COE328 Lab 6 Report

Design Of a Simple General-Purpose Processor

Nathaniel Recto
501162644
Section - 12

Table of Contents

Introduction:

The purpose of this lab was to showcase a student number while experimenting with various number tricks using the last 4 digits. To achieve this, Two latches were used for storing data, an ALU for mathematical operations, and a control unit comprising a Moore FSM and a 4x16 decoder.

The latches served as temporary storage for input values before transferring them to the ALU, which performed the desired mathematical operations. The ALU executed these operations based on assigned codes and displayed an 8-bit result on two 7-segment displays.

The control unit, incorporating a Moore FSM and a 4x16 decoder, functioned as a selector for the ALU, determining the actions performed. The FSM acted as an up-counter, cycling through states 0-8 and sending these states to the 4x16 decoder, which subsequently sent a 16-bit message to the ALU.

Components:

Latch 1 and Latch 2:

Two of the equivalent latches act as storage units which store input values (“A” and “B”) that will be passed on to the ALU and used in the boolean operations. The inputs are 8-bit binary representations of the last 4 digits of my student number in hex form, meaning that input A is $(26)_{16}$ and converted to (00100110) and input B is $(44)_{16}$ and converted to (01000100).

Clock	B Input $(44)_{16}$	Q (Output)
0	01000100	00000000
1	01000100	01000100
0	01000100	01000100
1	01000100	01000100

Table 1: Truth Table for Latch 2

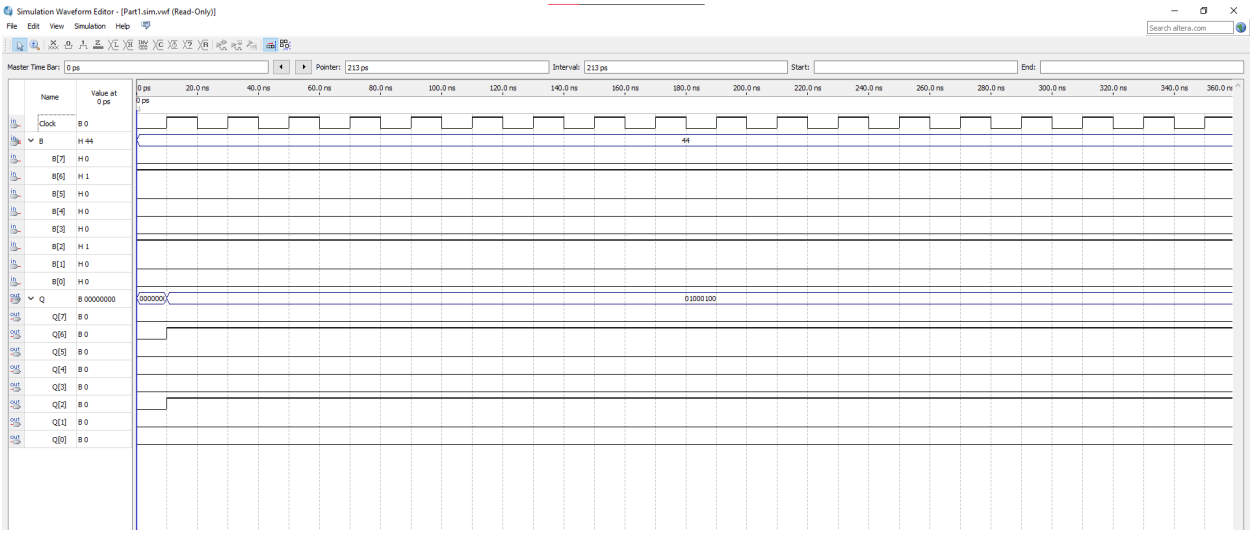

Figure 1: Waveform of latch 2 with input B (44_{16} in 8-bit binary representation)

