Lab 1
Brendan Bovenschen A20323473, Landon Fox A20319340

Section 1: Introduction

For any logic design to function properly, rigorous and extensive testing must be completed. For our purposes, the tool that conducts these processes is the testbench. While most labs in this course will focus on the design and implementation of a logic system, our work on this project was focused on the testbench and the functions inside of it. Unlike in our handwritten work, the logic implemented in this circuit is far more complicated due to the increased number of variables, inputs, and outputs. Since the dataset produced by this design is extremely large for human interpretation, it's impractical to manually report each combination of input and output to determine if the system is working as desired. This is where the functions of the testbench are implemented.

The testbench is part of every lab in this course as well as a useful service for all digital logic applications. To experience the range of processes the testbench contains, we were first tasked with creating a 4-bit adder containing a carry-in and a 5-bit output. This was then tested using the testbench and implemented to display through 4 hex digits (as seen in Figures 2.1 - 2.3). We also created a system within the testbench to test our design along with our expected result with a large number of random inputs. At the end of our work, we were able to provide a substantial data set that proved that our design acted as the lab instructions intended as well as a tangible model which properly reacts to various physical stimuli.

Section 2: Baseline Design

Before we began creating our ripple carry adder, we first moved our initial 1-bit full adder (Figure 3) from our first lab to our new workspace. Initially, we created a 2-bit adder comprised of 2 full adders to properly map the outputs of one group to another. Once this was performing as expected, we expanded the system to 4 bits for each input (Figure 4). Since our design is a combinational ripple carry adder, each segment is not synced with the other, unlike a parallel adder. This means that for a small segment of time, the device will display incorrectly as it performs the math of each full adder. Although this would not be noticeable by any user, it could affect other logic which relies on this circuit. To see if our design would output the correct sum, we implemented a waveform graph of the two 4-bit inputs (A and B), a 1-bit carry-in (Cin), and the 5-bit output sum (Sum). In the first example from Figure 1, the hex numbers 04, 02, and 00 were added together to amount to 06 as anticipated. On our physical board, we controlled our inputs with a set of 8 switches (4 for each input number) and a button for our carry-in along with four 7 segment displays for the data.

Section 3: Detailed Design

For our 4-bit ripple carry adder, the two inputs can be viewed as a 4-bit binary number (A, and B). We also included a 1-bit carry-in, as well as a 5-bit output sum (S). Each bit of our input arrays corresponds to a full adder within our system where the addition of those bits will occur with the carry-in being a part of the first full adder. As seen in Figure 4, the bits A[0], B[0], and Cin (carry-in) are added together through the boolean algebra within the first full adder. The outputs from this component are S[0] (the value of the sum in the 0th digit) and C[0] (the first carry out). This process ripples through each component in the design until the last full adder. The very last carry out is stored as the most significant bit of the sum. For our board implementation, the file top_demo.sv contains the controls for the input and output. A and B are determined by switches 7:4 and 3:0 and Cin is controlled by button 0. To display this, our data was run through a 7-segment display driver to produce 3 hex numbers (being A, B, and Sum). Since Sum is a 5-bit packed array, it requires 2 digits of the display. This system can be observed in Figures 2.1-2.3.

Section 4: Testing Strategy

To test the ripple carry adder (RCA), we created a loop to generate random values for the numbers to be added by the RCA. When testing the results, we first looked at the waveform generated by the simulation running off of the .do file (Figure 1) observing the results to make sure the RCA is working as expected. Next, using the test

bench we saved the three numbers being added (A B, and Cin), the sum, correct sum, and the comparison of the sum and correct sum to a unique line in the file labeled rca.out (Figure 5) that shows the aforementioned variables and, most importantly, the comparison of the sum and correct sum ensuring us of the RCA's functionality. The sum produced by our design is compared to the unpacked array Sum_corr which is created from the sum of A, B, and Cin through the logic of SystemVerilog. With this value as a control, the testbench determines if this value is the same as Sum and outputs the boolean equivalency value at the end of each line. The final test we administered was the implementation of the seven-segment displays on the physical board (Figures 2.1-2.3) which also proved the RCA's functionality. In Figure 2.2, the A value of 0101 (hex 5) and the B value of 0011 (hex 3) are determined by the series of switches near the bottom of the board. In this case, the Cin value (controlled by button 0) is 0. The values of A and B are then represented in hexadecimal format in positions 3 and 2 on the display. Then, the value of Sum produced by the RCA is sent to the last two digits of the display. Through these results, we can determine the accuracy of the RCA with any input.

