Full Adder Report

Kiran Nadkarni

CSE2300W – 005L

Elizabeth

9/30/19

Objectives:

Our goal with this lab is to combine the elements of logic involving addition using signed magnitude and 2's compliment to create a circuit in the LogicWorks software that is able to add both positive and negative hex numbers and calculate whether there is overflow in the circuit. This lab tests understanding of half and full adders, as well as being able to add both positive and negative numbers and displaying values between +7 and -8 through the use of the hex display to discern the value of the answer, and two binary probes to display the sign of the answer and if there is overflow respectively.

Introduction:

This lab does not focus too much on different number systems, as we are only really using hex and binary to understand 2's complement. We want to mainly focus on the idea of simply adding two numbers together, whether either of them is negative or positive. As mentioned before, the concepts we want to address are those involved in the addition of binary numbers, signed magnitude and 2's complement. Both of these are ideas presented to represent a binary number as a negative number. Since we are using 4-bit binary numbers, an overview of signed magnitude would be to say that the most significant bit of the binary number is a "sign bit" which determines the sign of the entire number. A 1 MSB represents a negative number and a 0 means positive. The 2's complement method of displaying negative numbers is a little more involved, where a positive number is represented the same as it would be in binary, but a negative number is created by inverting all of the bits of the positive version of that number (ex: -4 becomes 4, and -9 becomes 9), and inverting them, then adding a binary 1 (ex: 3 in binary is 0011, which is then inverted to 1100 and 1 is added, so finally 1101).

Within LogicWorks we are going to be making use of a variety of components to create our circuit. The first major component we are using is the hex keyboard, which allows us to select a single value from the 16 single bit values of the hex numeral system, from 0-F. This allows easy selection of input numbers to add, so we don't have to mess around with many binary switches to create a single number. We also want to use XOR gates to invert all of the bits, which is essentially making our input negative shown by 2's complement. LogicWorks also has 4bit adder components built in, which have two 4-bit inputs a carry-in, carry-out, and a 4-bit output. We are going to use these to handle the addition between bits, because they can do simple binary addition. The hex display and binary probes were mentioned before, and the major components we will need are some NOT, AND, and OR gates to manage our overflow detection system.

Procedure:

First Adder-

1- This circuit is fairly simple to create and understand, we want to use this to demonstrate understanding of the components we will be using and converting added numbers from binary to hex. The first step is to get out the three hex keyboards we will be using in this circuit. We will also use two adders, with two hex keyboards

taking up the inputs A0-A3 and B0-B3. The output for that adder will be attached to the input for the next adder, with the third hex keyboard also being attached to that input.

2- The final step is to attach the output for this cascaded adder to the hex display, which will display the result in single bit hex. It is important to understand that this does not account for overflow and does not support adding negative numbers, which the second adder circuit will do. The adders also do not need anything attached to the carry-out ports and the carry-in inputs have to be grounded.

Second Adder-

3- I started off this lab by placing the two hex keyboards and giving each one of them a binary switch next to them. Then I sent each of the four hex keyboard outputs to a XOR gate each, 4 in total. The other input of those XOR gates were connected to the binary switch that I paired with each hex keyboard, to fulfill the condition for the XOR gates to make the hex keyboard input negative at the flip of a switch. The four outputs to the XOR gates are connected to the adder now, at four of the eight inputs, B3-B0. The other four inputs are grounded because we are just using these two adders as a stage to carry in the value from the binary switches. The carry-in spot of these adders is attached to the respective binary switch of the adder and hex keyboard, because we are "carrying in" the sign bit from the switch.

4- The outputs S3-S0 of the two adders we have are now being routed into a second stage adder, which will do the actual addition calculation. This means that all eight inputs of this second stage adder are being used, and the four outputs are being routed into another set of XOR gates. We do this because the hex display cannot correctly read the 2's complement form of negative numbers, so we have to switch our answer back into readable signed magnitude form after the second adder adds the inputs together.

5- The output of this adder is where we put our first binary probe to determine the sign of the answer in signed magnitude. We implement the probe right before the previously mentioned XOR gates on the MSB because the MSB is the bit that determines sign. The second input of all of the XOR gates are all chained together again, and connected to the MSB of their input because this is the bit that determines their sign in 2's complement.

6- The outputs of the XOR gates feed into the B3-B0 of another adder, where the A3-A1 inputs and the carry-in are grounded and the A0 input is connected to the second inputs of the previous row of XOR gates that were chained together and connected to the MSB. The reason we do this is because converting to signed magnitude from 2's complement requires us to add a binary 1 to our reinverted answer again, and we do this automatically using this system. Now we can just connect final adder to the hex display since there are four inputs and four outputs and we move on to our last step.