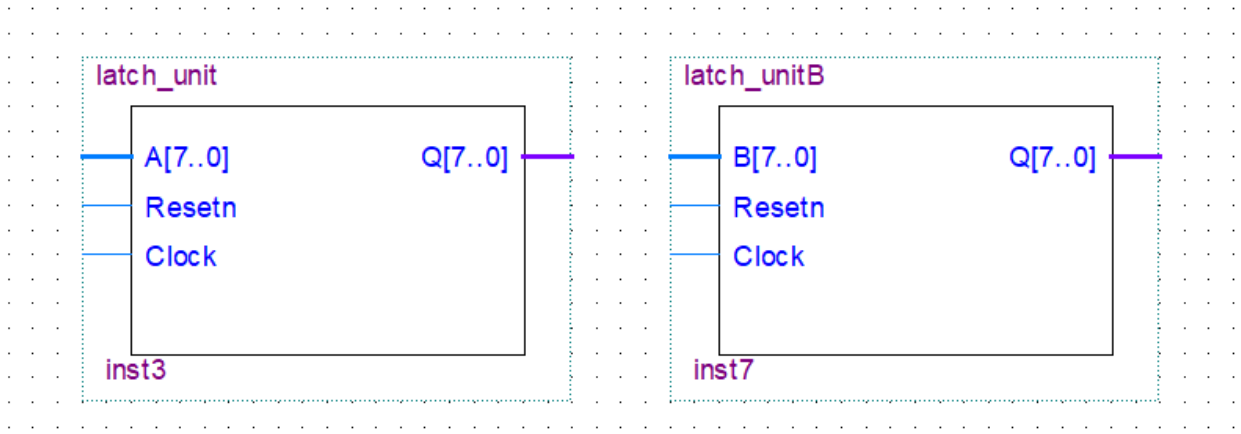

Figure 2: Block diagram of Latch 1 and Latch 2

Finite State Machine (FSM)

The FSM handles a 4-bit student ID code across nine states using clock and reset inputs. It accurately displays the current state and student ID through systematic Moore logic, in which the current state passes to the decoder.

Current State	Next State w = 0	Next State w = 1	Output (student ID)
0000	0000	0001	0101
0001	0001	0010	0000
0010	0010	0011	0001
0011	0011	0100	0001

0100	0100	0101	0110
0101	0101	0110	0010
0110	0110	0111	0110
0111	0111	1000	0100
1000	1000	0000	0100

