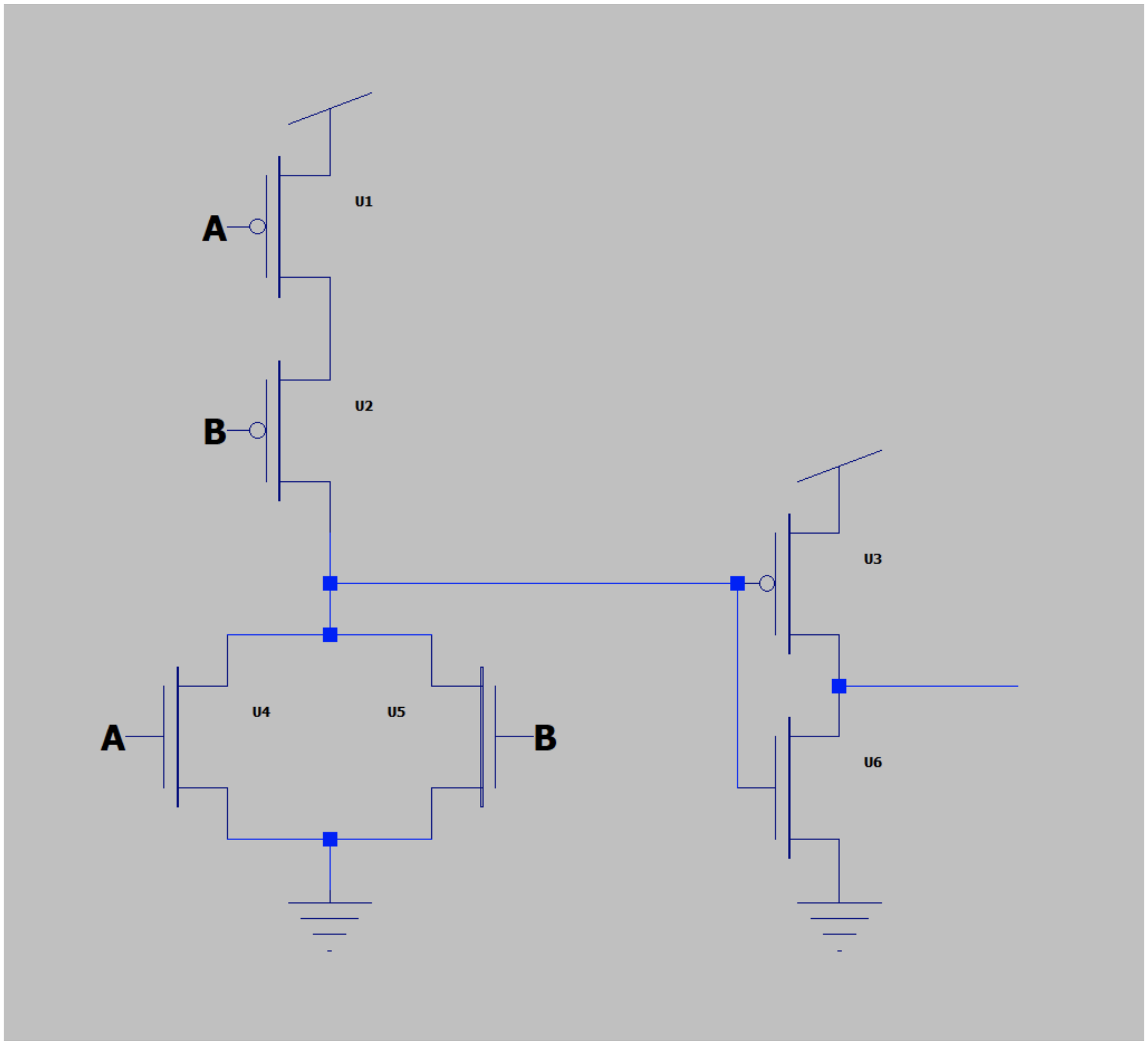

Figure 12: OR Gate Schematic

In order to find the voltage transfer characteristic of the OR gate, we held one input at ground while sweeping the other input from ground to Vdd. The idea being that once the voltage we are sweeping crosses over from a logical low to a logical high voltage that the output voltage of the OR gate would start rising toward Vdd. We see this happen in the voltage transfer characteristic plot for the OR gate. We expect that it should start making the transition at 2.5V. However, we see that transition start a little bit before the expected 2.5V. This means that the threshold between logical low and high for this gate is in between 2V and 2.5V.

