CompArch Lab 0

William Lu, Kathryn Hite, & Ian Hill

September 29, 2016

1 Introduction

In this lab, we implement a full 4-bit adder simulation. In order to explore the performance and timing of this circuit design, we then run the full adder module on an FPGA board and work through a series of test cases.

2 Full Adder Design

The goal of the full adder design is to add together two binary numbers, A and B, with each of these strings containing four individual bits. When producing the resulting sum, we also return the carryout and overflow bits to determine the validity of the result.

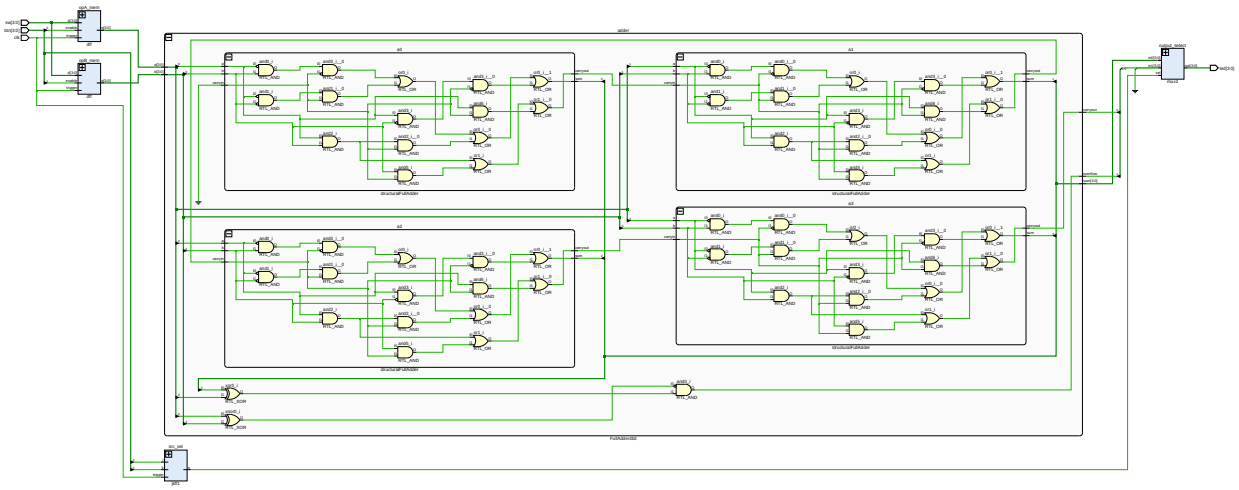

Figure 1: Circuit schematic generated in Vivado of our full 4-bit adder.

3 Testing

Because exhaustively testing our four bit adder would require 256 test cases, we chose tests to cover all four possible carryout and overflow scenarios. These scenarios and the test cases we chose are in Table 1 on the following page.

Table 1: Four Bit Adder Test Cases

Test Inputs		Test Results			Expected Test Results		
A	**B**	**Sum**	**Carryout**	**Overflow**	**Sum**	**Carryout**	**Overflow**
0000	0000	0000	0	0	0000	0	0
0001	0001	0010	0	0	0010	0	0
0011	0100	0111	0	0	0111	0	0
0001	1001	1010	0	0	1010	0	0
1100	1100	1000	1	0	1000	1	0
1111	1111	1110	1	0	1110	1	0
0110	1111	0101	1	0	0101	1	0
0100	1100	0000	1	0	0000	1	0
0100	0100	1000	0	1	1000	0	1
0111	0111	1110	0	1	1110	0	1
0110	0111	1110	0	1	1110	0	1
0101	0110	1011	0	1	1011	0	1
1000	1000	0000	1	1	0000	1	1
1000	1111	0111	1	1	0111	1	1
1010	1001	0011	1	1	0011	1	1
1011	1010	0101	1	1	0101	1	1

3.1 Test Case Motivation

For each scenario, we chose test cases that covered the extremes of the scenario as well as cases that tested normal operation within that scenario. For example, in the scenario covering no carryout and no overflow, we tested $0_{10} + 0_{10}$ and $4_{10} + 3_{10}$.

In creating test cases, we also discovered certain patterns within scenarios. Overflow occurs when you add two positive or two negative numbers together and the sum is the opposite sign of the augend and addend. You can guarantee overflow by adding two negative numbers together because the MSB of any negative number in two's complement will be 1, which guarantees that the MSB of the sum will be 0, making the sum positive, while ensuring there will be a carryout of 1.