Table 2: Truth Table for FSM

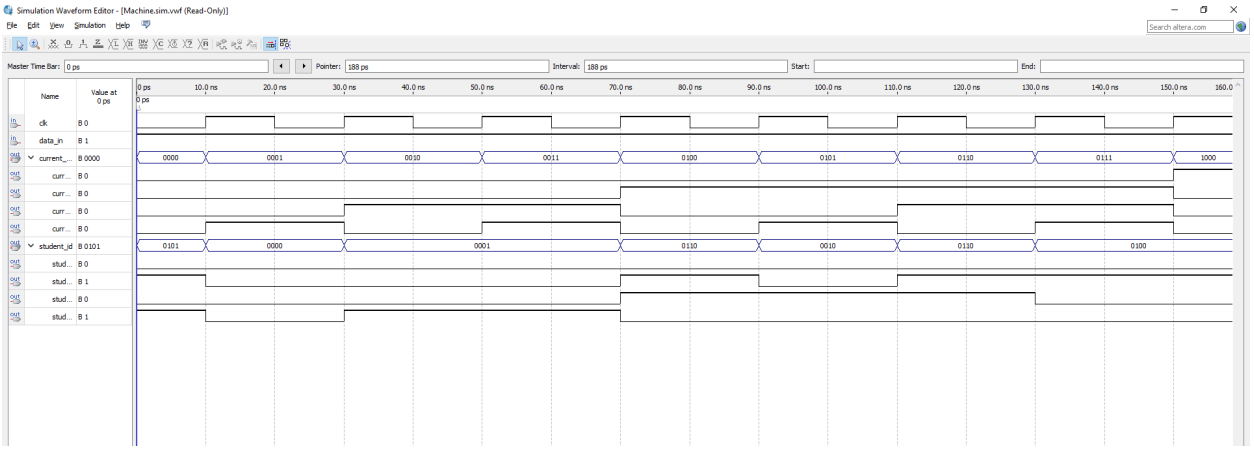

Figure 3: Waveform of FSM with student number

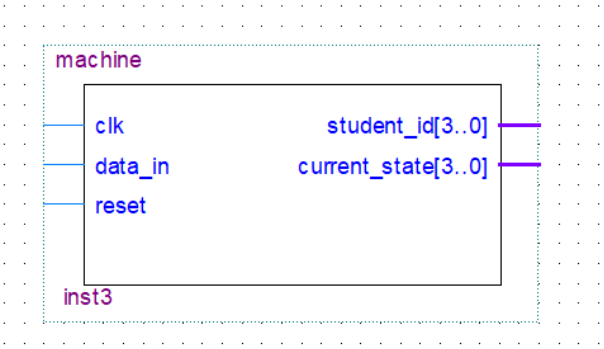

Figure 4: Block diagram of FSM

4x16 Decoder

The 4x16 receives the 4-bit current state from the FSM, decodes the value of the current state and produces a 16-bit output signal. This 16-bit output (OP) corresponds to one of nine predefined operations for ALU. The rest of the operations (10 to 16) will produce nothing (“0”) since it is irrelevant. The enable input signal is always “1” so that the decoder produces the 16-bit output signal accordingly from the current state.

Enable = “1”

Current State	Operation
0000	0000000000000001 (-- Operation 1)
0001	0000000000000010 (-- Operation 2)
0010	0000000000000100 (-- Operation 3)
0011	0000000000001000 (-- Operation 4)
0100	0000000000010000 (-- Operation 5)
0101	0000000000100000 (-- Operation 6)
0110	0000000001000000 (-- Operation 7)
0111	0000000010000000 (-- Operation 8)
1000	0000000100000000 (-- Operation 9)
1001	0000000000000000 (-- No operation)
1010	0000000000000000 (-- No operation)

Table 3: Truth Table for 4x16 Decoder

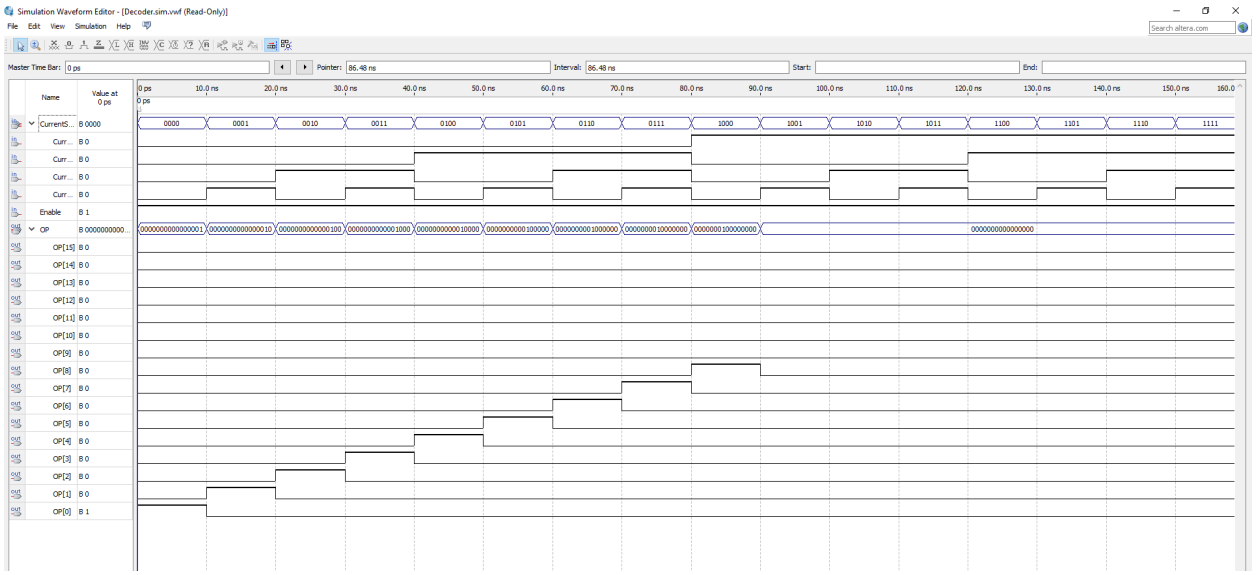

Figure 5: Waveform of 4x16 Decoder

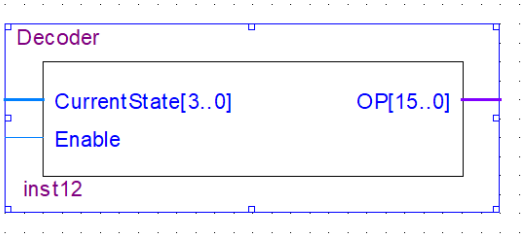

Figure 6: Block diagram of 4x16 Decoder

ALU_1 For Problem Set 1

The ALU, a crucial part of the GPU unit, executes the various arithmetic and logical operations. With 8 inputs (A, B, Clock, Data_in, En, ResetA, ResetB, FSM_Reset) and 4 outputs (Neg, student_id, R1, and R2), it aims to perform precise boolean operations between 8-bit values A and B based on its assigned microcode. To achieve this, the ALU leverages the 16-bit “current state” of the FSM as a selector for the specific operation to be executed between A and B.