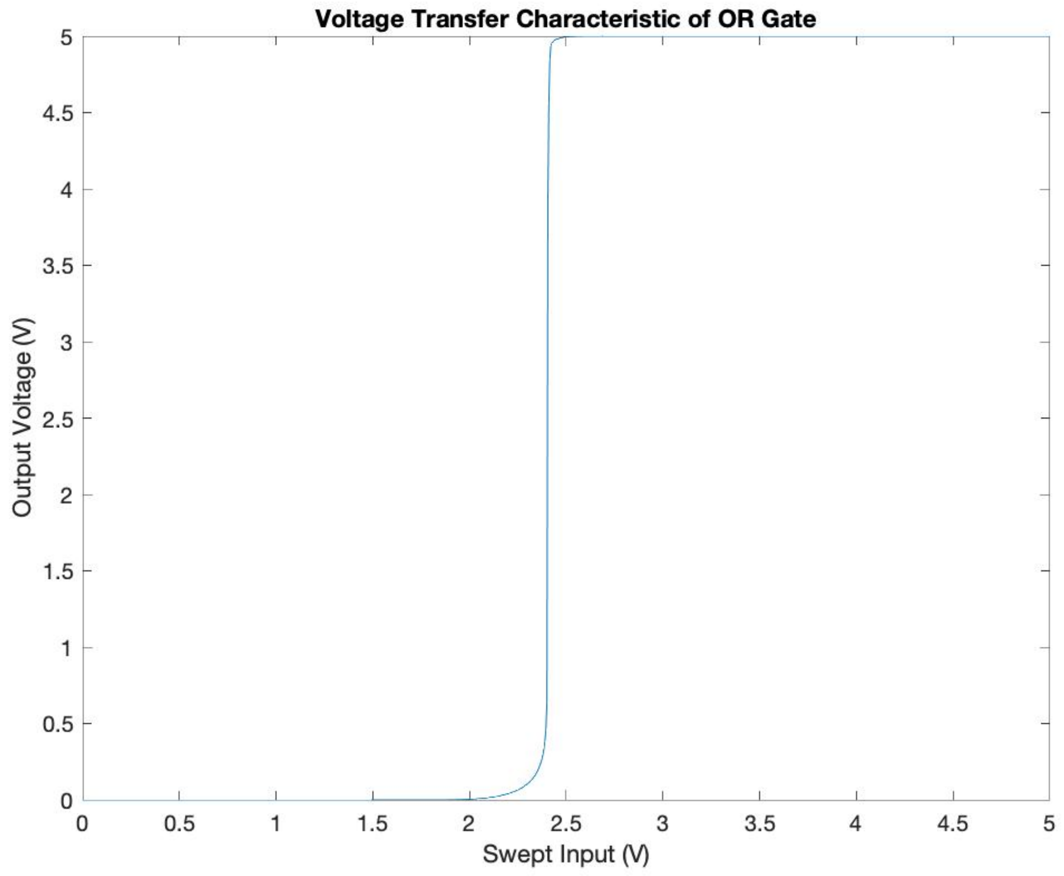

Figure 13: Voltage transfer characteristic of the OR gate.

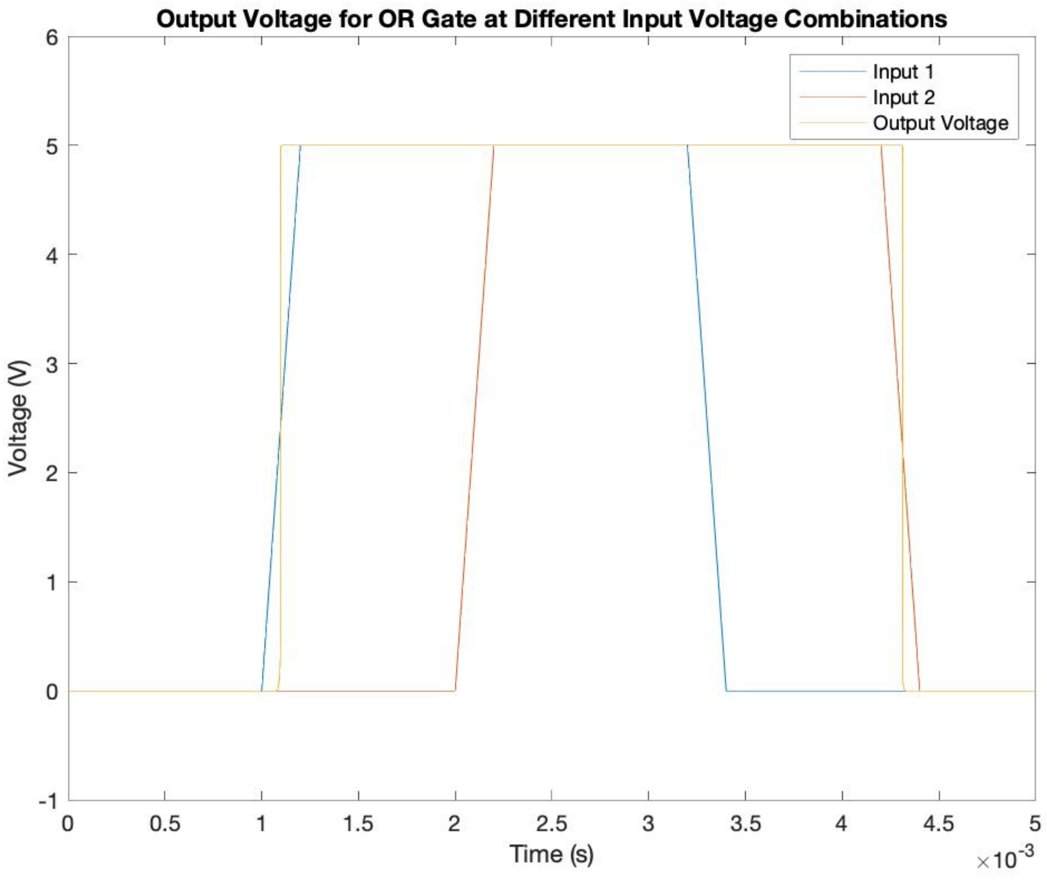

Figure 14: Output voltage at different input voltage combinations for the OR gate.

From sweeping the two inputs with a phase difference between them, we were able to find the propagation delay of the OR gate. The calculated propagation delay for this OR gate was 1.42e-9 seconds. It is exactly the same as the AND gate. This makes sense because they both have the same number of components in them.

