Lab Assignment 3
Design of an Arithmetic-Logic Unit

Zhenming Yang, Connor McEleney

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

Instructor: Prof. Zagieboylo

d.zagieboylo@northeastern.edu

Due Date: October 5th, 2023

Submit Date: October , 2023

Abstract

In this lab an ALU that operates on 8-bit unsigned integers was constructed and simulated on the software Quartus Prime and then uploaded to the DE1-SoC using the 7-segment display and switches to perform calculations.

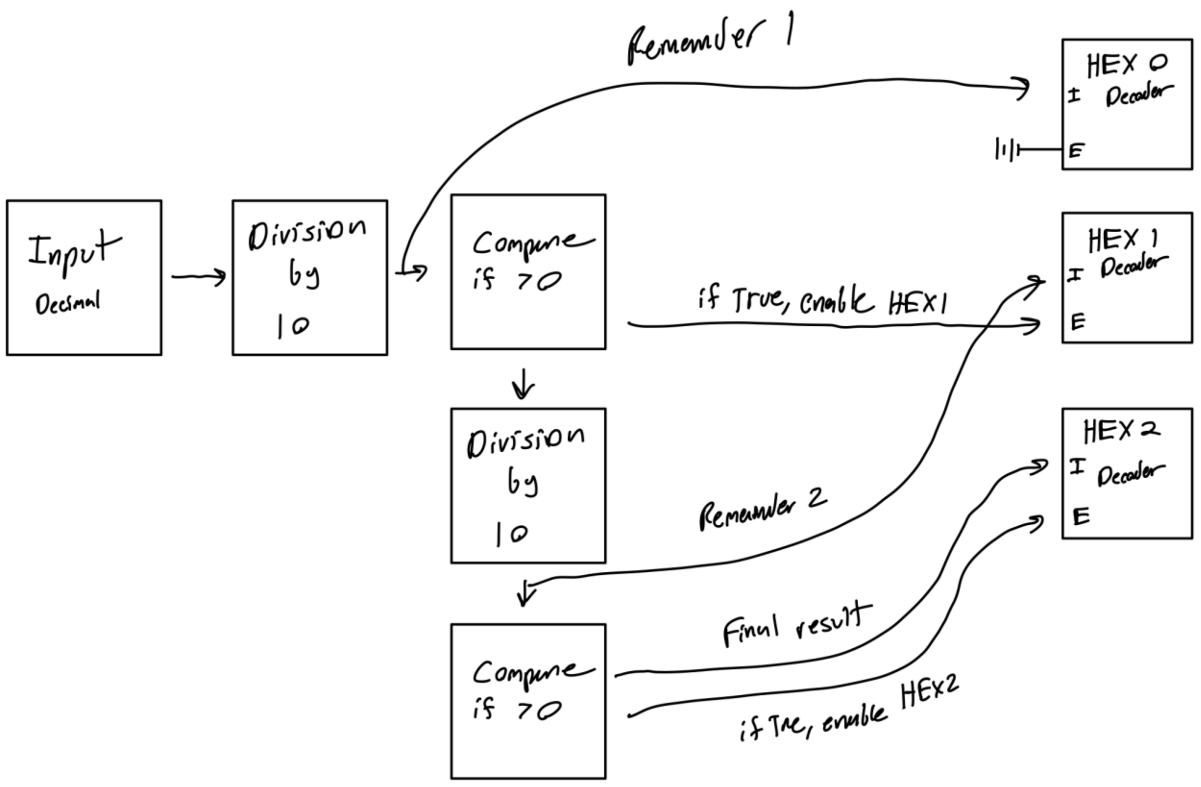
Introduction

The objective of this laboratory exercise is to create an arithmetic-logic unit (ALU) capable of performing operations on 8-bit unsigned integers. This ALU is designed to handle four distinct operations: addition, subtraction, multiplication, and division. The outcomes of these operations will be presented in binary format using LEDs and in decimal format using the 7-Segment displays on the DE1-SoC board [1].

