Lab Assignment 1
Introduction to Quartus Schematic
Binary Adders Using Simulation/Intel DE1-SoC Board

Zhenming Yang, Connor McEleney

yang.zhenm@northeastern.edu, mceleney.c@northeastern.edu

Instructor: Prof. Zagieboylo

d.zagieboylo@northeastern.edu

Due Date: September 21st, 2023

Submit Date: September 20th, 2023

Abstract

In this lab a four-bit adder was constructed using the Quartus Prime Schematic tool to simulate the behavior of the circuit and upload it to the DE1-SoC to test the circuit.

Introduction

The DE1-SoC is an embedded computing device built around the Altera System-on-Chip (SoC) FPGA and an ARM processor running a Linux operating system [1]. This platform offers a wealth of hardware features, with our primary emphasis placed on the ARM processor (CPU) and the Cyclone V FPGA, both integral components of Altera's System-on-Chip (SoC), which serves as the central chip within the DE1-SoC. Furthermore, the DE1-SoC includes an extensive array of input/output (I/O) devices, encompassing LED lights, switches, buttons, and 7-segment displays, some of which we will utilize [2].

The objective of this lab is to craft a circuit capable of performing binary addition. We employed the Quartus Prime Schematic tool for simulating and validating the behavior of this circuit. This software facilitates the construction of intricate circuits using a hierarchical schematic design, subsequently enabling us to subject them to various synthetic inputs. Additionally, we will transfer our designs onto the FPGA situated on the DE1-SoC Board and validate these designs by interfacing with the board's Switches and LEDs.

Lab Setup

Pre-Lab

Half Adder Truth Table:

A	B	Sum	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Equipment

DE1-SoC:

- The DE1-SoC is a hardware design platform built around the Altera System-on-Chip (SoC) FPGA. The DE1-SoC is designed for experiments on computer organization and embedded systems. It includes embedded processors, memory, audio and video devices, and some simple I/O peripherals.

Results and Analysis

Results

Part 1: Half-Adder

In this section, we created a digital half-adder circuit on the Quartus Prime Lite and tested it both in **Simulation Waveform** and on the **DE1-SoC FPGA**. We first constructed the half-adder based on the instructions (Figure 1) by using the predefined "**And gates**", "**Or gates**", and "**Not**

gates". The gates were connected with each other with wires, and the two inputs are labeled as SW[0]/ SW[1] and outputs as Carry/Sum.

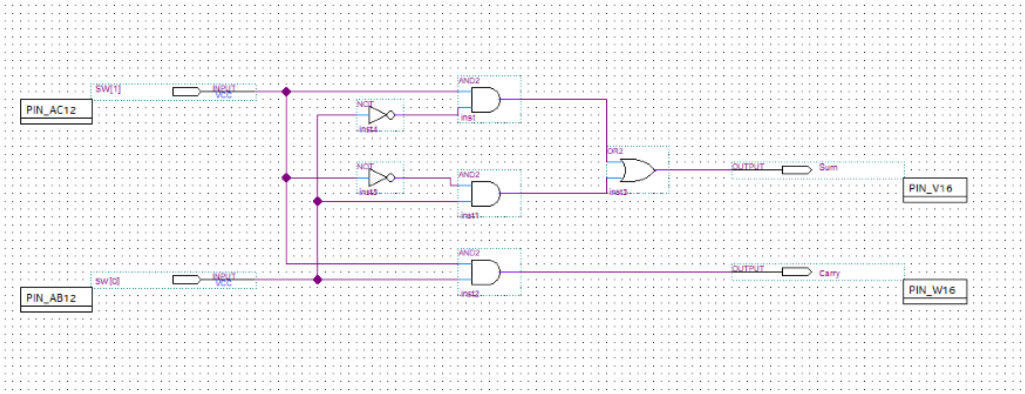

Figure 1: The Schematics of the Half-Adder. Noted the inputs and outputs are all configured with physical pins indicated next to their names.

We then created a simulation of our digital circuit on the Simulation Waveform Editor (Figure 2). We can see that when the input (SW[1] and SW[2]) are both 0, the Carry and Sum gives 0, while either one Input is 1 the Sum is 1, and when both Inputs are 1 the Sum is 0 and the Carry is 1. This result **directly corresponds to our truth table of a half adder** in our pre-lab section.

