Binary Addition Using Multiplexers

Toki Nishikawa

September 27 2022
6-8pm Tuesday lab with Kim Nguyen/Nick Boudreau

The objective of the lab is to construct a circuit that adds two 2-bit binary numbers, producing a 3-bit result, using 8:1 multiplexers.

1 Design

The first step in the design process is to produce all of the truth tables needed for 2-bit binary addition. Let M and N represent the two 2-bit binary numbers, and let R represent the 3-bit binary sum. The addition equation is

$$\begin{array}{ccc} & M_1 & M_0 \\ + & N_1 & N_0 \\ \hline R_2 & R_1 & R_0 \end{array}$$

Tables 1-4 show the truth tables associated with the least significant bit, the carry out of the least significant bit, the second bit, and the carry out of the second bit (which is just the third bit).

M_0	N_0	R_0
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: Truth table for the least significant bit

Each of the four truth tables can be implemented using an 8:1 multiplexer. **Figure 1** shows the logic diagrams associated with each truth table. Note that each multiplexer has two outputs, a standard output Y and an inverted output W. Also note that the first two truth tables (**Tables 1 and 2**) only have two inputs, M_0 and N_0, and so their corresponding multiplexers will have four unused inputs. As shown in **Figure 1**, these unused inputs are connected to ground, along with the unused select line C. The unused inverted output W is left unconnected for all multiplexers.

M_0	N_0	C_{out0}
0	0	0
0	1	0
1	0	0
1	1	1

Table 2: Truth table for the carry out of the least significant bit.

M_1	N_1	C_{out0}	R_1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Table 3: Truth table for the second bit.

M_1	N_1	C_{out0}	R_2
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 4: Truth table for the carry out of the second bit.

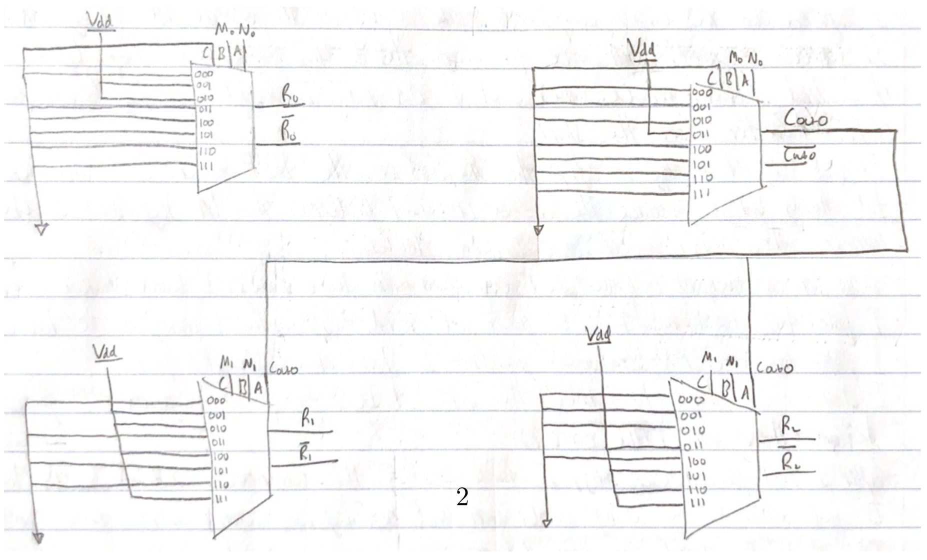

Figure 1: Logic diagrams associated with the truth tables in tables 1-4

The inputs M_0, N_0, M_1, and N_1 shown in **Figure 1** are connected to the DIP switch circuit shown in **Figure 2**. You may connect the outputs R_0, R_1, and R_2 to LEDs to easily visualize the outputs, but I was lazy and just used a multimeter to check the voltages.

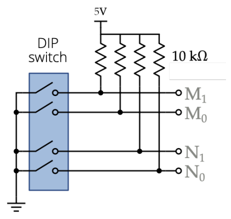

Figure 2: DIP switch circuit

The full 2-bit adder circuit is shown in **Figure 3**. In order to enable the inputs, $\bar{G}$ must be at a low logic level. Note the *carry wire* which connects the output of the second multiplexer, corresponding to **Table 2**, with the inputs of the third and fourth multiplexers, corresponding to **Tables 3 and 4**.