Lab Setup

Pre-Lab

The Pre-Lab consisted of a flow diagram outlining the circuits logistical process



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: Display 8-Bit Unsigned Integer

In this part, we leverage certain components that were previously created in earlier labs, such as the 8-bit adder and the 7-Segment display with enable. Constructing all the connections from scratch using a circuit schematic editor can be quite time-consuming. Therefore, we enhance our previous circuits by integrating pre-designed libraries provided within the Quartus Prime software.

We first created a new project on Quartus Prime and set up all the prerequisites. Then we started to design the controller to display the 8-bit unsigned integer using the switches. In the process of creating this 8-bit to decimal display decoder, consider a scenario where the input binary value is 01111111 (equivalent to 127 in decimal). In this case, HEX2 will indicate '1', HEX1 will indicate '2', and HEX0 will display '7'. One approach to achieve this is by taking the input, dividing it by 10, and using the remainder (represented in 4 bits as values from 0 to 9) to display on HEX0. Subsequently, the quotient can be divided by 10 once more, with the new remainder displayed on the next display, and so forth (Figure 1).

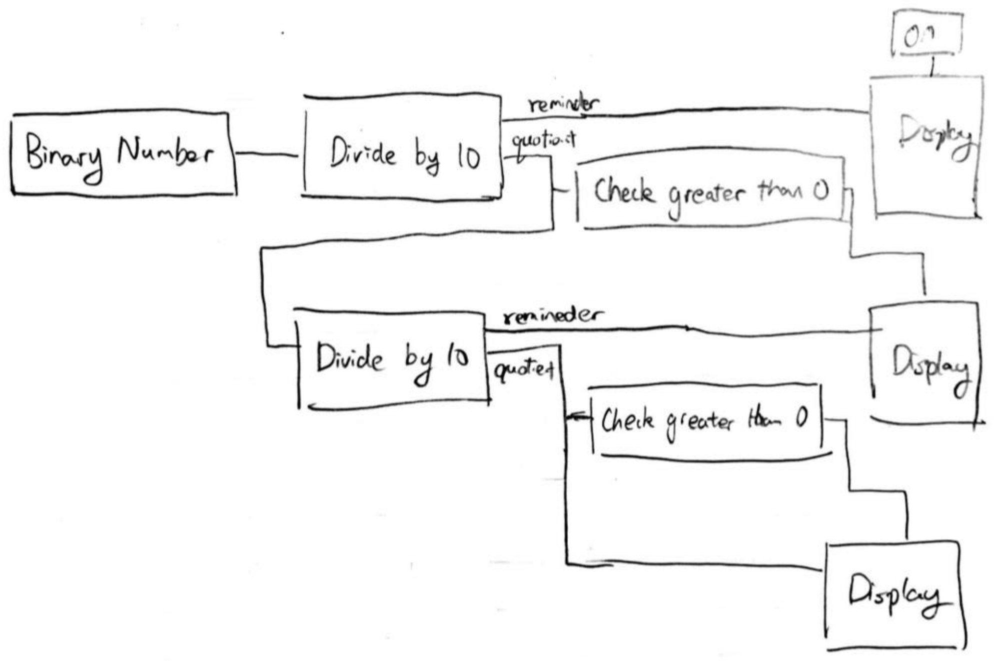


Figure 1: The design sketch of the 8-bit unsigned integer decoder.

We had to make sure that HEX0 is always enabled (connected to a logic high) so that it can display results even when the result is 0. However, the other displays should only be activated when necessary; otherwise, they should remain inactive. For instance, when displaying a number like 25, HEX1 will show '2,' HEX0 will show '5,' and HEX2 will be turned off (displaying nothing). To achieve this functionality, consider evaluating whether the quotient obtained after the initial division by 10 is greater than 0 or not. If it is indeed greater than 0, then enable that display; otherwise, keep it turned off. However, it's important to note that HEX1 must be able to display '0' for decimal numbers like 205.