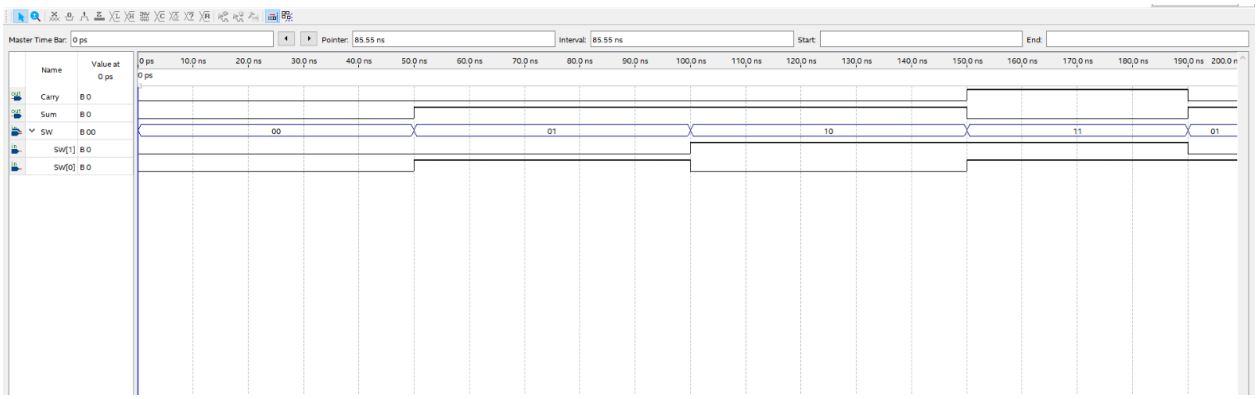

Figure 2: The Simulation Waveform result of a Half-Adder. Noted the change in the Sum and Carry output based on SW[1] and SW[2]

After the simulation, we implemented and tested our half-adder schematics on the DE1-SoC board. We first assigned each input and output to their corresponding pins for the switches and LEDs (see in Figure 1). And then we connected our DE1-SoC board to the computer and baked our circuits onto the FPGA (Figure 3). The image shows the state of the logic when SW[1] and SW[2] are both 0, And the output gives us 0 for Sum, and 0 for Carry, agreeing with the simulation and our truth table for a half-adder.

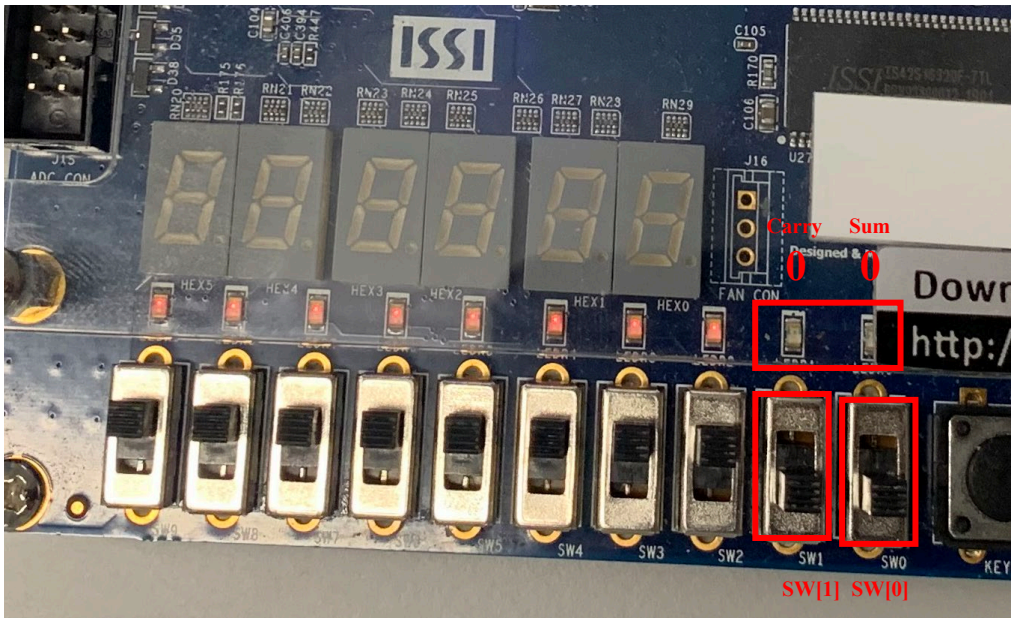

Figure 3: The Switches are both in 0 position and both the Sum and Carry are 0 to indicate the binary addition 0+0=0

We then turned both the switches on to see the Carry LED light up. The following image shows the result of the board showing the correct respond to **two 1s input with "10" as output**.