7- The last step of this design is the overflow detector, which connects into the pre-existing parts of this circuit. I want to work backwards and start by putting in the

binary probe that will determine a 0 for no overflow and a 1 for overflow. We then connect this to the output of an OR gate, because we want separate conditions to display the 1 for confirming overflow. Both inputs of the OR gate connect to the outputs of 3-input AND gates because there are multiple conditions that have to be met to confirm overflow. We want three NOT gates to be after these AND gates, where one of the outputs of the NOT gates is connected to the input of one of the AND gates and the outputs of the other two NOT gates are connected to two of the inputs of the second AND gate. The NOT gate which is alone has its input connecting to the MSB output S3 of the second phase adder from step 3. The other two inputs of the AND gate that this NOT gate is attached to will connect to the inputs of the other two NOT gates, and then these each of these AND inputs from the first AND gate connect to the two binary switches from step 1. The final AND gate input that is unattached to anything will now be attached to the same second stage adder output MSB as the previously mentioned binary probe. This final overflow detector makes sense because the combination of input conditions for the AND gate accounts for any addition of signs of positive or negative numbers we might have. The locations that we attach these gates to are determined by what determines the signs of the input numbers (the switches) and what determines the sign of the result (MSB of result).

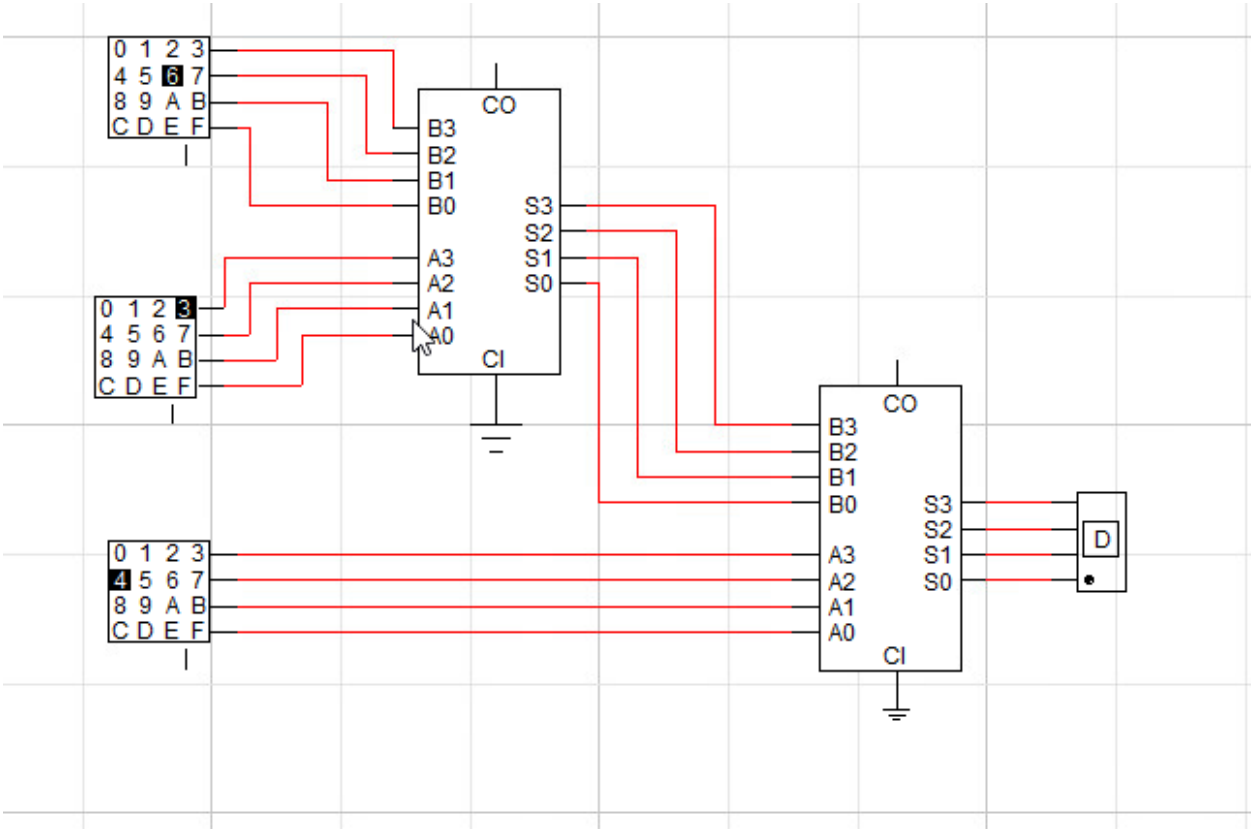

Figure 2: This is the diagram of the first adder circuit where we have three hex keyboards that are attached to the two cascaded 4-bit adders, resulting in a single bit hex display.