Section 5: Evaluation

In this lab, we furthered our knowledge of the workings and utilities of the test bench in SystemVerilog. By designing a ripple carry adder and testing it with over 150 vectors we were able to discover more about using random variables, for loops, and writing outputs to a file that allows us to better utilize the information from the digital logic. Not only did we begin to understand the syntax of a SystemVerilog testbench, but we were also able to utilize the many tools available to us to ensure the certainty of our design. To represent our input in output data, we applied logic systems as a series of arrays rather than many separate logic variables which greatly increased the readability and organization of our HDL system. The implementation of a for loop in conjunction with an output line for our data was crucial in the examination of our RCA. Understanding the basics of the testbench and being able to use and build upon these ideas will be needed for future labs in the course.

Figure 1

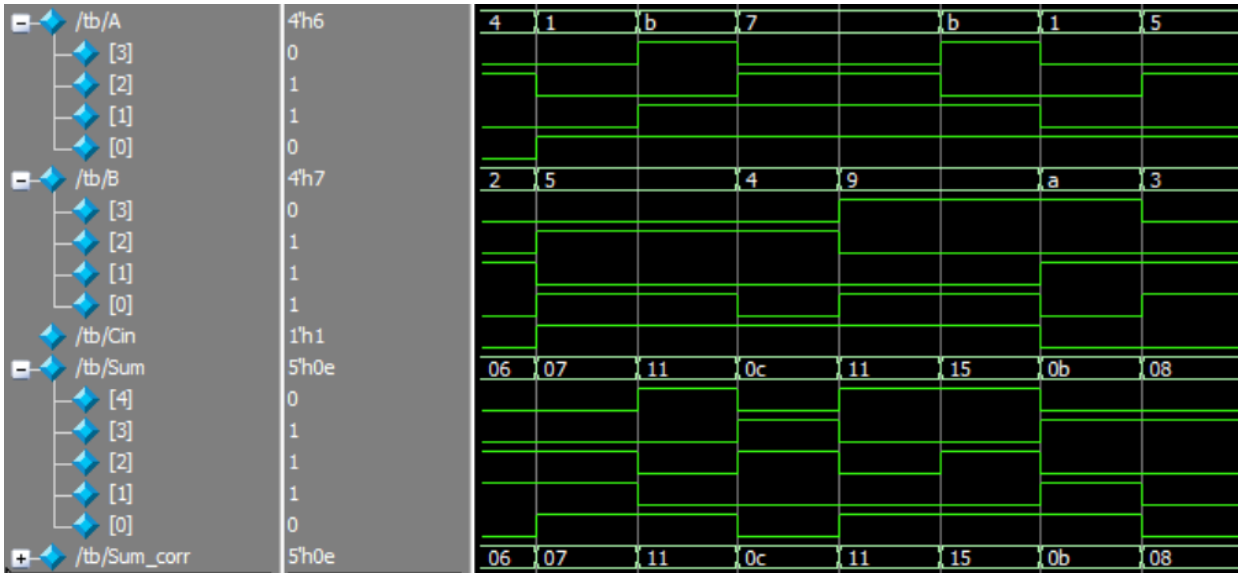
/tb/A
4'h6
[3]
0
[2]
1
[1]
1
[0]
0
/tb/B
4'h7
[3]
0
[2]
1
[1]
1
[0]
1
/tb/Cin
1'h1
/tb/Sum
5'h0e
[4]
0
[3]
1
[2]
1
[1]
1
[0]
0
/tb/Sum_corr
5'h0e
4
1
b
7
b
1
5
2
5
4
9
a
3
06
07
11
0c
11
15
0b
08
06
07
11
0c
11
15
0b
08