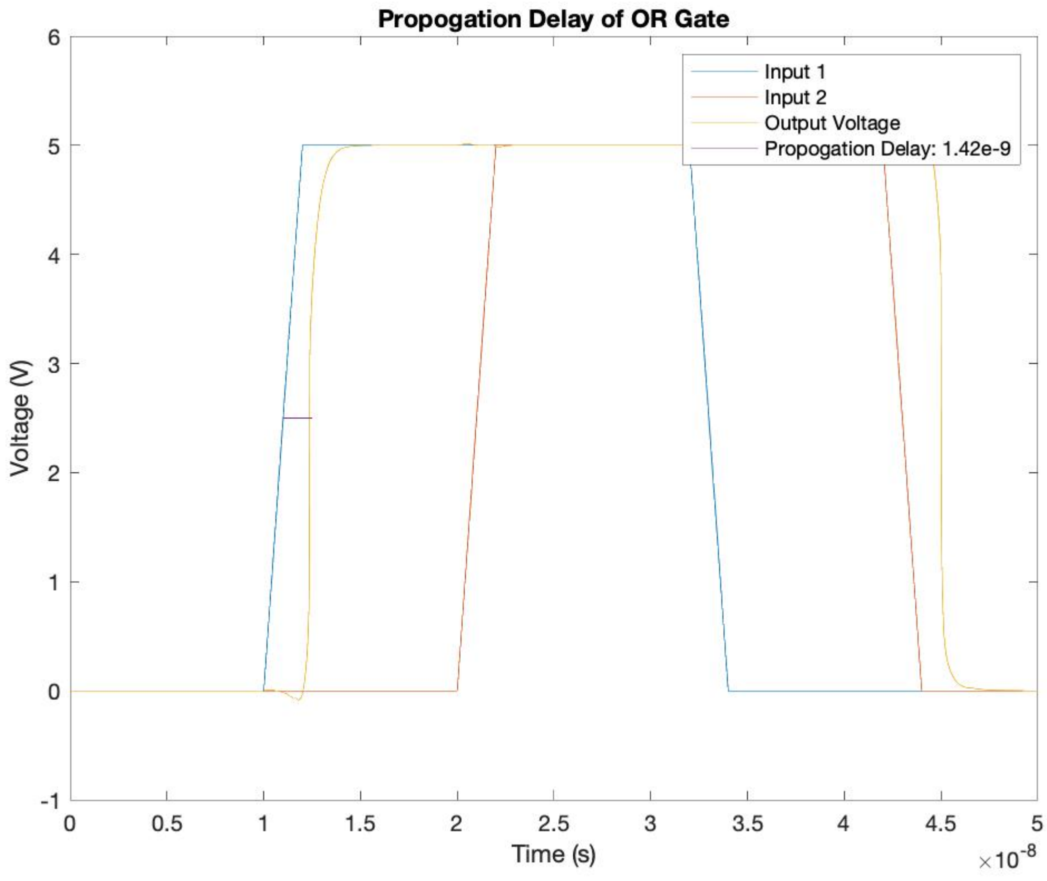

Figure 15: Propagation delay for the OR gate.

One-Bit Full Adder

The next layer of abstraction in creating the multiplier, is building out a full adder from the different gates we build above. A full adder is a circuit that can add binary digits together. The circuit has 3 inputs: the first binary digit, the second binary digit, and an input carry in. The two binary digit inputs are the ones that would be added together. The carry in is a digit that could have potentially carried over from a previous addition. If there is no previous addition this value is automatically 0. A full adder outputs the sum of the 3 inputs and a carry out if one exists.

Here is a case where a carry out would exist with regards to binary addition:
Input A = 1
Input B = 1
Carryin = 1
1 + 1 + 1 = 11
The rightmost 1 of the answer is the sum output and the leftmost 1 in the answer is the carryout. It is really similar to decimal addition!

Table 2.1

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
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

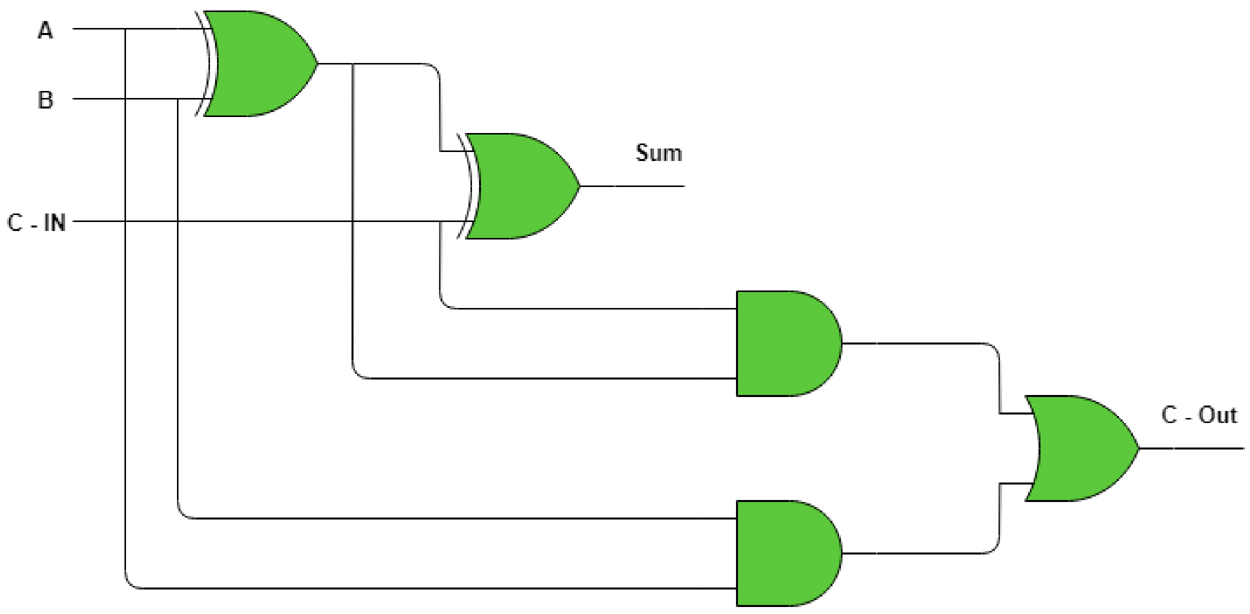

Figure 15: Full adder schematic using logic gates.

Figure 15 shows us how a full adder is constructed using logic gates. We choose to represent the gates with the symbols shown in the gate descriptions because it is much easier to understand what is going on as opposed to figure 6 which is considerably more complicated.

Two of the inputs, A and B, and sent through an XOR gate. The result of this XOR gate is the sum of A and B without considering the carry in value. Then, we are able to xor this value with the carry in value to find the final sum of the 3 inputs. For illustration as to why an XOR gate functions as adding its two inputs refer to table 2.2.

```
Table 2.2
  A  |  B  | Sum
------------------
  0     0     0     Explanation 0 + 0 = 0
  0     1     1                 0 + 1 = 1
  1     0     1                 1 + 0 = 1
  1     1     0                 1 + 1 = 10 (0 is sum and 1 is the carryout)
```

In order to find the accurate carry out, we pass the A and B inputs into an AND gate, and also the initial sum and the carry in input into another AND gate. We pass these two outputs into another OR gate, and the final output from that is our carry out value.