In this problem set, only 5 out of 8 inputs were utilized. “A” and “B” stand for the values of the last 4 digits of my student number, while “OP” selects the desired boolean operation. “Clock” dictates the timeframe for output activation. The input Data_in and En (enabled) are always “1” to produce constant inputs, and operations, and maintain a specific state during certain clock cycles. I decided not to use the reset inputs due to propagation delays as the reset inputs cause it to not display any results. The outputs “R1” and “R2” divide the 8-bit boolean operation result into two 4-bit numbers in order to display the 7-segment component on the FPGA board. Meanwhile, “Neg” indicates the negative sign if the boolean operation yields a negative result on the FPGA board. The output “student_id” displays my student number from the FSM at each current state. Every output is displayed as 7 segments in order to show on the FPGA board.

Function #	Microcode	Boolean Operation / Function
1	0000000000000001	sum(A, B) 6A
2	0000000000000010	diff(A, B) -1E
3	0000000000000100	$\overline{A}$ d9
4	0000000000001000	$\overline{A \cdot B}$ Fb
5	0000000000010000	$\overline{A + B}$ 99
6	0000000000100000	A · B 04
7	0000000001000000	A ⊕ B 62
8	0000000010000000	A + B 6A
9	0000000100000000	$\overline{A \oplus B}$ 9d