Based on these intuitions and our initial design sketch, we created our schematics using **the 7-segment display controller, the LPM_Divide, LPM_Comper, and LPM_Constant modules** (Figure 2). The LPM modules are found in the IP library, where we configured them to fit our specific use case.

To connect the 1-bit input from the switches to the 8-bit LPM modules, we used data buses to convert the single input and push them into a data bus with the input in order (Figure 2). After the 8-bit division is finished computed, the result data buses are again split into 1-bit output connected to our 7-segment display decoder and put to the specific pins for each segment on the HEX display.

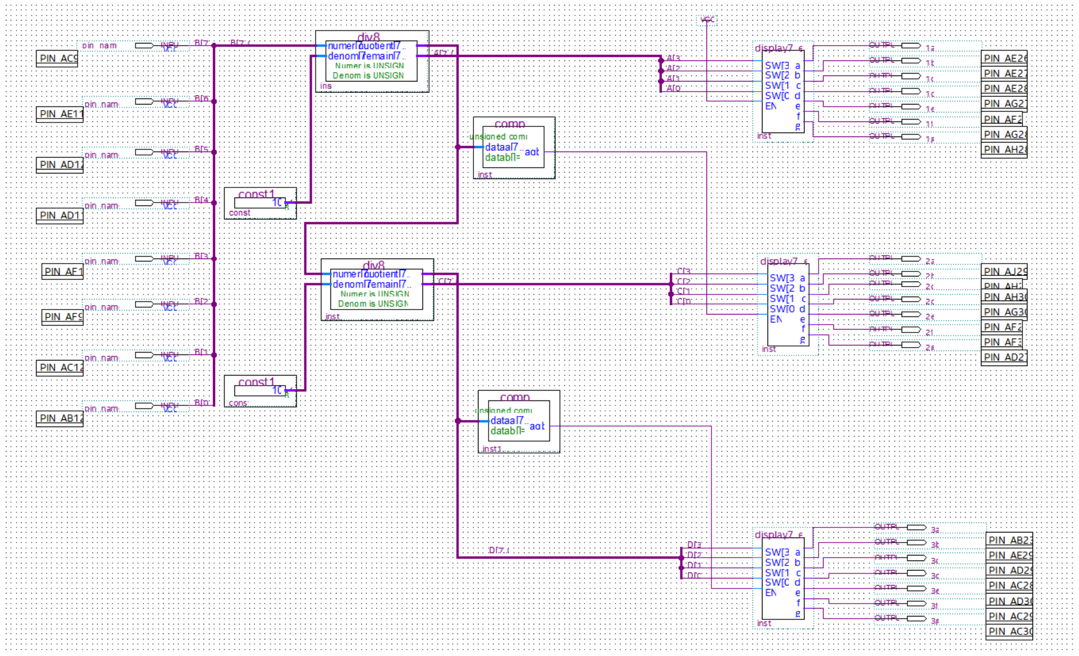


Figure 2: The schematics of the 8-bit unsigned integer decoder that translate binary numbers from the switches to decimal numbers on the HEX display.

We then uploaded our design into the DE1-SoC board and tested it on the 7-Segment displays with switches as your 8-bits input. We first generated a integer with only one digit to test our enable logic. We can see that the logic is successful with only the first digit HEX display on with its represented number 7 (Figure 3).