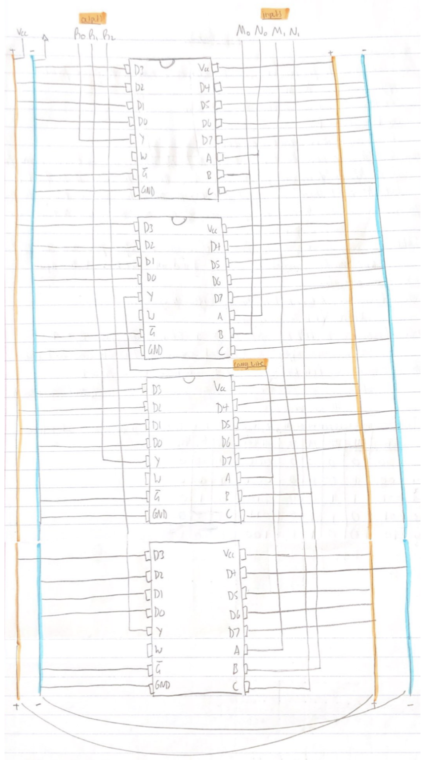

Figure 3: Full 2-bit adder circuit

2 Implementation

The completed circuit is shown in **Figure 4**. The first step was to create the circuit for the DIP switch shown in **Figure 2**. I had no issues with this part – an *on* state produced a low logic value while an *off* state produced a high logic value.

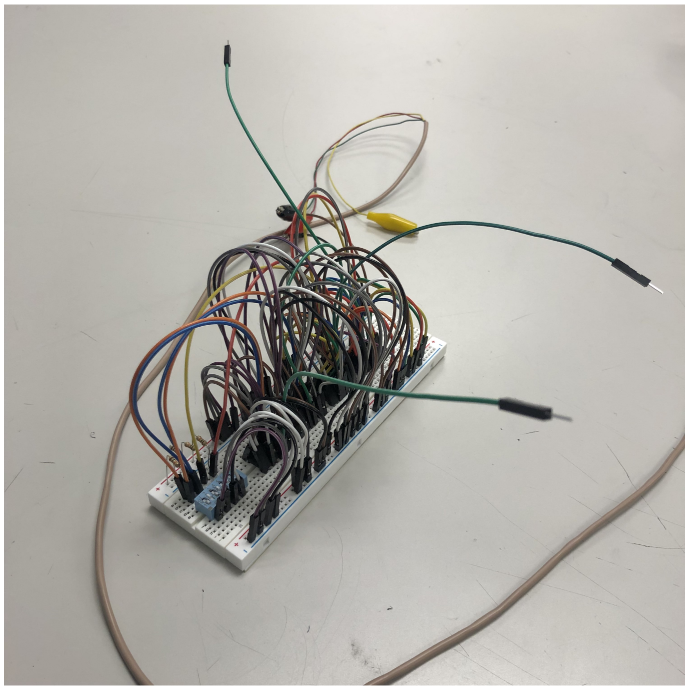

Figure 4: Final circuit on breadboard

Next, I set up the multiplexer corresponding to the addition of the least significant bits, M_0 and N_0. The inputs A and B were connected to N_0 and M_0, respectively. The output R_0 represents the value of the least significant bit, and I had connected this pin to an LED, according to the circuit in **Figure 5**, to easily visualize the change in voltage.

Figure 5: LED circuit

After making all the necessary connections for the first multiplexer, I tested it by setting the DIP switch *on* for both inputs, N_0 and M_0, which corresponds to low logic levels for both inputs. As can be seen in **Table 1**, this should produce a low output. However, the LED was on. My plan for debugging was to check that the input voltages were correct, and then check the wiring. The input voltages were fine, but some of the wiring was wrong. More specifically, I had forgotten to leave the W output free, and so the $\bar{G}$ and GND wires were shifted up one pin. After that fix, all combinations of the inputs produced the correct output.

I then set up the multiplexer corresponding to the carry out of least significant bit. Again, the inputs A and B were connected to N_0 and M_0, respectively. The output C_{out0} represents the carry out of the least significant bit, and I had connected this pin to an LED, according to the circuit in **Figure 5**, to easily visualize the change in voltage. After making all the necessary connections, I tested it, but the LED would not light up for any combination of the inputs. I immediately checked the orientation of the LED, and sure enough, it was the wrong way around.