Figure 2.1

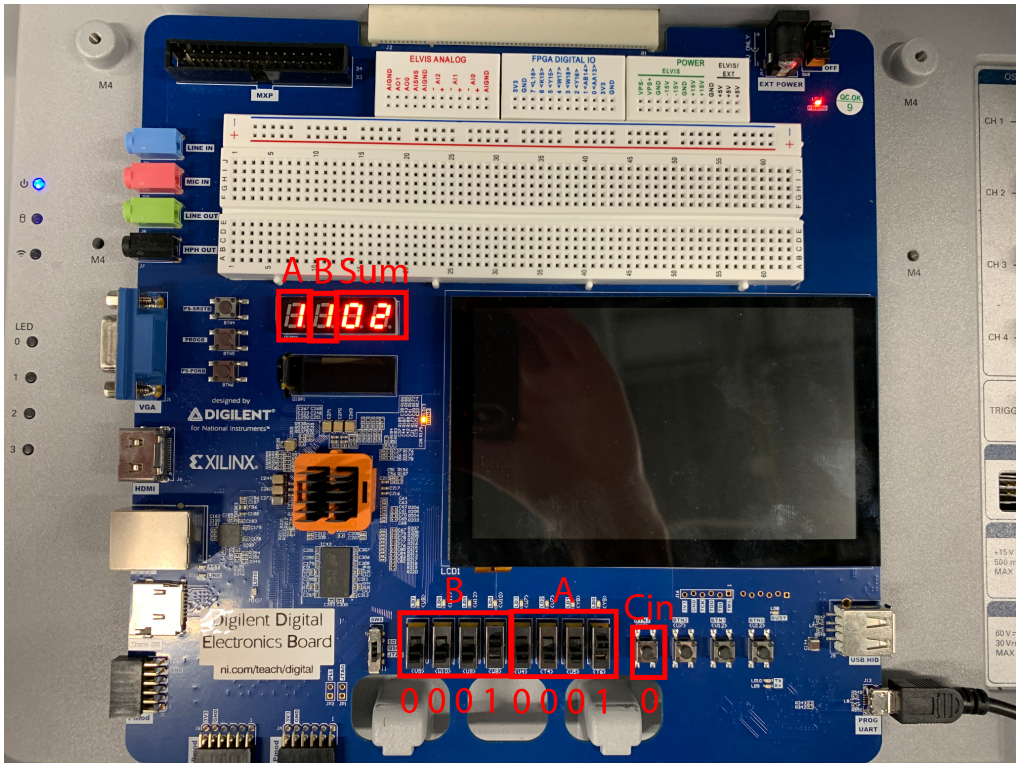

Figure 2.2

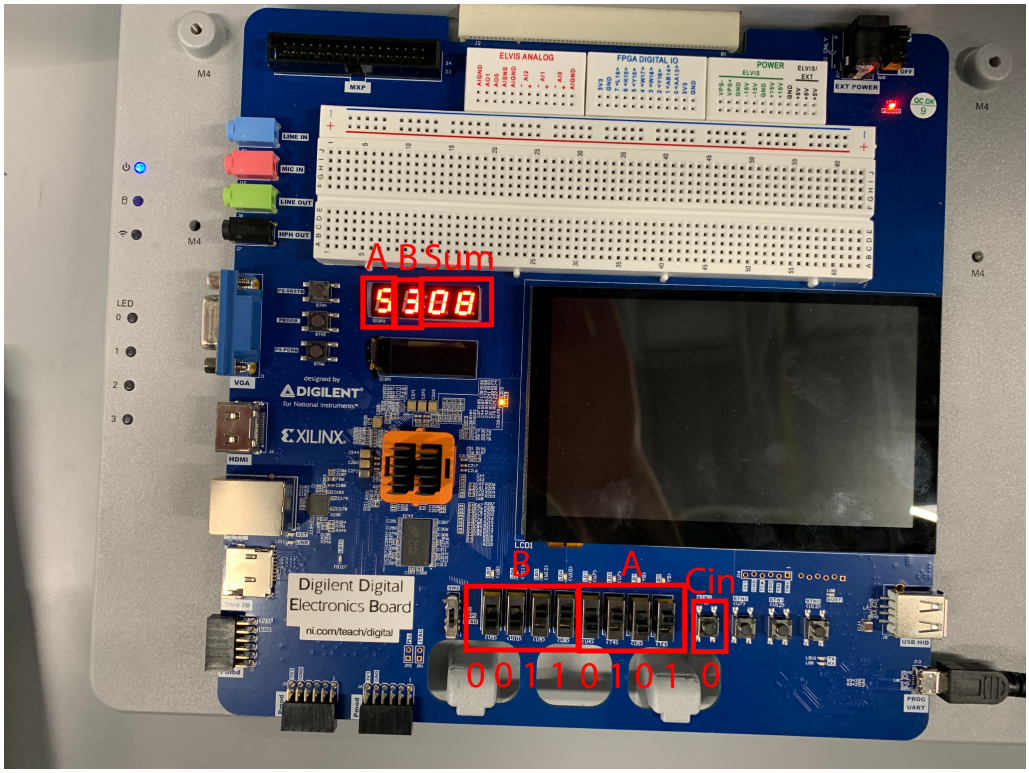

Figure 2.3

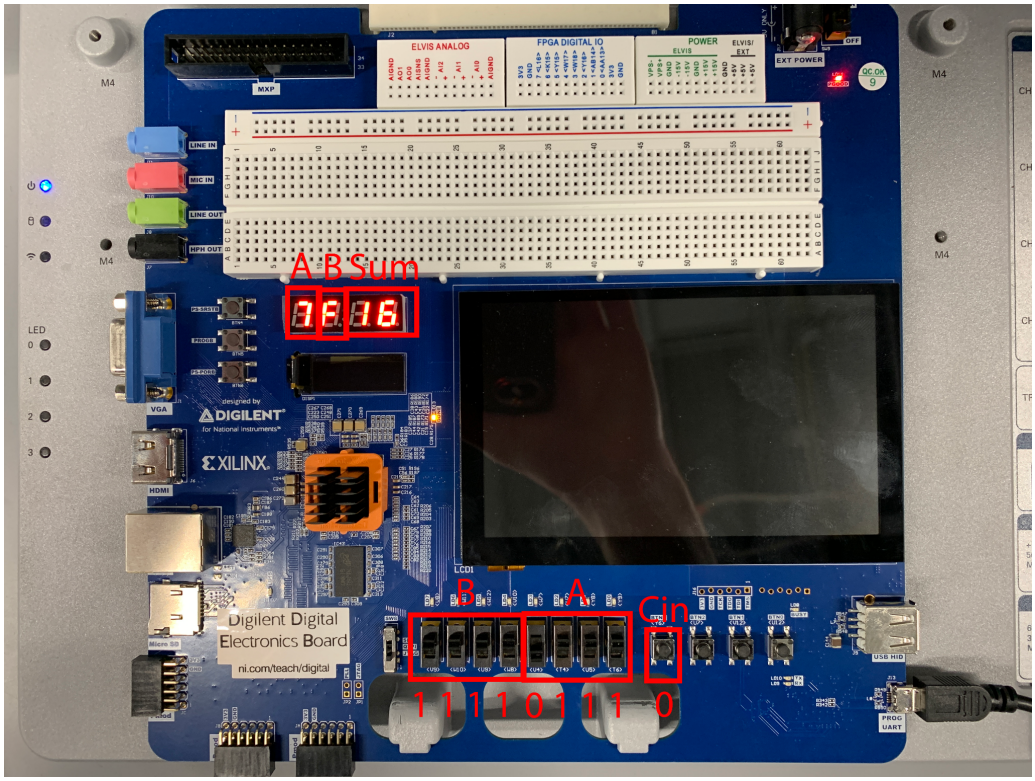

Figure 3

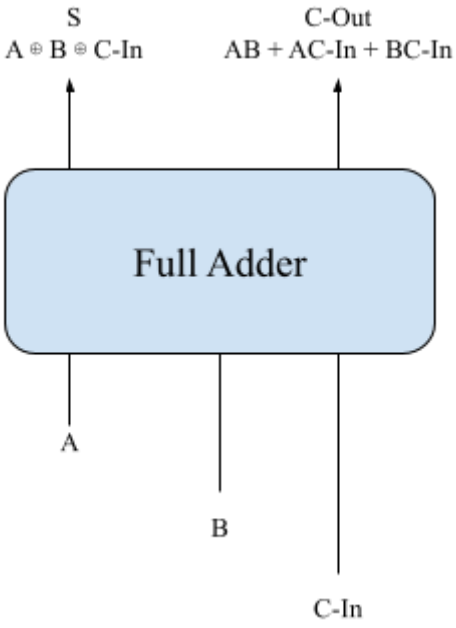

Figure 4

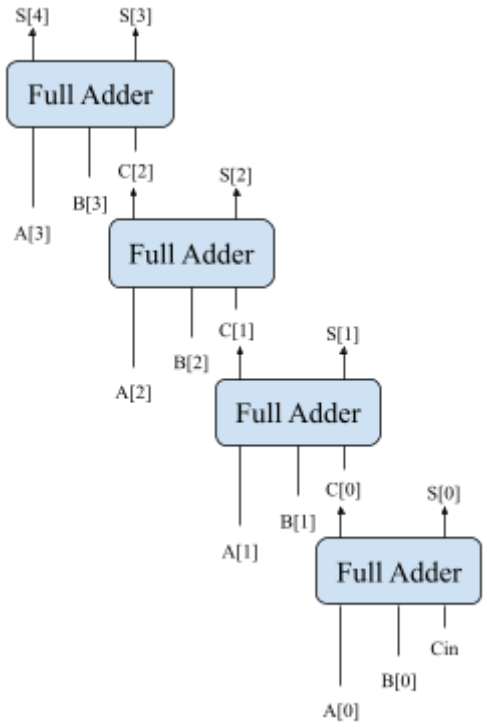

Figure 5

```
4 1 1 || 06 | 06 | 1
3 d 1 || 11 | 11 | 1
5 2 1 || 08 | 08 | 1
d 6 1 || 14 | 14 | 1
d c 1 || 1a | 1a | 1
6 5 0 || 0b | 0b | 1
5 7 0 || 0c | 0c | 1
f 2 0 || 11 | 11 | 1
8 5 0 || 0d | 0d | 1
d d 1 || 1b | 1b | 1
3 a 0 || 0d | 0d | 1
0 a 1 || 0b | 0b | 1
6 3 1 || 0a | 0a | 1
3 b 1 || 0f | 0f | 1
2 e 1 || 11 | 11 | 1
f 3 0 || 12 | 12 | 1
```