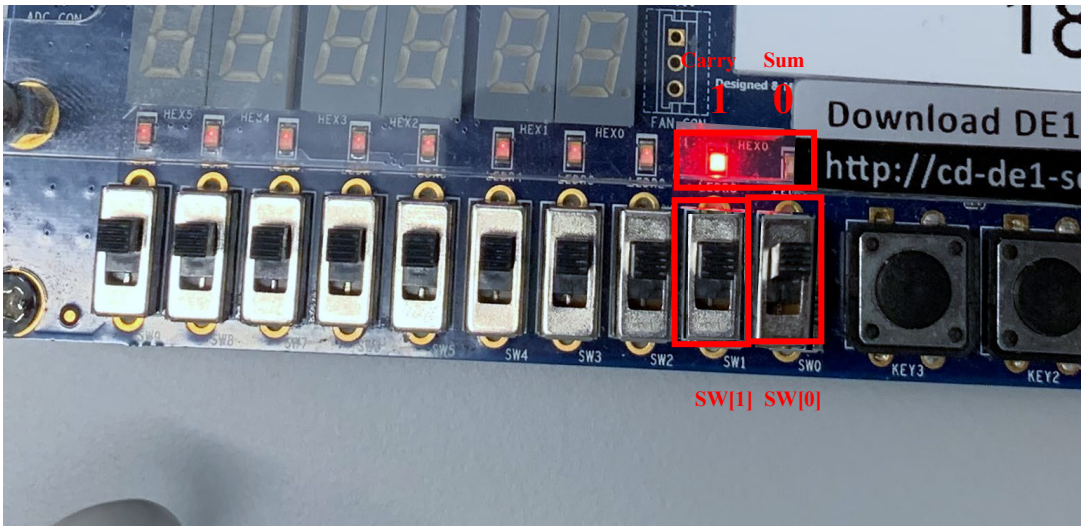

Figure 4: The Switches are both in 1 position and the Carry turns 1 and the Sum is 0 to indicate the binary addition 1+1=10

At last, we saved the half adder as a module for future usage in Part 2 and Part 3.

Part 2: Full-Adder

We first completed the truth table of a full adder to understand the logic of it and its connection to a half adder. We then created full-adder schematics on the Quartus Prime Lite and

tested it both in Simulation Waveform and on the DE1-SoC Board. By following the instructions, we created a full-adder schematic with **two half-adders** and an "**Or gate**" (Figure 4).

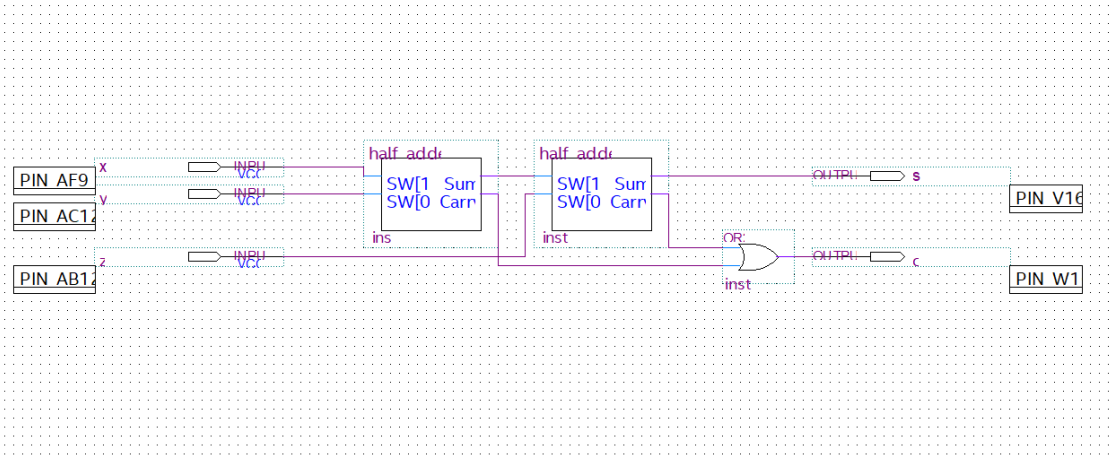

Figure 5: The schematics of a Full-Adder using two half-adders and or gate.