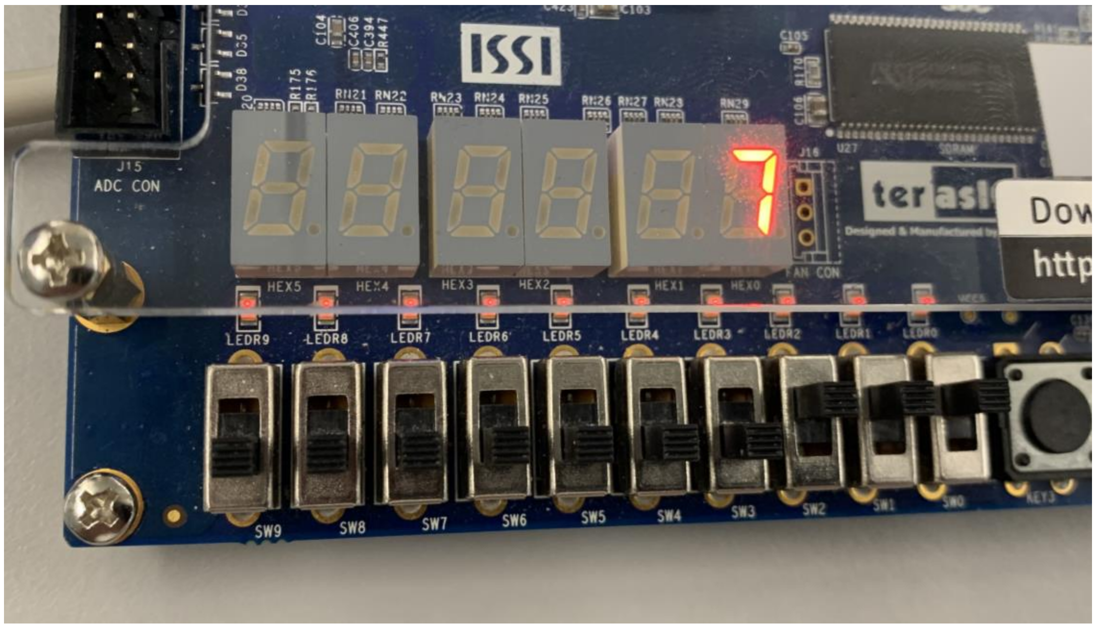


Figure 3: The Board displaying number "7" from our binary input "111." Noted the other HEX displays are not on when the number doesn't need them.

We then tested numbers that required two and three digits to see if the display correctly turned on and shows the corresponding number. The results are very promising (Figure 4/5).

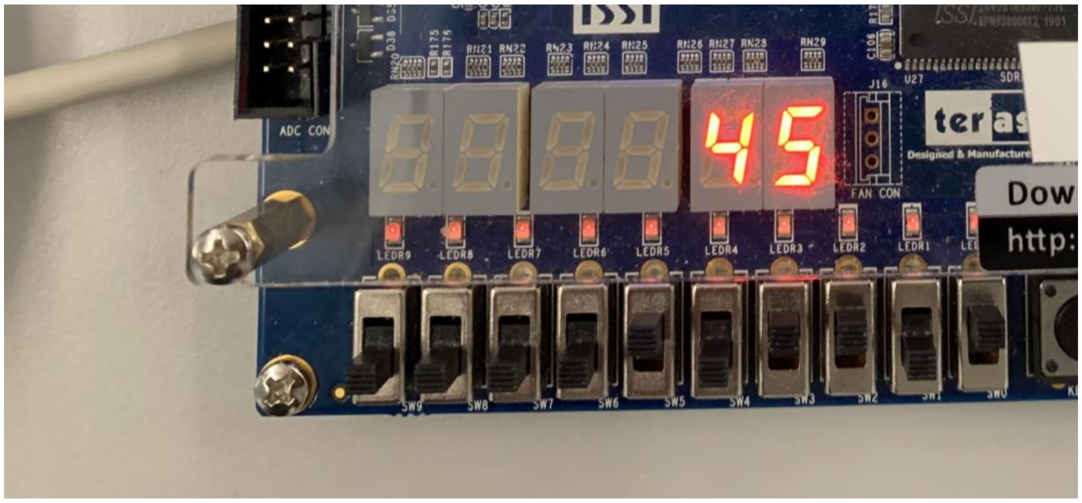


Figure 4: The Board displaying number "45 from our binary input "101101." Noted the other HEX displays are not on when the number doesn't need them.