Essentially, what we are doing here is checking for an overflow with the two initial inputs and then checking for an overflow with the sum of all three. ANDing checks for an overflow because there will only be an overflow if the two values are 1, otherwise the final sum will only be a 1-bit value. Passing the output of those two AND gates into an or gate says that if there is an overflow in either of the two adding processes, there will be an overflow out of the circuit.

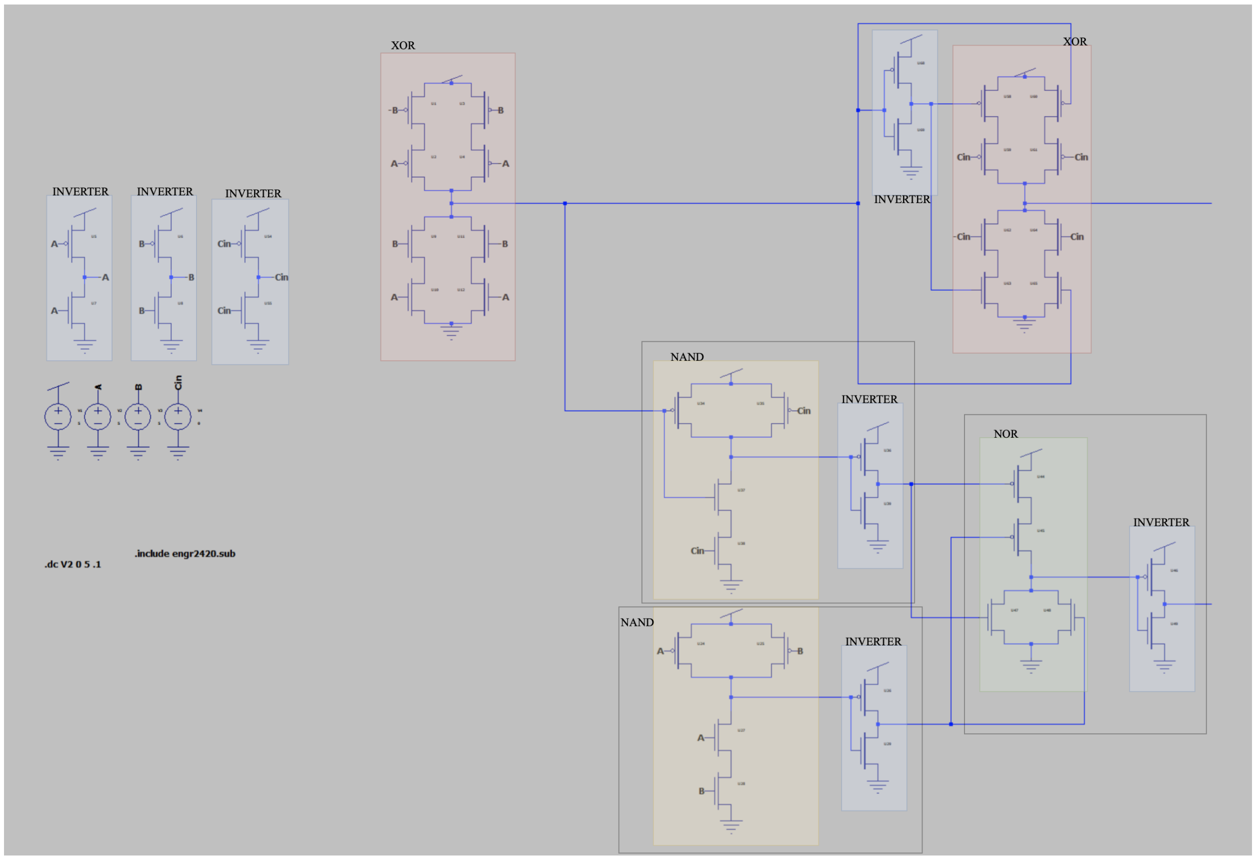

Figure 15: Schematic of a Full Adder in LTSpice. The components defined earlier are shown in the different colored boxes with the inverters paired with their corresponding uninverted gates.

The LTSpice model of the circuit shown in figure 5 is shown in figure 6.

In order to calculate the propagation delay of the full adder, we increased and decreased the two inputs with some time delay between them. As the carryout had more components before the final answer is reached, it would have a larger delay. Therefore, we held the propagation delay of the carryout to be the propagation delay of the whole full adder. We found this value to be 1.34e-8 seconds.