Then, we ran the Simulation and tested our logic by assigning the x, y, z inputs to different values. We can see that from the simulation (Figure 6). **The Sum is always 1 when only one input is 1. The Carry is 1 when at least two of the inputs are 1. And Carry and Sum are both 1 when all three inputs are 1s.** This proves our computation to be correct and corresponds to our truth table of a full adder.

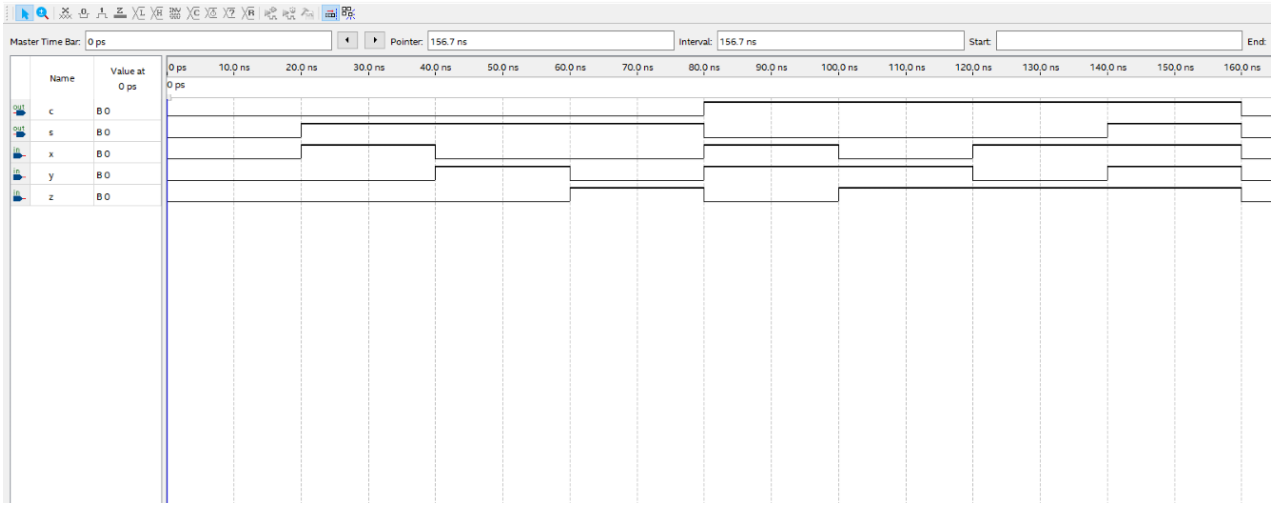

Figure 6: The simulation shows the correct output with the inputs from x, y, z.

We repeated the steps in Part 1 to bake our schematic to our Board and tested the switches with different positions. We first **only switch on one out of the three switches to get "01" as the respond** (Figure 7). And we then tested the scenario where **three switches are all on and get "11" as the output** (Figure 8).