Table 3: Microcode Table of ALU_1

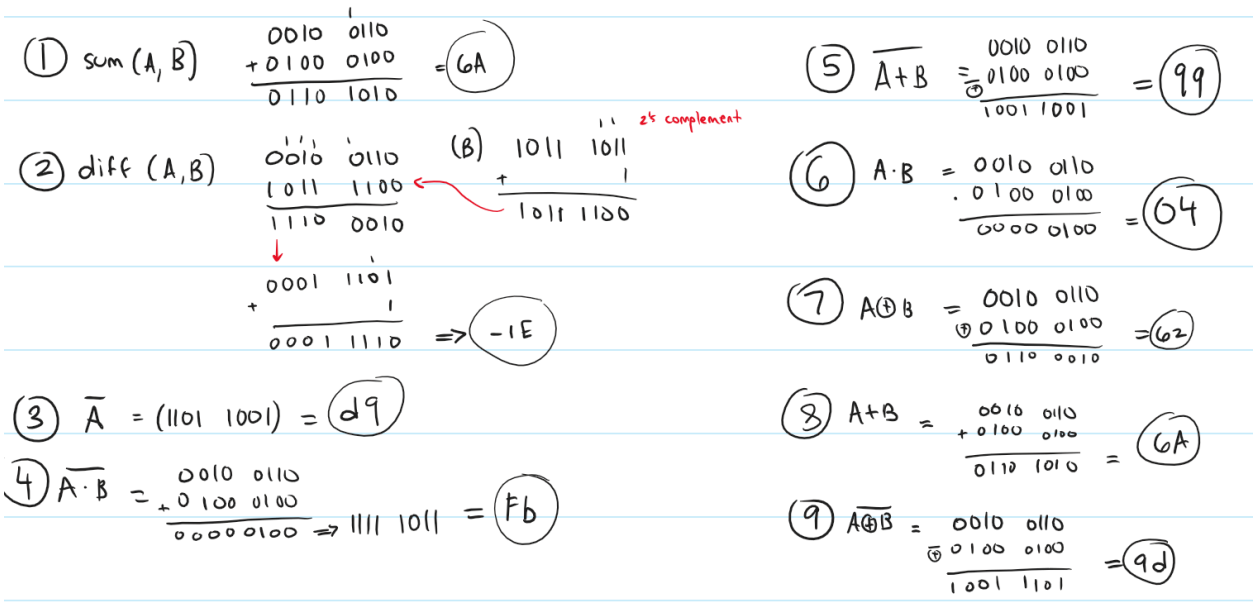

Figure 7: Handwritten results of each function

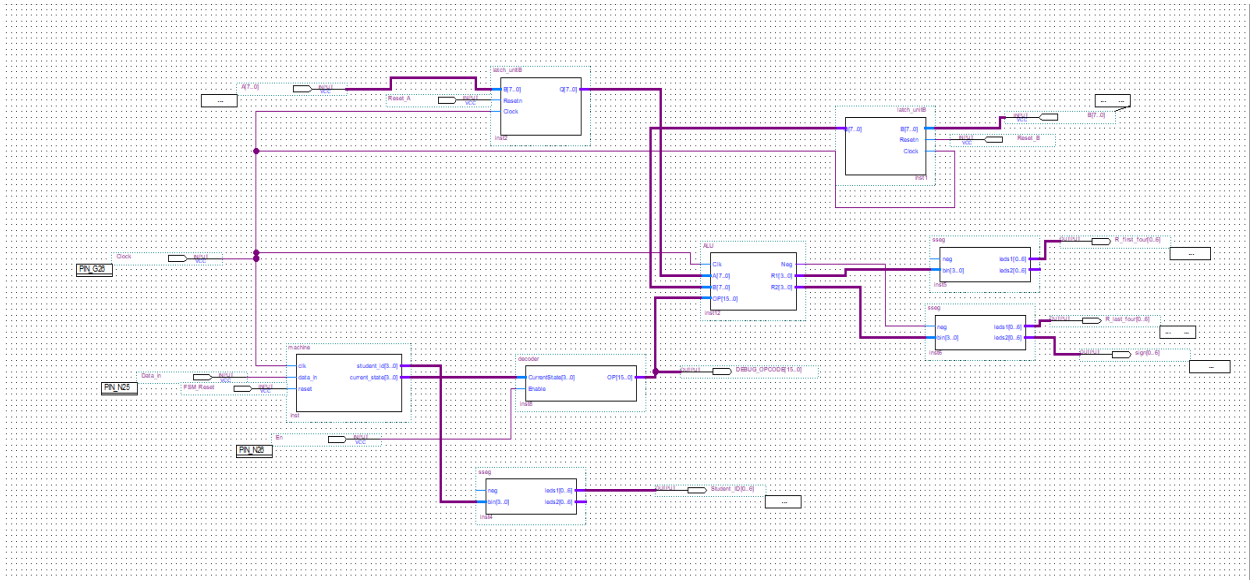

Figure 8: Block diagram of Problem Set 1

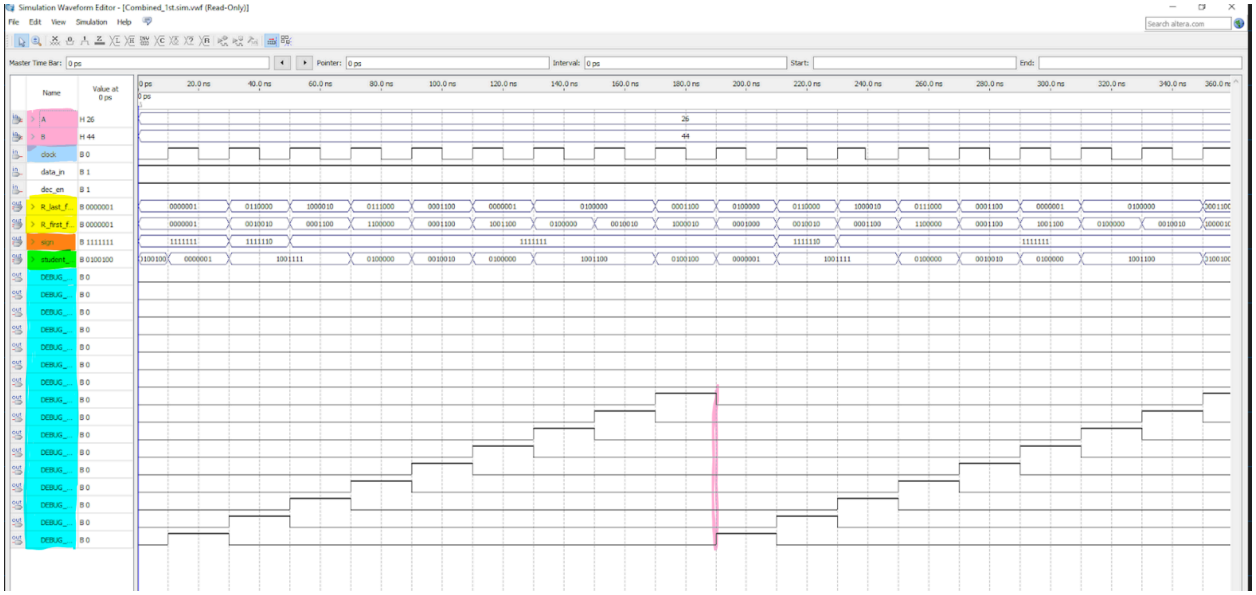

Figure 9: Waveform of Problem Set 1

It is shown that input “data_in” and “En” are always “1”, the highlighted blue is the clock input where it clocks high and low, highlighted pink are the “A” and “B” inputs, highlighted yellow are the “R1” and “R2” outputs and the highlighted green is my student id. All outputs are in 7-segments. The pink line in the middle is the clock recycle where it now shows the results from operation 1 to 9.

To understand the ALU operation, I added an output “DEBUG_OPCODE” (turquoise highlighted) that displays which operation the ALU is in. According to the waveform, the operation is delayed by a microcode, making the first operation start at

the second student number. This was caused due to both FSM and ALU clocking at the same time and the result of the ALU lags behind the FSM/Decoder output. This happens because the opcode does not have time to get to the ALU before the ALU updates its function since they both update at the same time in the same clock cycle.