3.2 Test Case Failures

The adder was developed in two phases. We first implemented the adder without considering overflow to verify the basic circuit. This version would therefore fail all but our first four test cases, which do not produce carryout or overflow. After implementing the second portion of the adder deign, all of the test cases ran successfully.

4 Performance

4.1 Onboard FPGA Testing

To verify the performance of the full adder on the FPGA board, we utilized a test case where input A, input B, sum, and the carryout/overflow would all display as different bit patterns on the LEDs.

The test case shown in Figure 2, Figure 4 on the following page, and Figure 4 on the next page is the addition of inputs A and B. A is equal to b1010, which represents d-6 in the 2's compliment syste. The value of B is b1011, or d11. Adding these two numbers produces the sum b0101, which is d5. We chose this test case due to the fact that all of the inputs and outputs are different bitstrings, which allowed us to verify that the system was changing the LED sequence when it was supposed to. This case also required both carryout and overflow, which displayed as shown in Figure 4 on the following page.

The process of computing the sum consisted of first inputting A using switches 0 to 3 and pressing button 0 to set the value of A. After changing the switches to value of B, we then pressed button 1, and the sum displayed on LEDs 0 through 3. Button 3 produced the carryout and overflow display, and button 3 returns the LED pattern to the sum.

On comparing the test case performance to Table 1 on the previous page, we see that the sum, carryout, and overflow values produced by the module running on the FPGA board match those of the test case given our values of A and B.

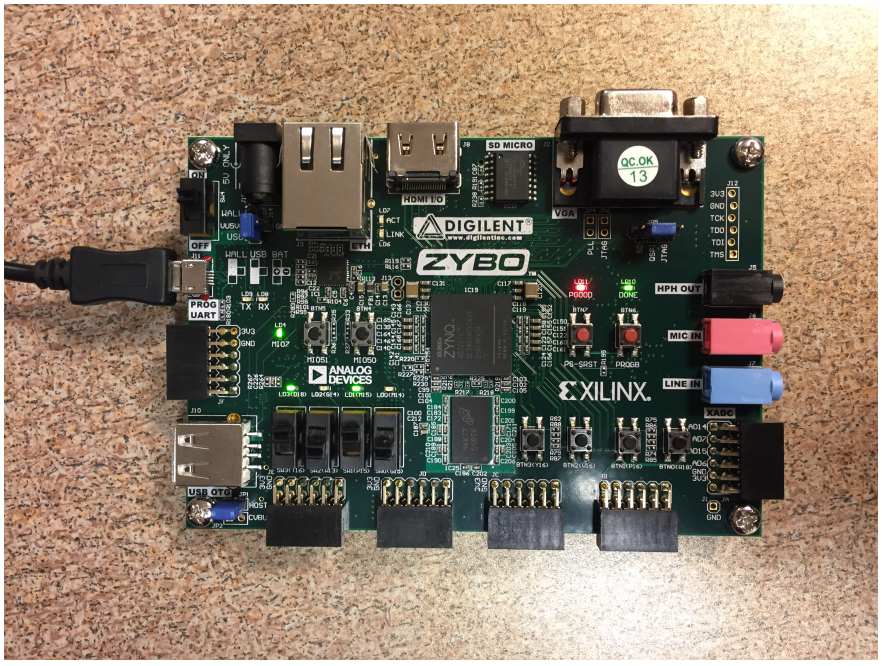

Figure 2: The value of A was input using switches 0 to 3, giving b1010 or d10. Button 0 initialized the input once the switches had been set.