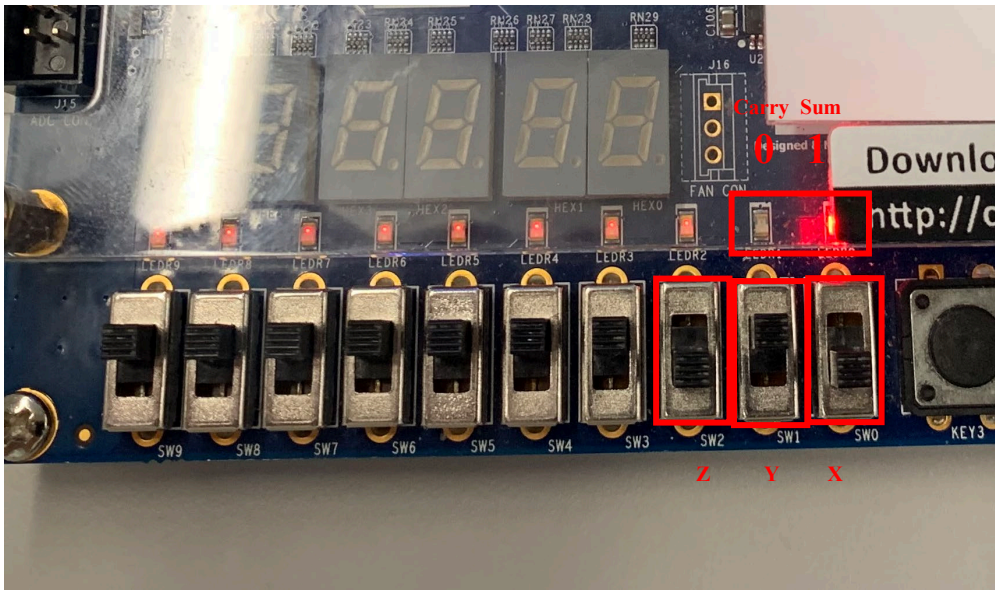

Figure 7: x switch is the only switch on, and we get the Sum to be 1 and Carry to be 0, indicating the binary addition 0+1+0=01

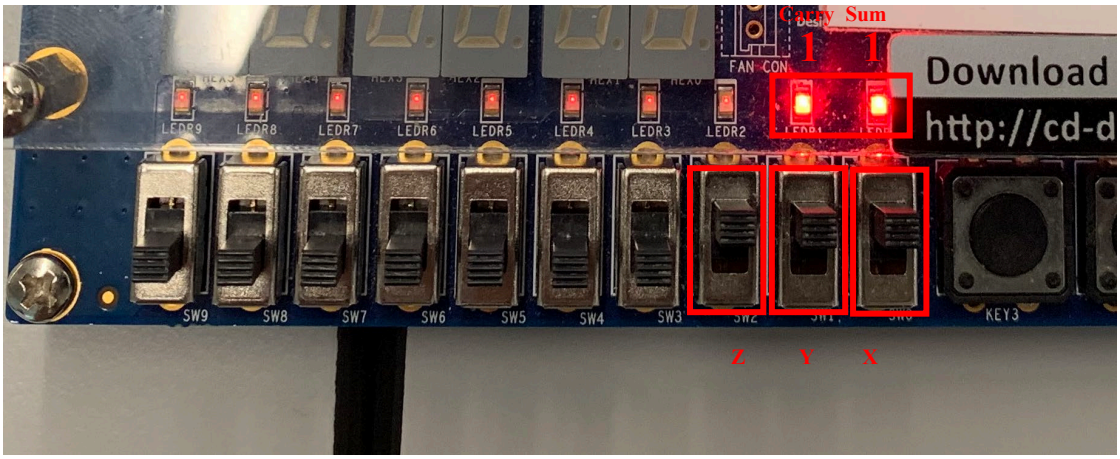

Figure 8: x, y, z switches are all on, and we get the Sum to be 1 and Carry to be 1, indicating the binary addition 1+1+1=11

We saved the full-adder to be used for our 4-Bit Adder in Part 3.

Part 3: 4-Bit Adder

To create a 4-Bit Adder, we first added four full-adders into our new project. We then connect the carry for each full-adder to the z input of the next one (except the first and the last full-adder). The first full-adder's z input is connected to ground and the carry for the last full adder is left empty for future changes. We then connect the four inputs for each number connect to their corresponding inputs (x and y) on the four full-adders, and the sum for each full-adder to the four-bit output (Figure 9). To implement the overflow logic in a two's complement adder. We **added an "XOR" gate to connect the carry from the last full-adder (which was left empty) and the**

second last full-adder (branching out). This logic detects overflow because the adder **only experiences overflow when two negative numbers or positive numbers add together and exceed the limit** which happens **only if last full-adder has a carry or only if the second last full-adder has a carry** (Figure 9).

