Lab Assignment 6

Jack Fenton
fenton.j@husky.neu.edu
Nicholas Abadir
abadir.n@husky.neu.edu

Submit date: 3/22/2020
Due Date: 3/22/2020

Abstract

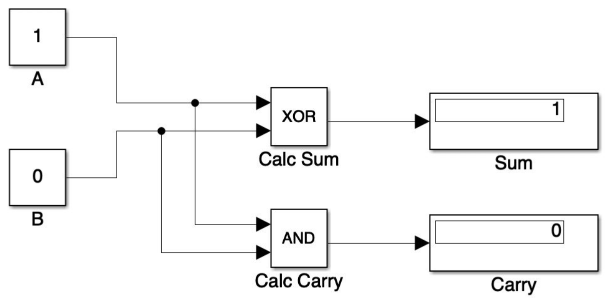
In this lab we were introduced to a new program called Simulink in which we are to create logic circuits through figures that then create the code to act like the circuit that is described in the figure. We also practiced logic circuit construction so that we are able to create more advanced circuits in the future.

Introduction

The goal of this lab was to introduce Simulink and use Simulink to create logic circuits. The advantage of Simulink is that it creates the code in matlab for each of the figures as seen in the images below allowing us to understand both in logic and as code (which are essentially the same thing but different). This lab also introduces circuits beyond the basic AND and OR gates plus their variations so that in the future we can create more complex circuits.

Lab Discussion

In the prelab, we were asked to create a half adder in Simulink which can be seen below.



This Half adder takes two inputs and puts them through an AND and a XOR gate and spits out their two outputs in the display objects seen above.

Half Adder

A	B	S	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

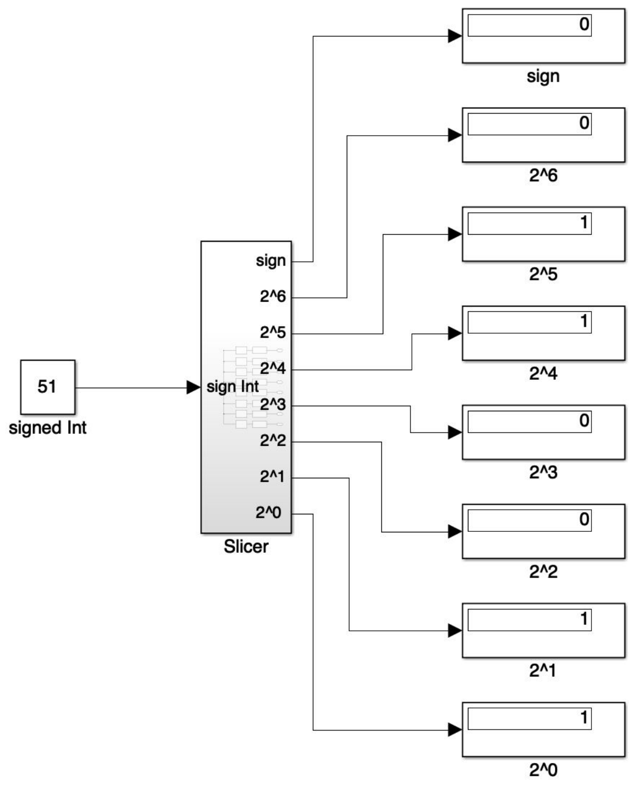
Full Adder

A	B	C_{in}	S	C_{out}
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

This lab used less components than previous due to extenuating circumstances so the only components we used were our computers and Simulink through MATLAB.