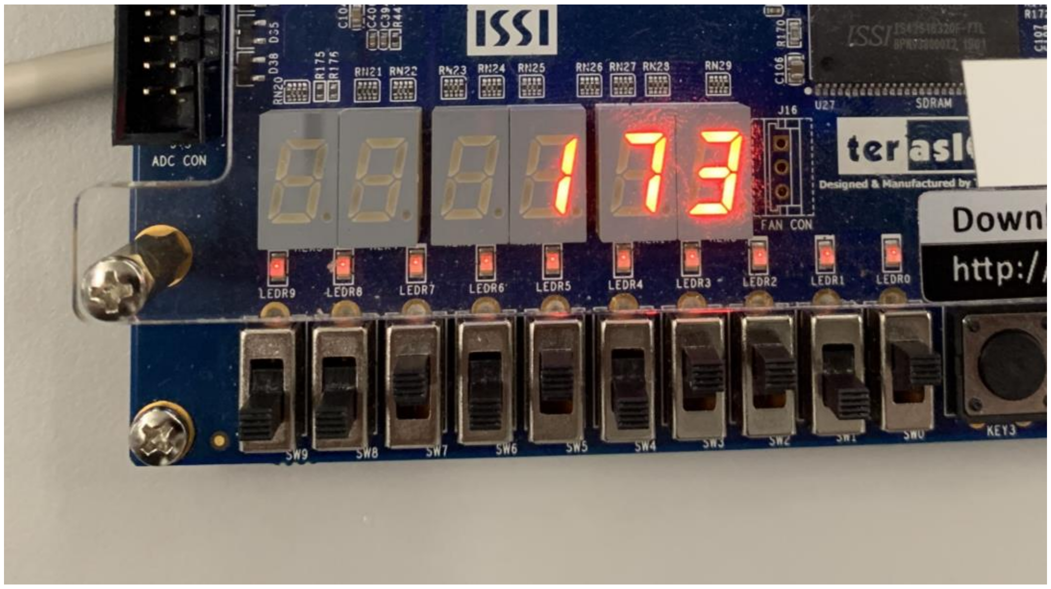


Figure 5: The Board displaying number "173" from our binary input "10101101." Noted the all the HEX displays are on to display every single digit of the number.

Finally, the schematics is packaged as a bmf file for Part 2 of the lab.

Part 2: The 8-bit ALU Design

In this part, we assembled the entire 8-bit ALU, utilizing all the logic components developed in both this and the preceding laboratory sessions. These components encompass the 8-bit adder, 8-bit subtractor, 8-bit multiplier, 8-bit divider, and the 8-bit 4-to-1 multiplexer.

We first created our schematics using **the 8-bit Unsigned Integer Display, the LPM_MUX, LPM_Divide, LPM_Multiply, LPM_Add, LPM_Sutract, and LPM_Constant modules** (Figure 2). The multiplexer takes the 2-bit input from the last two switches and controls what result is fed to the HEX display and the LEDs. Since we only have 10 switches in total, we set the first number put into each operation to be "32" and we connected all the input switches to a data bus and fed them to the second operands. The result from the MUX switch is connected to the 8 LEDs and the HEX display controller to output the final result in both binary and decimal (Figure 6).