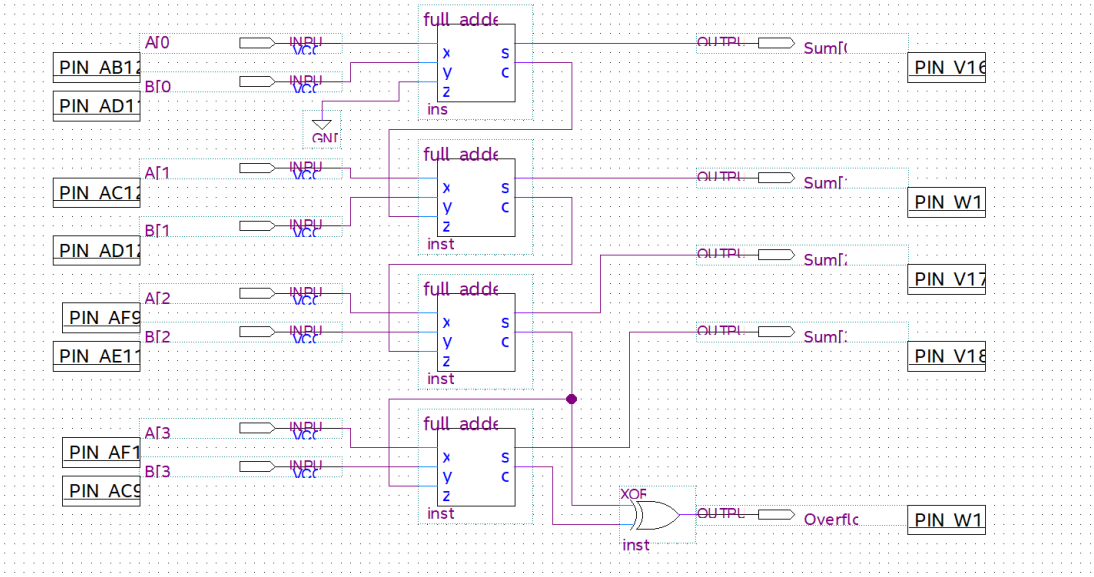

Figure 9: The schematics of a 4-Bit adder with an overflow detection

We then tested our digital circuit with the simulation tool and got our expected values. We can see the first case is when two negative numbers added together and caused an overflow and then follow with the case where two negative 1 added together and it didn't trigger overflow and gave us the right answer. The third case is two positive numbers cause an overflow and follows with not overflow. And the final case is one positive and one negative number added together and cause no overflow (Figure 10).

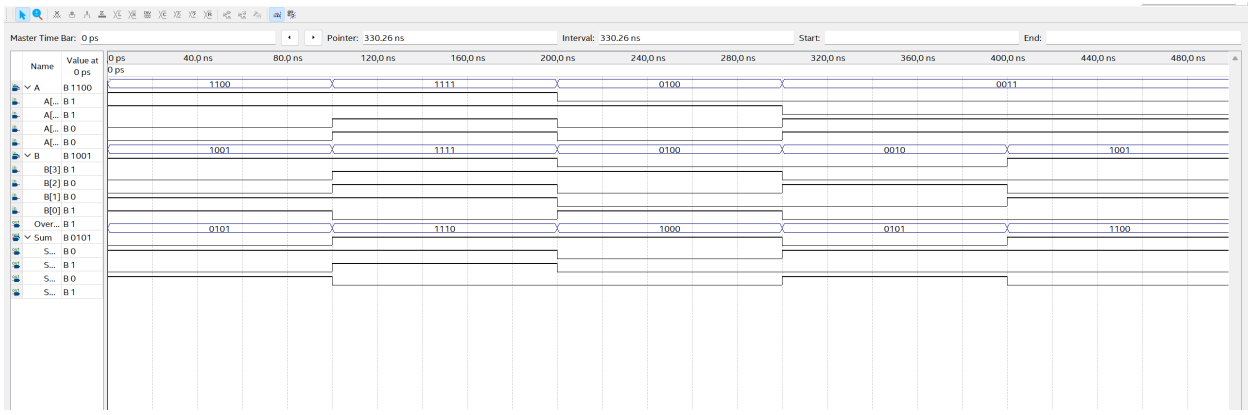

Figure 10: The simulation result of two four bits two's complement numbers added together and triggers overflow.

We tested our schematic on the DE1-SoC board. The following figures shows three scenarios: when **two positive numbers are added together and cause overflow** (Figure 11), when **two negative numbers added together and cause no overflow** (Figure 12), and when a **positive and negative number added together and cause no overflow** (Figure 13).