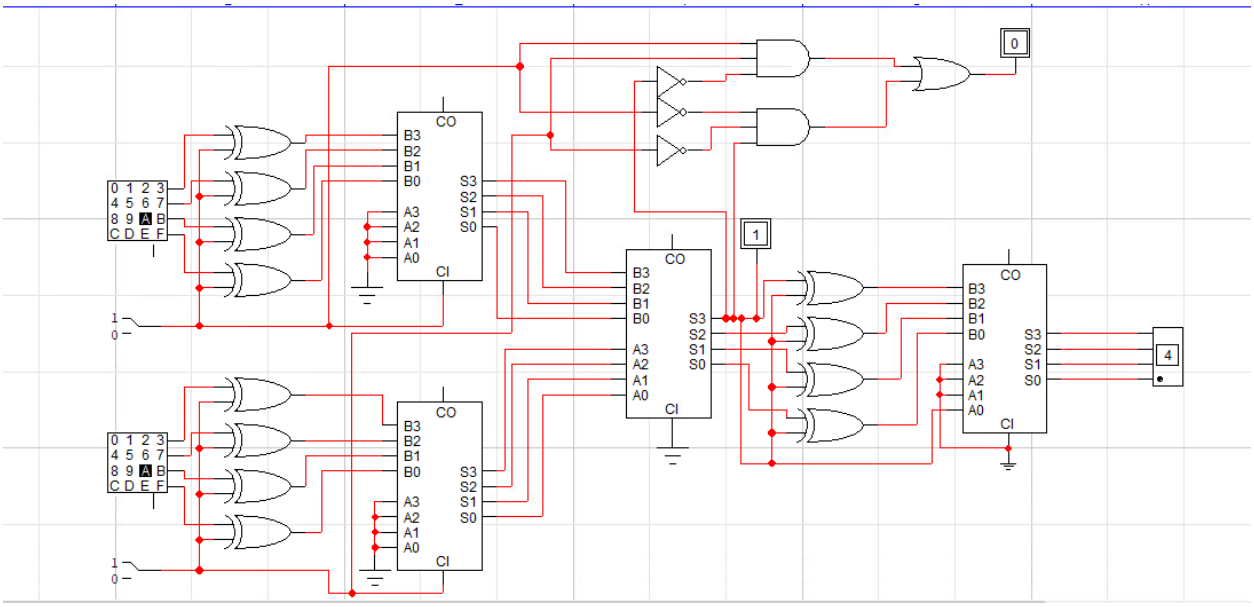

Figure 3: This is the second binary adder circuit. Note the two hex keyboard inputs, the two binary switches for number sign, and the two binary probes, one for overflow detection and one for sign detection of the input.

Results:

Circuit 1-

First Num	Second Num	Third Num		Result
0	0	0		0
1	2	3		6
3	4	5		C
5	5	5		F
5	6	7		2

Figure 3: These are the results from the first circuit, notice the case that exceeds a sum of 15 essentially takes the value it should be and subtracts 16 from it because we are working with hex. There are very few test cases here because there is only one boundary.

Circuit 2-

First Num (with sign)	Second Num (with sign)		Result (with sign)	Overflow
2	3		5	0
-5	-2		-7	0
3	-2		1	0
-6	4		-2	0
3	4		7	0
-5	-3		-8	0
6	5		-5	1
-7	-4		5	1
10	-2		-8	0
-11	2		7	0

Figure 4: This is the data from the second circuit. All of the results abide by the boundary conditions of the sum of the two numbers needing to be between -8 and 7.

Discussion:

The results of this lab were accurate and the design of the circuits are both correct. After a fair amount of debugging, I am sure all of the data from these circuits reflects the parameters set forth at the beginning of the lab. Something important to note is that even if overflow is not displayed for incorrect answers sometimes in circuit 2, the circuit is still working as intended because it is only able to display a sum between 7 and -8. I also think that learning concepts in the lectures previous to this lab helped immensely with the construction of the circuit, because so many of the components like the hex keyboard and the adder make it so there are less components that take up space and distract from the complete product.

Conclusion:

The most difficult part of this lab was definitely understanding some of the concepts of the second circuit. The first circuit was fairly straightforward, but it had many limitations such as not being able to use negative numbers or not having any way to display overflow. The second circuit resolved many of these issues, but required many more components to make it work. There are slight variations to the implementation of the second circuit that could work, such as using less wires and using less gates for the overflow detector. In totality however, I think that this lab did solidify my knowledge of the concepts we used to create it, such as 2’s complement, overflow in binary addition, and basic LogicWorks circuit design.

Questions:

We decide which tests to make for the circuit because both of the circuits have their own restrictions in terms of what they can add. The first circuit for example can only add positive numbers and will experience overflow with any number past 15 because that is the last number with a single bit hex representation in F. Testing number outside of the restriction of summing to 15 is good to display what overflow cases display, and then using numbers with a sum below 15 is good to show some other results. For the second circuit, it's a good idea to add a mix of positive and negative numbers, so I want to try to add all 4 combinations of positive and negative numbers, and then also display four overflow cases outside of the restrictions being between the sum of -8 and 7, one of adding each combination of positive and negative numbers. I also want to tack on the two boundary cases here, which would be numbers that sum to those 7 and -8 boundary values for a total of 10 test cases.

A serial adder introduces the factor of time into adding these two 4-bit binary numbers. Serial adders would use less gates/components than the adder we implemented here, but it would reuse those gates to calculate the value one bit at a time. This results in a slower output, which we have never had to deal with in this class so far.