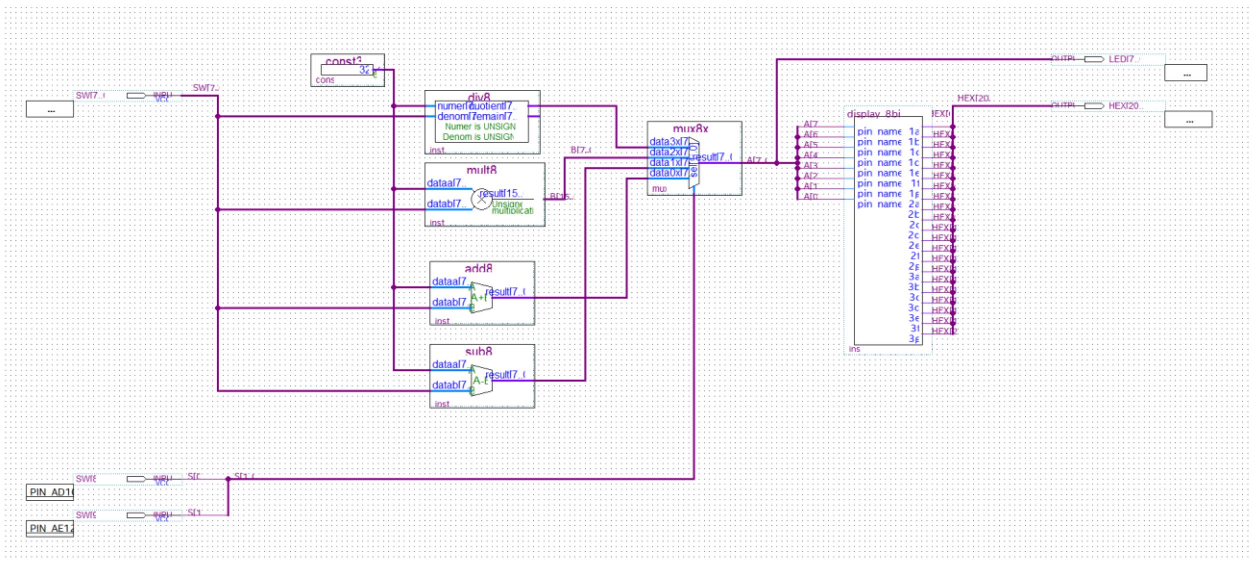


Figure 6: Figure 2: The schematics of the 8-bit ALU that takes binary numbers from the switches and do operations decided by the last two switches with "32" and display the result in decimal numbers on the HEX display and binary on the LEDs.

We then uploaded the schematics onto the DE1-SoC Board. We first switch the ALU into addition mode, and switch on switches to input number and get the correct answer (Figure 7). We then switch to subtraction mode by switching on SW8 and the ALU display the correct number on the display and the LEDs (Figure 8). We tried multiplication by only switching on SW9 and the ALU also displayed correctly (Figure 9). And finally, we turned into division by switching both SW8 and SW9 on and get the correct quotient for the division in decimal and binary (Figure 10).

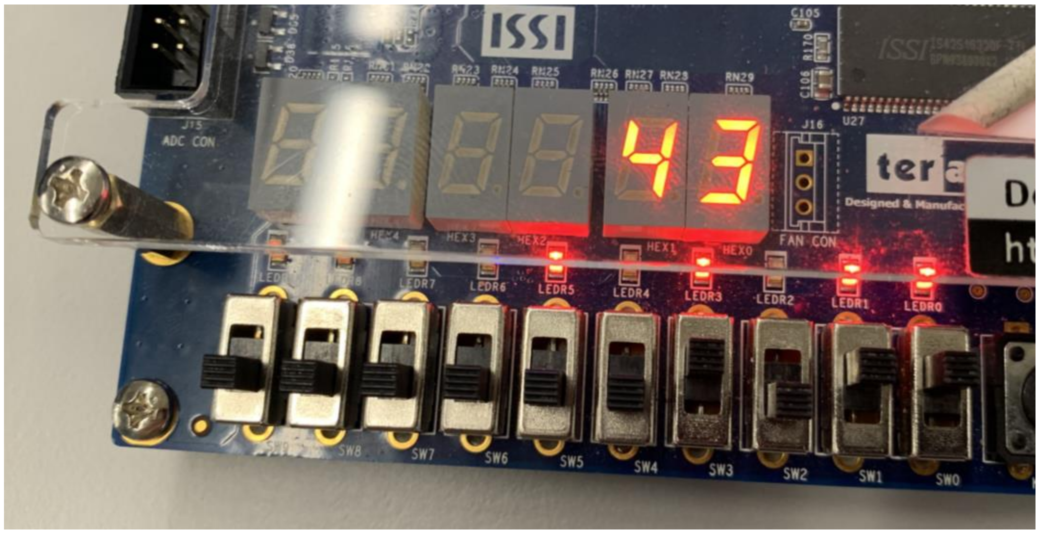


Figure 7: ALU in addition with SW8 and SW9 off. The second operand is number "11" from the first 8 switches and it is added with "32" to get "43" in decimal on the display and "101011" in binary with LEDs.