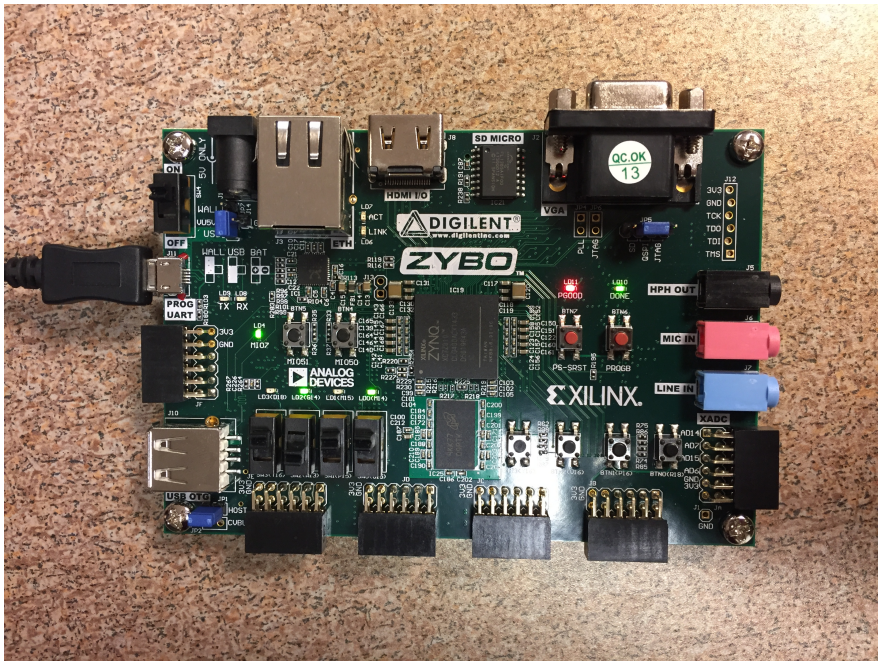

Figure 3: The value of B was also input using switches 0 to 3. The number b1011, or d-6 in 2's compliment, was given, and the sum appeared on LEDs 0 through 1 after button 1 was pressed to set the value of B. The shown sum is d0101, or 5.

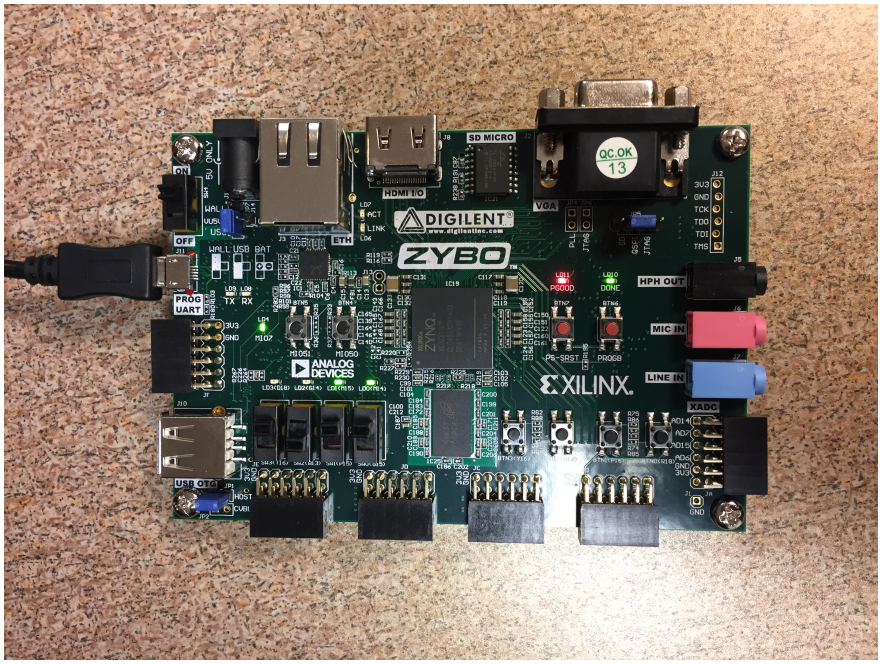

Figure 4: Carryout and overflow are displayed on pressing button 3.