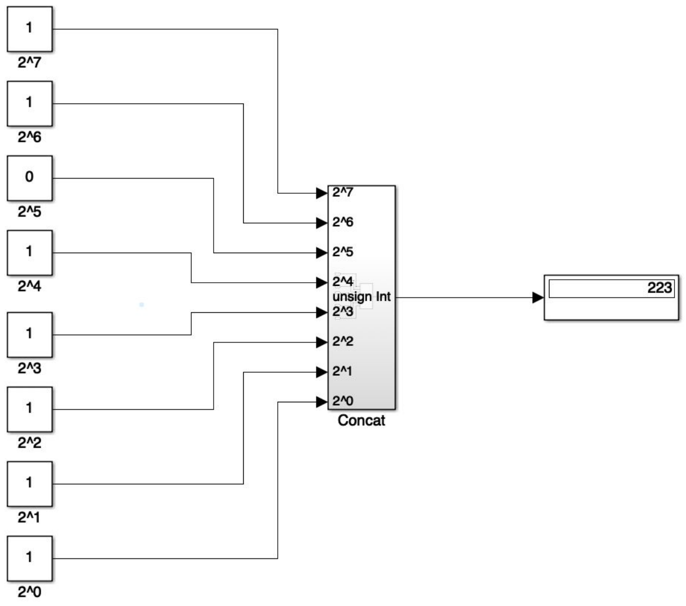
Results and Analysis

Lab 6.2 Testing the Design



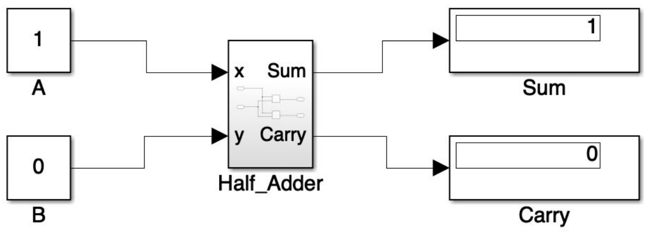
The figure shows the slicer we were required to make for assignment 1 with example number 51. The setting of each slicer was the same as that described in the lab so in ascending order the first bit was set MSB to 0 and LSB to 0 and so on. Then the data is converted through the slicer from base 10 to binary.

Lab 6.3 Merging multiple bits into a bus



The figure above represents a bit COMCAsT which performs the opposite function of the slicer in which it takes 8 single bit inputs to their respective place and outputs a base 10 number. MSB and LSB were set up in the same fashion.

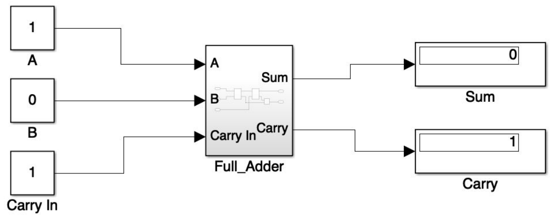
Lab 6.4 The half adder



A	B	S	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

For this assignment we simply took the half adder created in the pre-lab and packed into a logic block. This allowed us to use the block later in the program. When one and only one of the inputs is 1, then the sum is 1. When both of the inputs are 1, the carry out is 1, which is effectively 2.

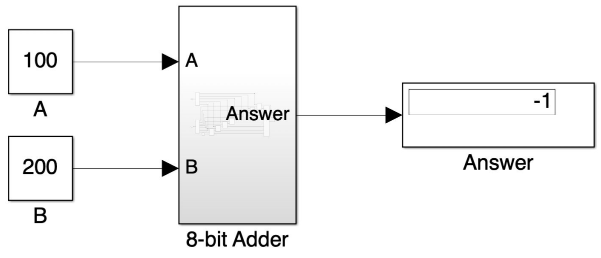
Lab 6.5 The full adder

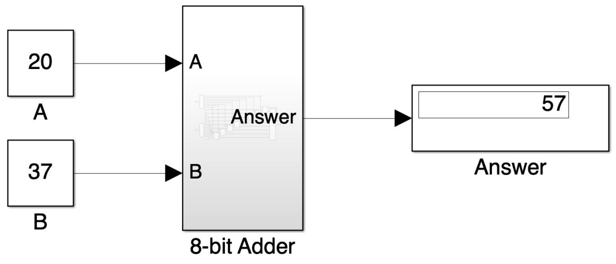


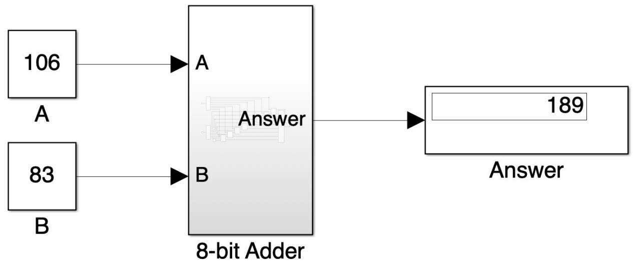
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