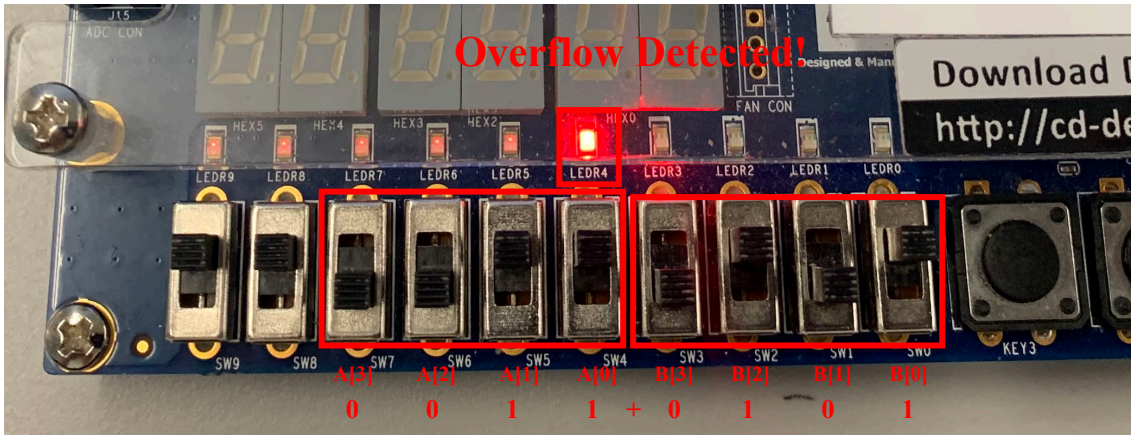

Figure 11: The two positive numbers added exceeds the largest number 4 bits two's complement number can represent and gives us the overflow error: 0011+0101=1000 which is false in two's complement (positive plus positive equals negative)

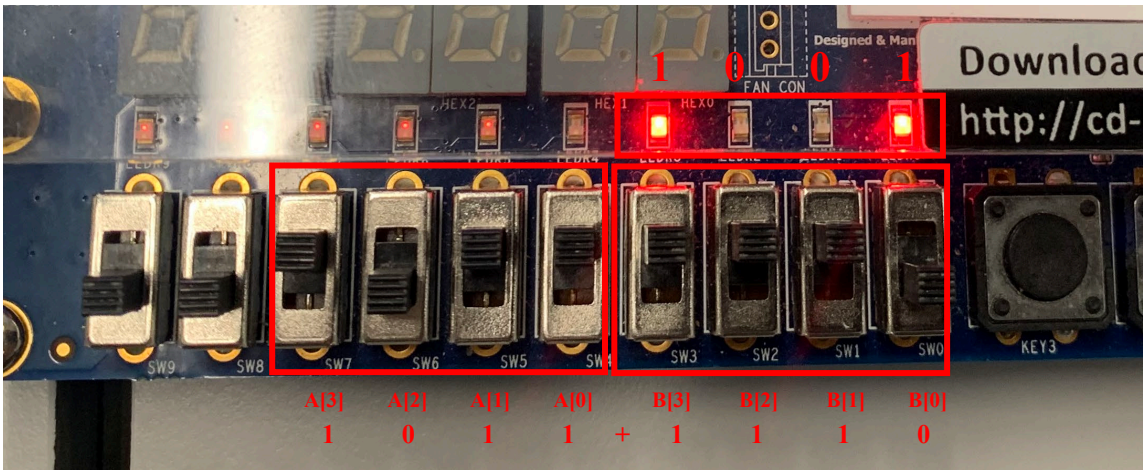

Figure 12: The switches are in position to represent the two negative numbers 1011 and 1110. The result shows the correct output 1001 and doesn't trigger overflow.