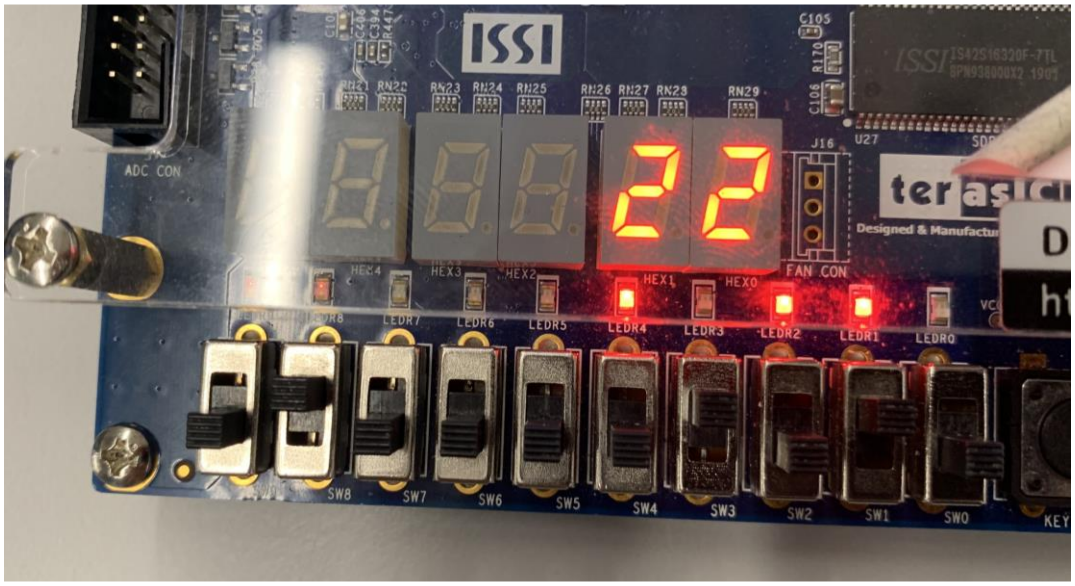


Figure 8: ALU in subtraction with SW8 on and SW9 off. The second operand is number "10" from the first 8 switches and it is subtracted from "32" to get "22" in decimal on the display and "10110" in binary with LEDs.