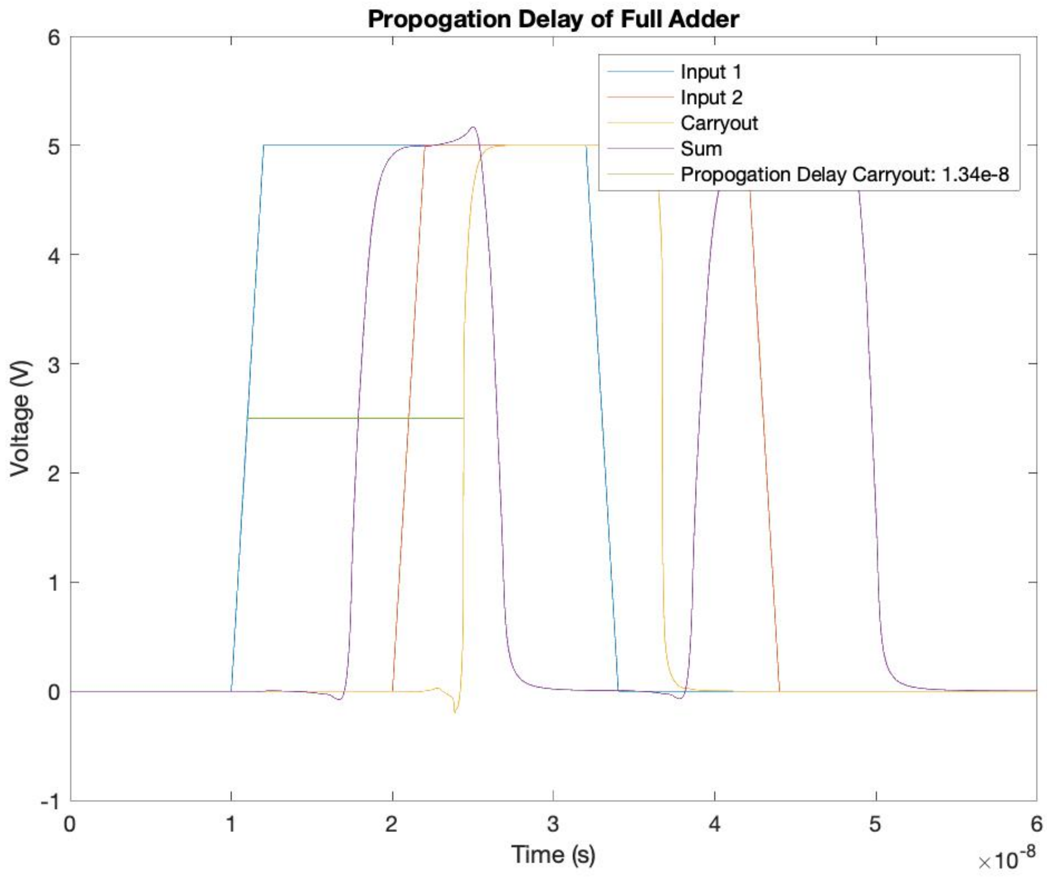

Figure 16: Propagation delay of a full adder.

Four-Bit Adder

In order to create a four-bit adder which brings us one step closer to a multiplier, we need to string together a few one bit adders in a way that reflects how you might to binary addition by hand.

```
Ex.
A = 0111, B = 0010

0 1 1 1
0 0 1 0 +
---------
      1 (no carryover)

0 1 1 1
0 0 1 0 +
---------
    1 1 (1 carryover)
```

```
  1
0 1 1 1
0 0 1 0 +
---------
  1 1 1 (1 carryover)

1
0 1 1 1
0 0 1 0 +
---------
1 1 1 1 (0 overflow)
```

As you can see in the example above, each column addition operation can be solved with a single one-bit full adder. And the carryover from the previous column gets pushed into the next column similar to a carryout from a previous one-bit full adder being connected to the next one's carry in value. Therefore, we can design a circuit as shown below in figure 17.

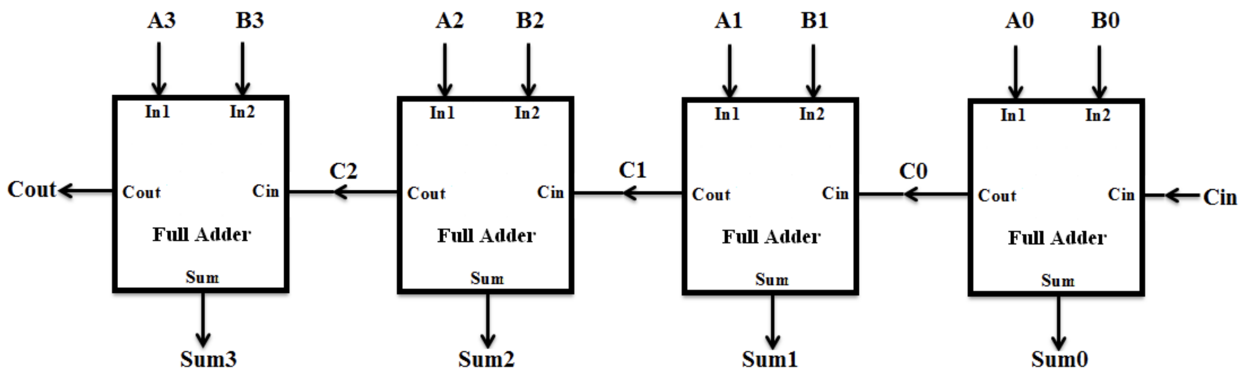

Figure 17: Four bit adder created by stringing together four one-bit adders.

We took the model above and used the LTSpice models of the full adder shown in figure 6 to model the four-bit adder shown in figure 18.