4.2 Waveform and Delay Analysis

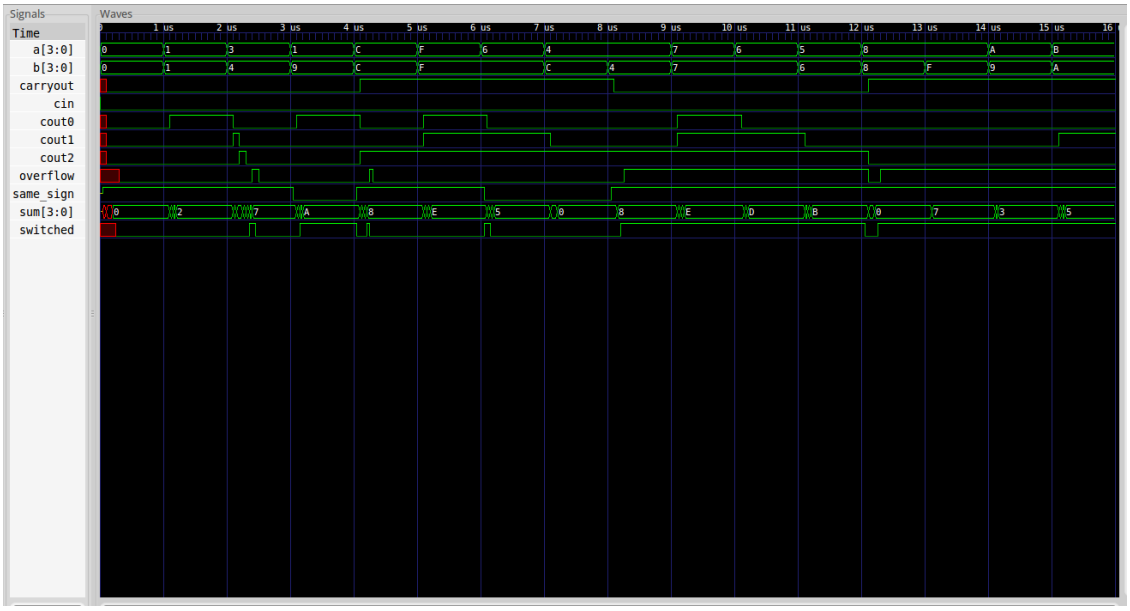

Figure 5: Waveform output from the full 4-bit adder testbench.

Using GTKWave, we generated a waveform and delay analysis using the structural full 4-bit adder we wrote in Verilog. The waveform very clearly shows the cascading effect that is present in the circuit, as you can see the propagation of bits from cout0 to cout1 to cout2. In the process, you can also see that the sum wire rapidly switches between values as bits propagate through the four individual adders within our circuit. There are also temporary fluctuations in overflow and switched before the sum wire stabilizes on a value after all bits finish propagating through our circuit.

5 Summary Statistics of Synthesized Design

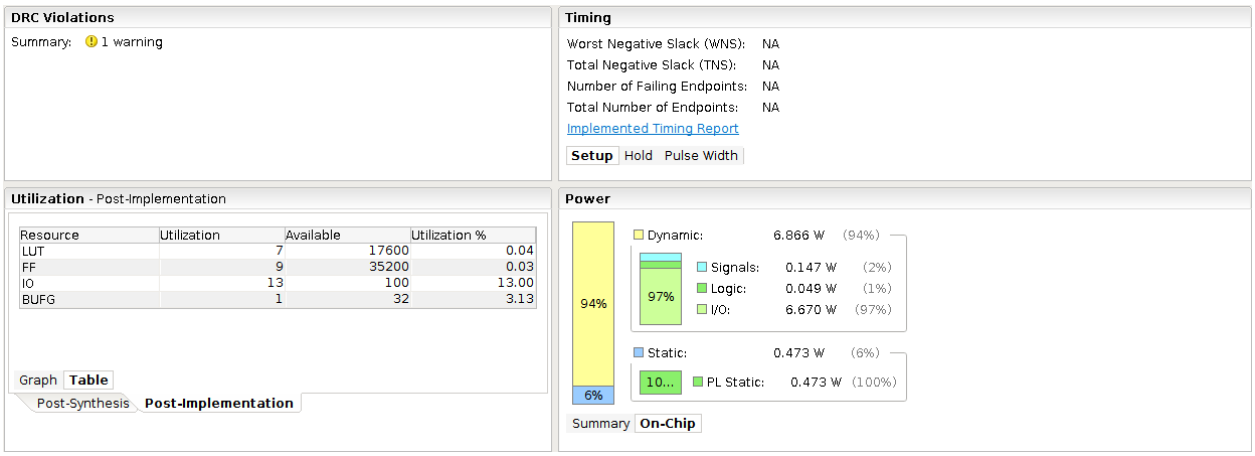

Figure 6: Performance statistics post-implementation on the FPGA exported by Vivado.

As seen in Figure 6, when we implemented our full 4-bit adder on the FPGA, it used 7 look up tables, 9 flip flops, 13 I/O elements, and 1 global buffer. Because the adder does not have a clock, we cannot evaluate the timing performance of our adder. Our adder also uses little power compared to board I/Os.