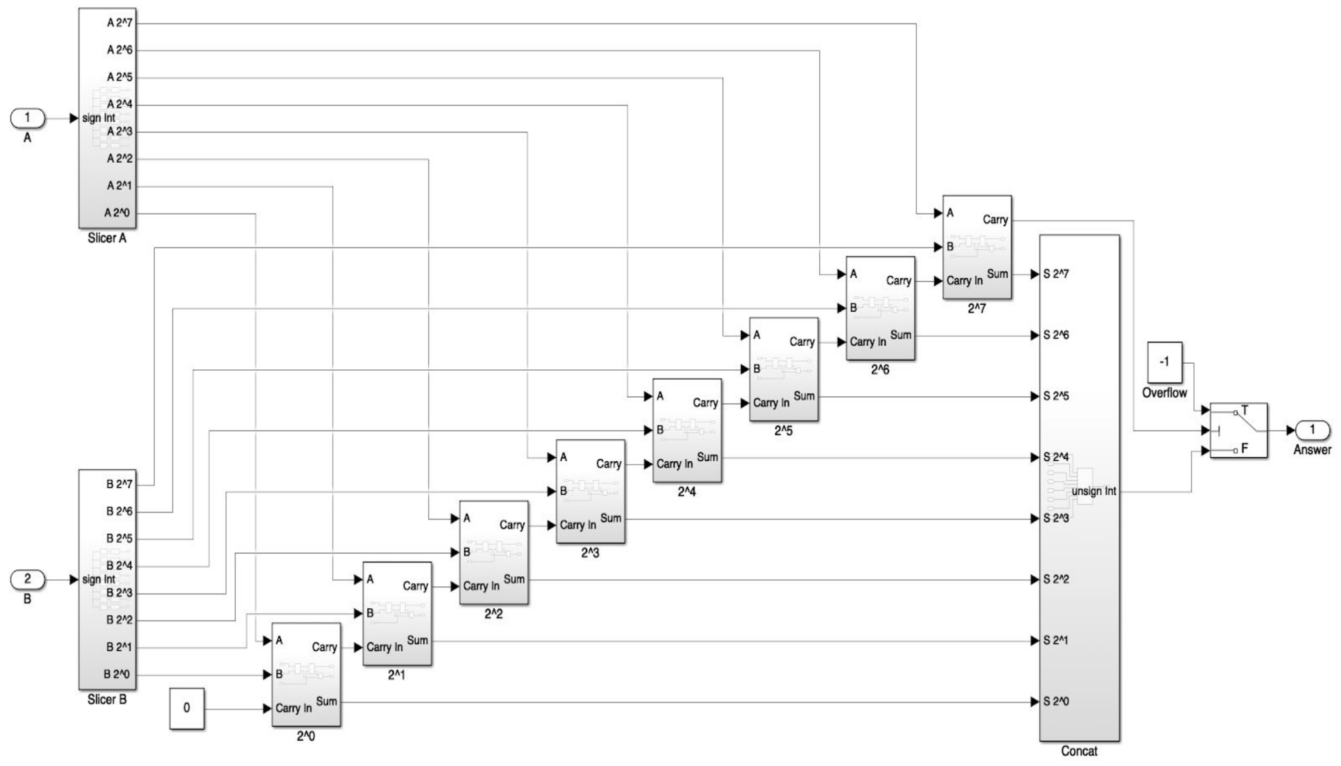
Then using the half adder blocks we created, we then constructed the Full Adder block based on the diagram in the lab. This block has 3 inputs and 2 outputs which outputs up to a sum of three or 11_2. A full adder is effectively a 3 input half adder, but to make this with circuit elements, a series of two different half adders, where 2 inputs go into one and that output and the third input go into the second. If either of the carrys are 1, then the resultant carry is also 1.

Lab 6.6 The 8-bit binary adder









Contrary to the instructions given for the lab, the 2 slicers used in the 8-bit adder had to be set to uint8 numbers, as the Bit Concat component in Matlab can only return unsigned integer values. If the slicers were not changed from int8 to uint8, the circuit would be incorrect. For the circuit created, A, B, and Answer have a range of [0,255]. A and B were both split into their bit components using slicers, the outputs of which were labeled in the form A 2^3, meaning the bit representing 2^3 for A. The individual bits were then combined using a Ripple Carry adder, where the first carry in was 0 and each subsequent carry in was the last adder's carry out. This was performed for bits 0-7, but the 7th adder's carry out gave whether or not there had been an overflow. The Sum components from each are combined in the Bit Concat block. The output is conditional on whether or not there has been an overflow, which is evaluated by the switch case. If the final carry is a 0, then there has not been an overflow and the number calculated is output. If there has been an overflow, then -1 is output by the circuit, signifying that there has been an overflow. Three sets of inputs are shown: 20+37=57, 106+83=189, and 100+200=-1. The third is an example of an output when there is an overflow.

Conclusion

This lab was a short introduction to Simulink and through that the construction of circuits. Some changes were made as the Concat could not take signed binary numbers and would cause a data mismatch so we were forced to switch to uint8 data type from Assignment 2 onwards. As well, the labeling of the outputs were arbitrary and misplaced so we relabeled them A 2^0 and so on. This lab will allow us to expand on our logic circuits so that we can create more complex circuits in the future. While logic circuits hold no place in many real world applications explicitly, they are behind every computer and code and understanding them allows us to manipulate both hardware and software much better.