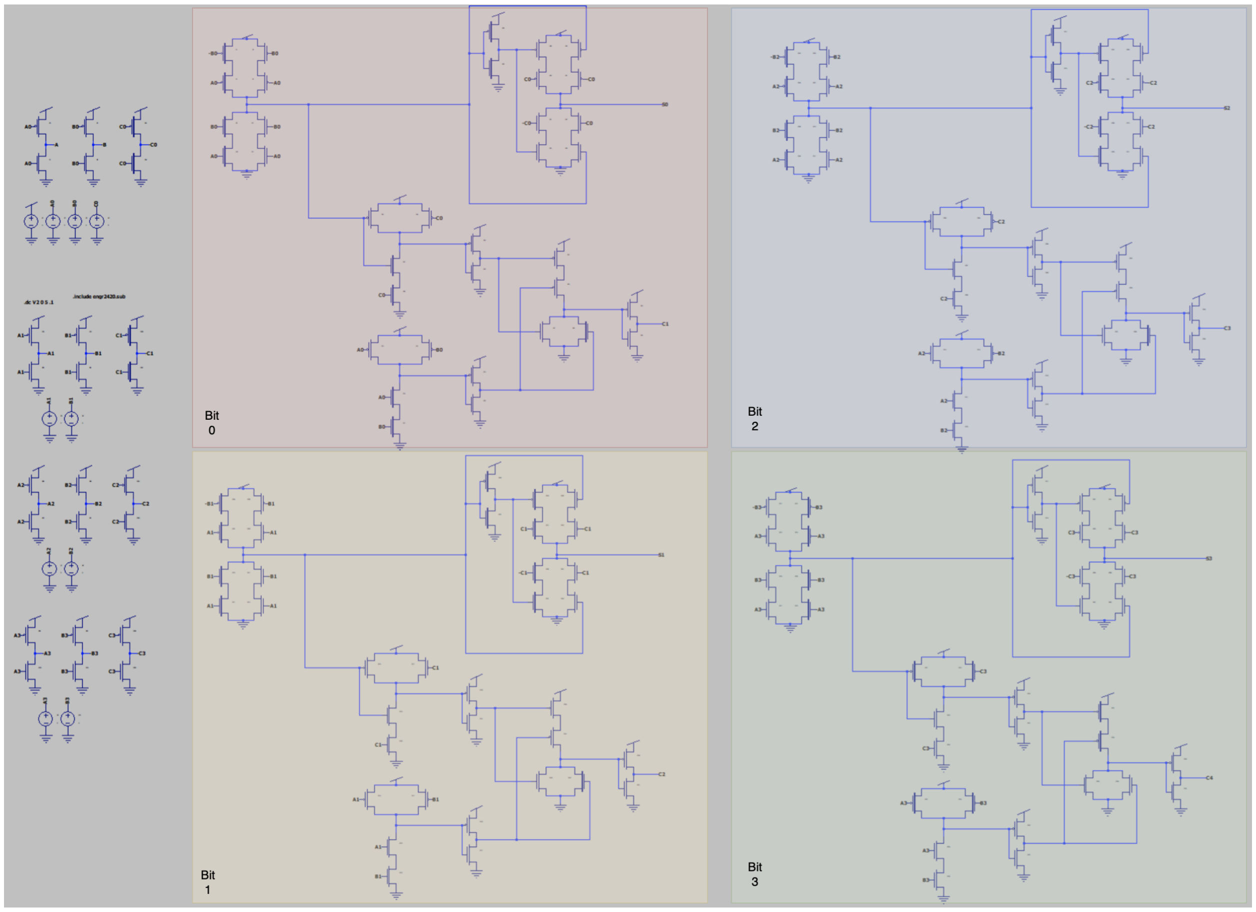

Figure 18: Four-bit adder LTSpice model. In order to connect things, we used nets to connect things rather than wires to keep the organization more clearly understood.

We tested the four-bit adder with the following values:

```
Table 2.1
    A     |    B     |   Expected Sum    |   Sum    |    Cout
----------------------------------------
   0000      0000         0000          0000        0
   1111      1111         1110          1110        1
   1111      0000         1111          1111        0
   0000      1111         1111          1111        0
   0111      0010         1111          1111        1
```

In addition, since the four-bit adder consists of four one-bit full adders, we can say that the propagation delay will be four times the propagation delay of the one-bit full adder. Therefore, the propagation delay of the four-bit adder is equal to 5.36e-8 seconds.

Four-Bit Multiplier for Unsigned Values

Now that we have all of the components ready to create a four-bit multiplier, we can assemble it.

```
How multiplication works...
                                B3      B2      B1      B0
                                A3      A2      A1      A0 *
-----------------------------------------------------------------------
        0       0       0     A0*B3   A0*B2   A0*B1   A0*B0
        0       0     A1*B3   A1*B2   A1*B1   A1*B0     0
        0     A2*B3   A2*B2   A2*B1   A2*B0     0       0
      A3*B3   A3*B2   A3*B1   A3*B0     0       0       0           +
-----------------------------------------------------------------------
D7      D6      D5      D4      D3      D2      D1      D0
```

Looking at the block above, we can see the general form for how multiplication works. The pattern we see is that the second multiplier gets multiplied with entire top row bitwise. We know that multiplying two one bit numbers is equivalent to putting them in an AND gate together. For example, A0*B0 is equivalent to putting A0 and B0 as two inputs to an AND gate. This is proven in table 4.1.

```
Table 4.1

A | B | Out
-----------
0   0    0   Explanation 0 * 0 = 0
1   0    0               1 * 0 = 0
0   1    0               0 * 1 = 1
1   1    1               1 * 1 = 1
```

Once we have multiplied everything out, we also need to add everything. For each column, we see that there are four numbers to add together. We can use the four bit adder that we made in a previous part of this paper to solve this. We also need to be careful about carry overs, so we simply need to string the previous adder's carry out to the next adder's carry in (the same technique we used in constructing the four-bit adder). Lastly, the final carryout from all of the adding becomes the eighth bit in the final answer. All of this culminates in the final schematic of the four-bit multiplier.