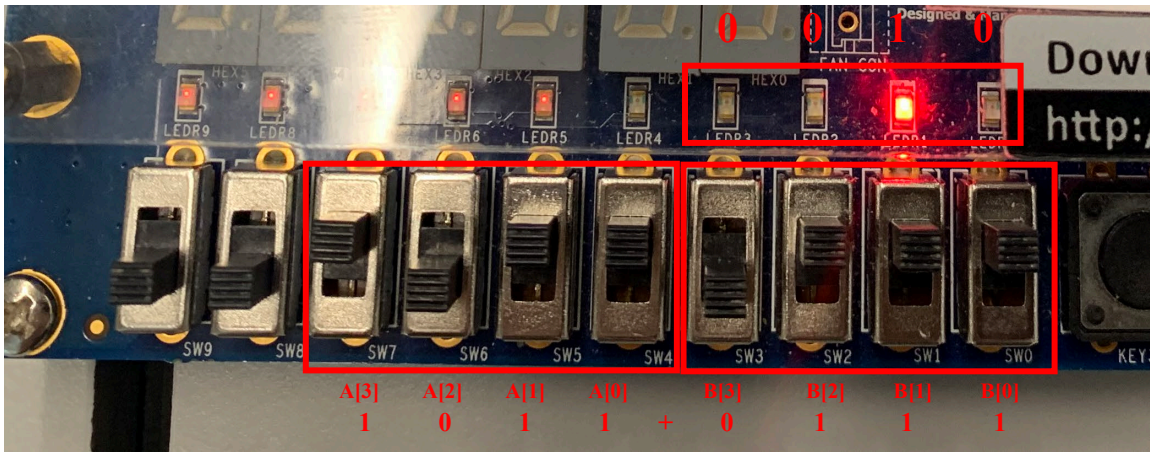

Figure 13: The switches are in position to represent the negative number 1011 and positive number 0111. The result shows the correct output 0010 and doesn't trigger overflow.

Analysis

The results of Part 1-3 all correspond to what we expected from our truth table and common logic. The overflow logic for the 4-bits adder was a bit complicated at first but we simplified it to just using the carry bits and an “XOR” gate (Figure 14).

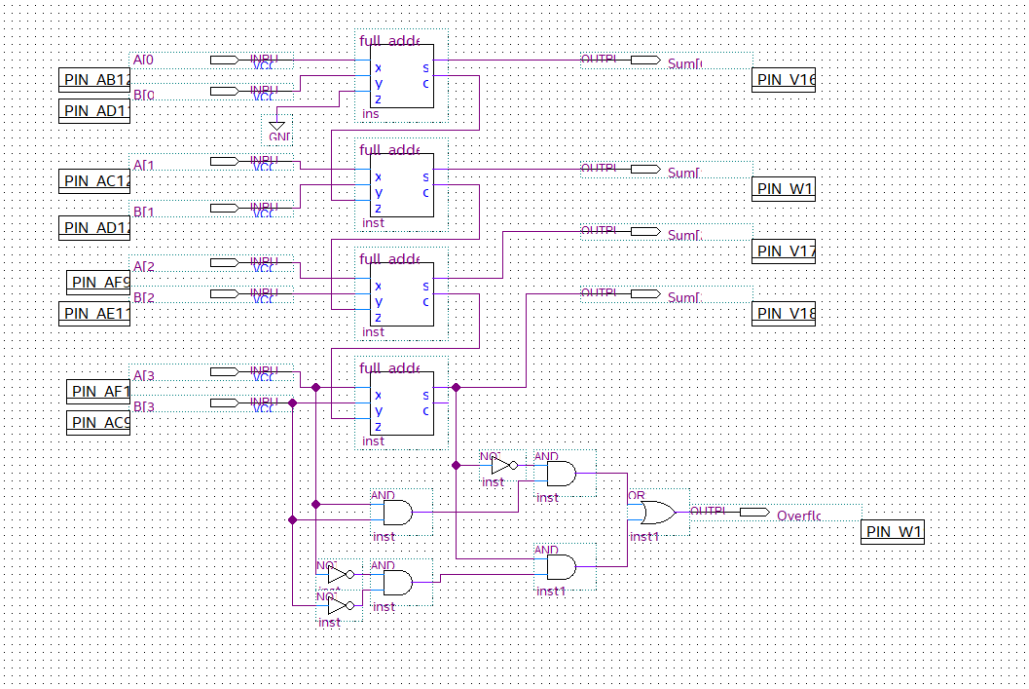

Figure 14: A not efficient way to check for the overflow on the adder. Complex logic is just a repetition of the full-adder logic and can be simplified as an "XOR" gate.

Conclusion

The experiments performed in this lab demonstrate the assembly of elementary logistic gate circuits into a more complex system to add numbers represented in binary two's complement. The experimental organization acted as a ladder process to assemble a four-bit adder as the final product, demonstrating fundamental principles of computational design. Rather than trying to construct the four-bit adder from raw logic gates, it is built in layers. Starting with the construction of the half adder, the half adder is then compiled into its own simplified circuit element which is then used to construct a full adder which is then compiled into its own element to construct the full adder. Chaining the adders together implies we can continuously chain them to add greater and greater numbers. The same logic associated with two's compliment for adding also infers the ability to subtract them by flipping the bits of the desired number to subtract and then adding the two.

References

[1] DE1-SoC User Manual

[2] DE1-SoC Computer System with ARM* Cortex* A9