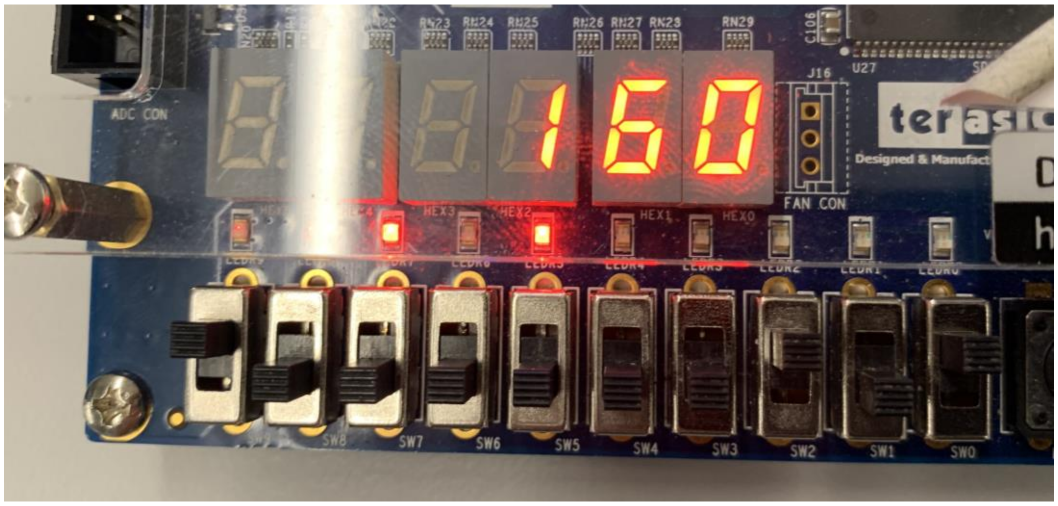


Figure 9: ALU in multiplication with SW8 off and SW9 on. The second operand is number "5" from the first 8 switches and it is multiplied with "32" to get "160" in decimal on the display and "10100000" in binary with LEDs.

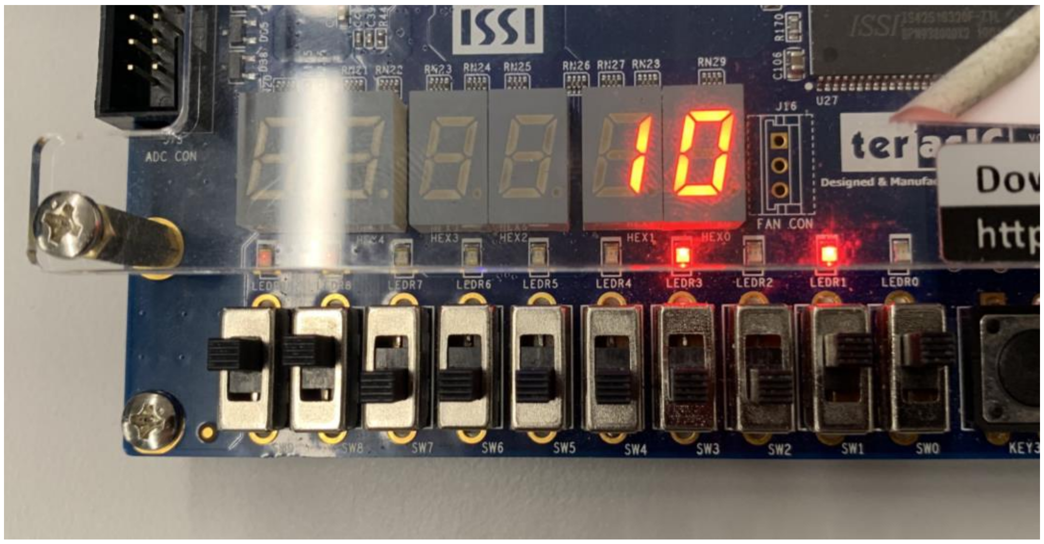


Figure 10: ALU in subtraction with both SW8 and SW9 on. The second operand is number "3" from the first 8 switches and it divides "32" to get "10" for the quotient in decimal on the display and "1010" in binary with LEDs.

Analysis

During Part 1 of the lab, we ran into some errors with the result not displaying the correct answer with the display kind of messed up. After going through our schematic and making sure the logic makes sense, we figured out that the problem happened during Pin assignment. After redoing every Pin again, the problem was resolved. We did not run into any problem with Part 2. With the help of LPM modules and the previously done 8-bit display controller, we implemented the ALU and produced the results with less difficulty.

Conclusion

This experiment integrated the accumulated skills and work learned and completed in the prior labs to assemble an ALU that operates on 8-bit unsigned integers. The organizational planning method of a flow chart was employed to lay the foundation and logic for the ALU before constructing the circuitry. The 8-bit adder and the segmented display unit with the enable bit were used in the construction of this ALU. The segmented display unit was modified to only display up to the most significant digit. This lab introduced and employed data buses which can carry multiple specific data inputs on one line. With the constant building upon the previous session from lab to lab it can be inferred that latches and memory circuits could be integrated with the existing circuitry to create a rudimentary calculator.

References

[1] DE1-SoC User Manual