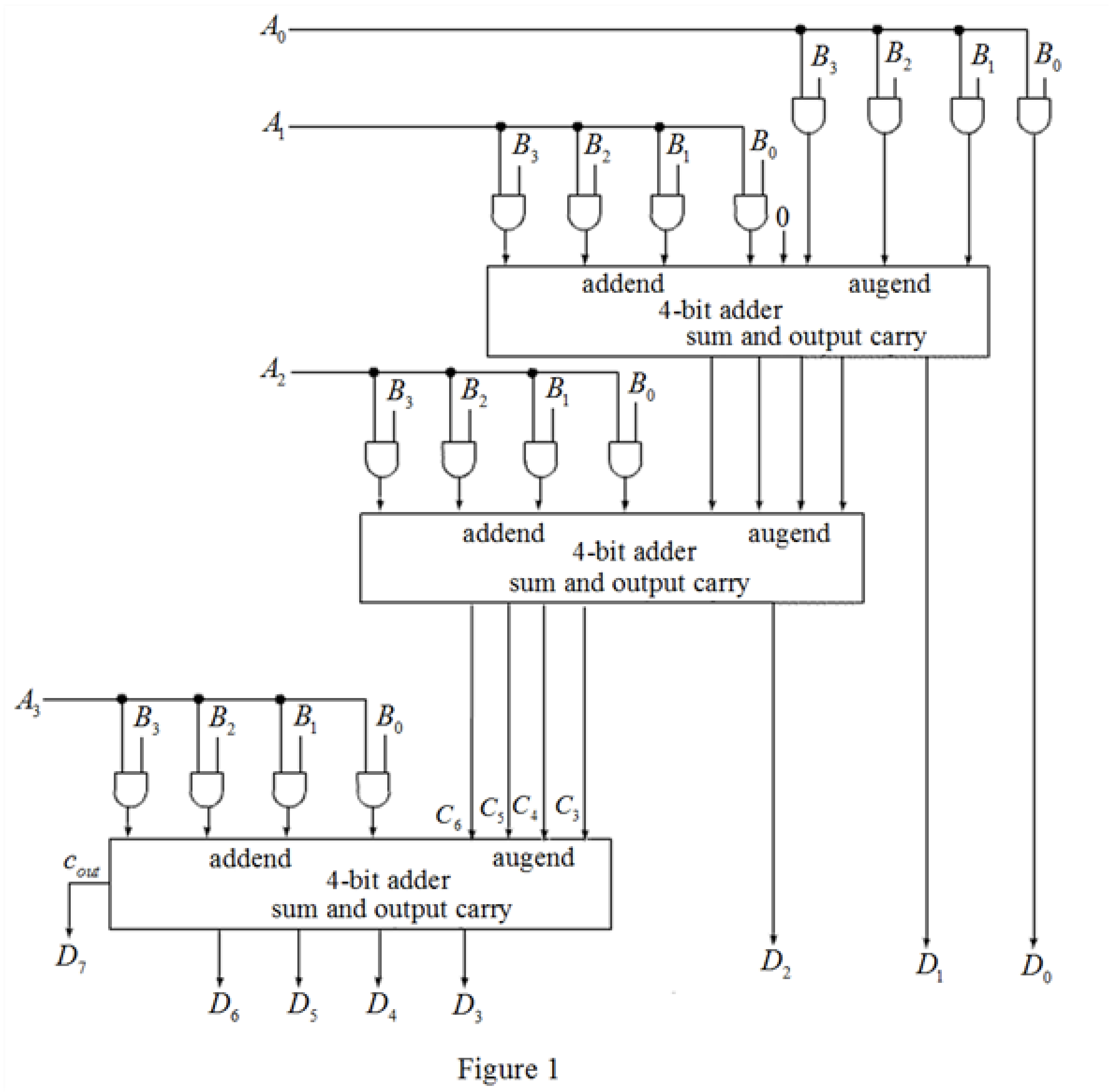

Figure1 9: Schematic using logic gates for a four-bit multiplier

The schematic from figure 9 matches the steps shown in the multiplication explanation above.

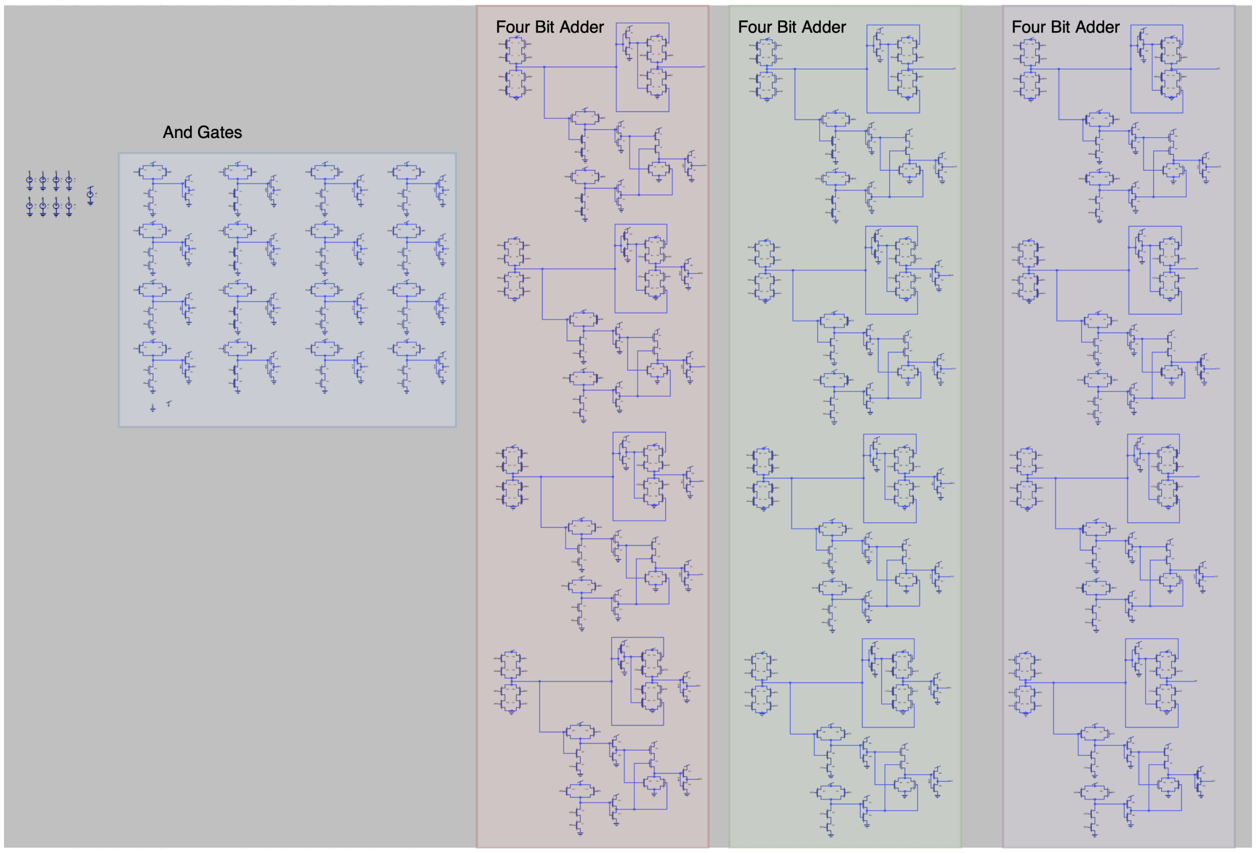

Figure 20: LTSpice schematic for the whole four-bit multiplier

We then tested the final 4-bit multiplier by multiplying 1111 with 1111. This ensures that all 8 bits will turn on. In order to see the propagation delay, we tracked the voltage values for each of the "bits".