Next, I connected the multiplexer corresponding to the second bit. The inputs A, B, and C were connected to C_{out0}, N_1, and M_1, respectively. The output R_1 represents the second bit, and I had connected this pin to an LED. After making all the necessary connections, I tested it with the inputs $M_1 = 0$, $N_1 = 0$, and $C_{out0} = 1$, which should yield a high output as can be seen in **Table 3**. However, the LED was off. I checked the voltages of the inputs, and it didn't take me long to realize that the power supply wasn't connected. Embarrassing.

Finally, I incorporated the last multiplexer corresponding to the carry out of the second bit. Again, the inputs A, B, and C were connected to C_{out0}, N_1, and M_1, respectively. The output R_2 represents the carry out of the second bit, and I had connected this pin to an LED. After making all the necessary connections, I tested it with the inputs $M_1 = 0$, $N_1 = 1$, and $C_{out0} = 1$, which should yield a high output as can be seen in **Table 4**. However, the LED was off. I checked the voltages of the inputs, and they were correct. After staring at the wiring for some time, I saw that the wire connecting the output to the LED was somewhat loose. I pushed it further into the breadboard, and everything worked.

In order to test the functionality of the complete circuit, I checked the voltages of the three outputs R_0, R_1, and R_2 with a multimeter for various combinations of the inputs. As I mentioned earlier, I could have connected each output to an LED to easily visualize the changes in voltage, but instead, I chose to measure the voltage of the output directly. The three unconnected green wires shown in **Figure 4** correspond to the three standard outputs.

Equation	M_0	N_0	M_1	N_1	Answer	R_0	R_1	R_2
00 + 00	0	0	0	0	000	0	0	0
01 + 00	1	0	0	0	001	1	0	0
01 + 01	1	1	0	0	010	0	1	0
10 + 01	0	1	1	0	011	1	1	0
10 + 10	0	0	1	1	100	0	0	1
01 + 10	1	0	0	1	011	1	1	0
11 + 01	1	1	1	0	100	0	0	1
11 + 10	1	0	1	1	101	1	0	1
11 + 11	1	1	1	1	110	0	1	1

Table 5: Test results

3 Testing

The results of the tests conducted on the final circuit are shown in **Table 5**. The first column shows the mathematical equation we are implementing in binary. For example, the equation in the second-to-last row is $11+10$ which corresponds to $3+2$ in base-10. The first number is M and the second number is N, and so $M_0 = 1$, $M_1 = 1$, $N_0 = 0$, and $N_1 = 1$. The circuit produces the result 101 in binary, which is 5 in base-10.

4 Reflection

The most valuable lesson from this lab was the importance of testing your circuit as you build it. It is clear from the description of my debugging struggles that I would have encountered a lot of problems if I had built the entire circuit in one go. However, since I tested the circuit after incorporating each multiplexer, I was able to quickly identify the bugs associated with each multiplexer implementation. After I was certain that one multiplexer was functioning as it should, I knew that any bugs I encountered while working on the next multiplexer had to be associated with that new multiplexer. Thus, for every new multiplexer, my debugging was limited to testing its inputs and outputs.

A skill I need to refine is methodically debugging. Although I didn't spend a great deal of time on any of the bugs I encountered, I could have solved these problems faster had I created some sort of methodology. The issues I had were as follows: incorrect wiring, wrong orientation of the LED, disconnected power supply, and disconnected wires. All of these issues are quite general and will likely happen again in the future. Since some are easier to check than others – for example, checking that the power supply is connected is much easier than checking if all the wiring is correct – it would be more efficient to check them in a certain order. When I was trying to debug the problem I was having with the third multiplexer, I checked the voltages of all the inputs before I realized that the power supply was disconnected. I would have saved time if I had initially checked the connection of the power supply. Thus, in order to debug more efficiently, I'll create a list of common bugs and order them from least to most time-consuming. As of now, that list is:

- Check connection of power supply
- Check orientation of circuit elements
- Check wiring
- Check connection of wires

Although checking the wiring is arguably more tedious than checking that the wires are properly connected to the breadboard, the latter is less common of a problem and so should be left to the end. This lab took around two hours to complete.

List of Tables