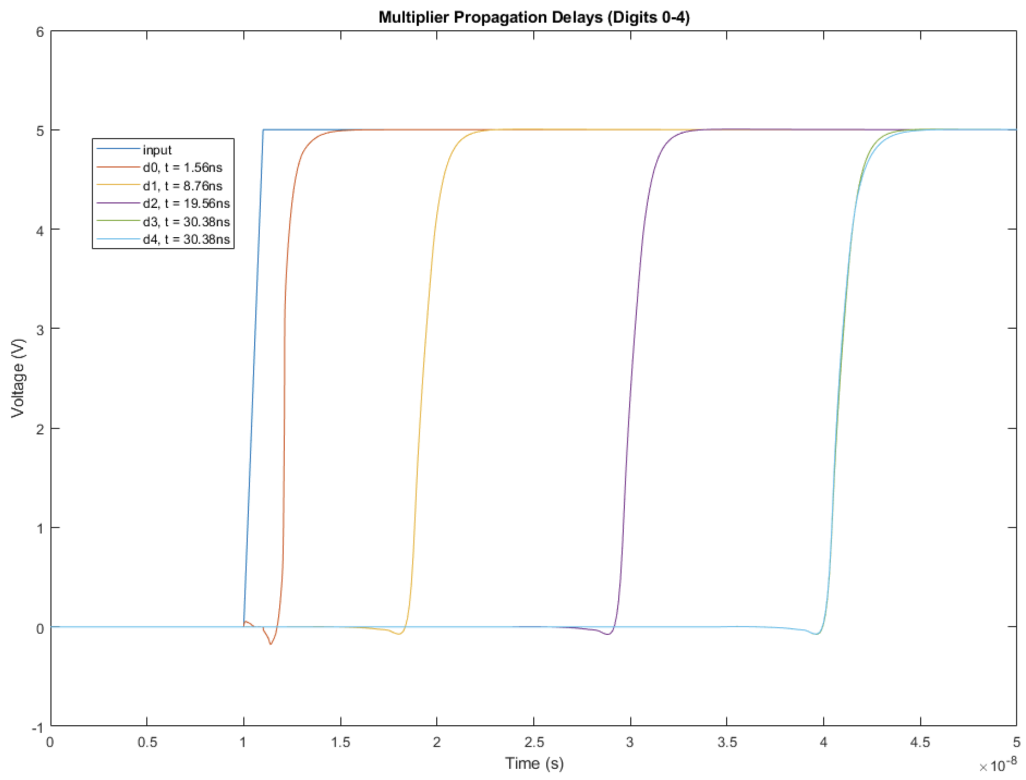

Figure 21: Multiplier voltages for the first 5 bits as the multiplication process is occurring.

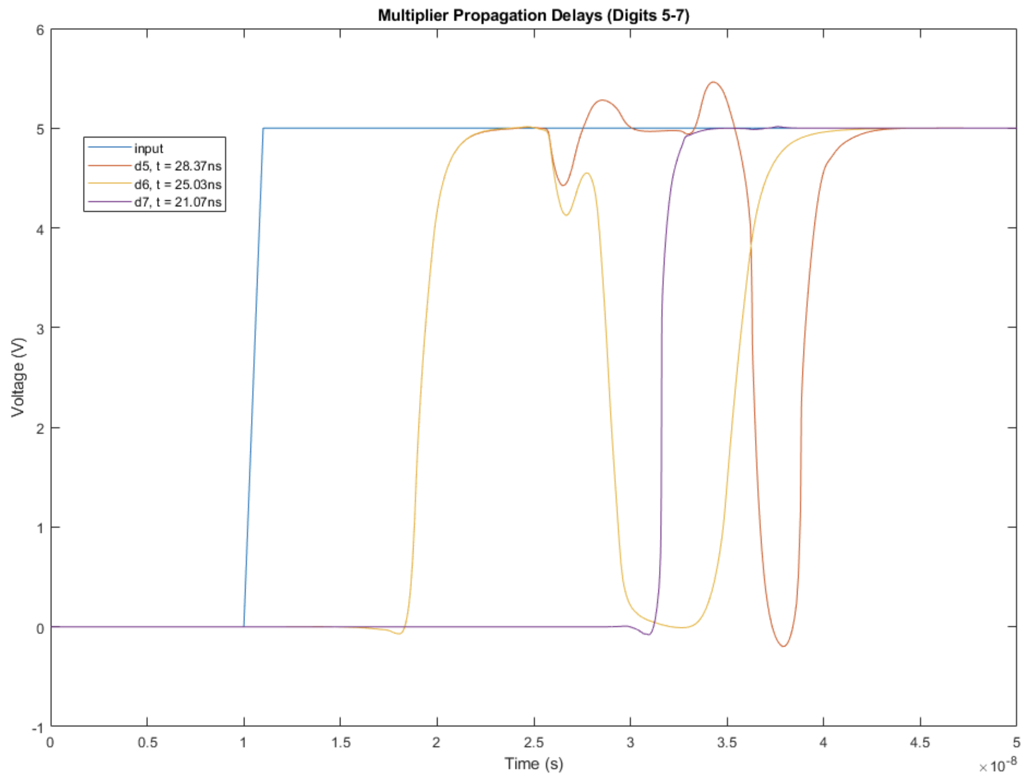

Figure 22: Multiplier voltages for the last 3 bits as the multiplication process is occurring.

We can see in figure 22 that the propagation delay for the multiplier is 21.07 nanoseconds.

We can also see that there is some variation in what some of the outputs are towards in figure 22. We also see that by the end of the last bit settling, they settle at the high voltage as well. In digital logic, the final output would only be looked at at the positive edge of the clock. The final theoretical clock speed is 32 MHz. This was calculated by looking at the reciprocal of the propagation delay of our slowest digit.

Normal CPU clock speeds are 3.5 to 4 GHz. Our multiplier is considerably slower than something that might work in a CPU. This is due to the hardware of the components we are using. If we used components that could switch faster, then our theoretical clock speed would be faster.

External Image References

https://www.homemade-circuits.com/logic-gates-work/

https://www.geeksforgeeks.org/full-adder-in-digital-logic/

https://esrd2014.blogspot.com/p/4-bit-carry-ripple-adder.html

https://www.chegg.com/homework-help/binary-multiplier-multiplies-two-unsigned-four-bit-numbers-u-chapter-4-problem-20p-solution-9780134529561-exc