Note that my 7-segment is inverted because the LED lights were inverted in the FPGA board.

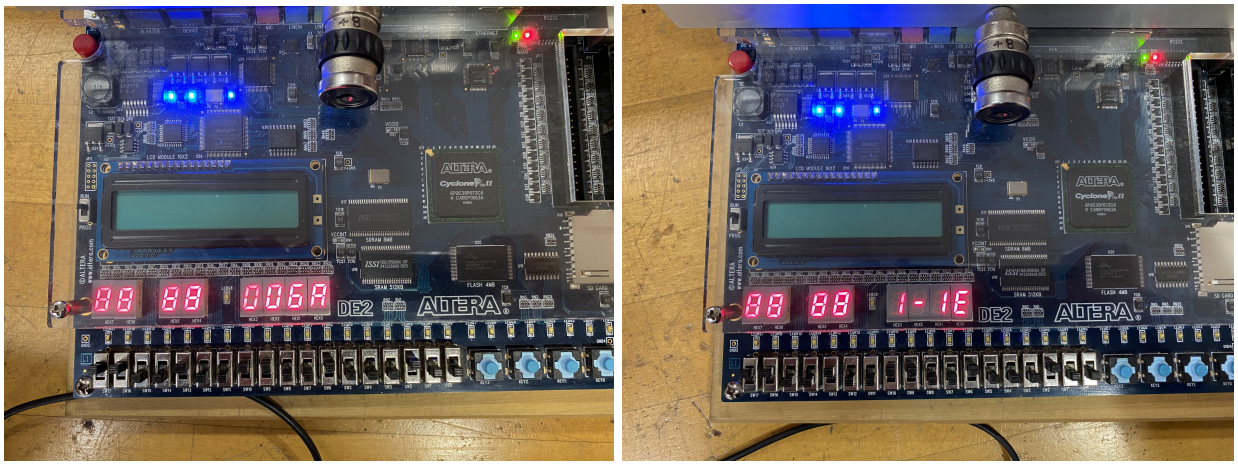

Figure 10: First two Results of ALU_1 Problem Set 1

On Hex[3], my student number is displayed, Hex[2] shows the sign, Hex[1] shows R2 output, and Hex[0] shows R1 output.

ALU_2 For Problem Set 2

The ALU design in this problem set is the same as the design from the previous problem set. Its aim is to execute distinct boolean functions on inputs A and B according to the designated microcode. The only difference is that all operations/functions are operated differently in the ALU for the A and B input values. This means that the same 5 out of 8 inputs were used, and 4 same outputs are being displayed and served the same purpose as ALU_1. The modified 9 operations I chose were part a).

a)

Function #	Operation / Function	
1	Increment **A** by 2	28
2	Shift **B** to right by two bits, input bit = 0 (SHR)	11
3	Shift **A** to right by four bits, input bit = 1 (SHR)	F2
4	Find the smaller value of **A** and **B** and produce the results (Min(**A**,**B**))	26
5	Rotate **A** to right by two bits (ROR)	89
6	Invert the bit-significance order of **B**	22
7	Produce the result of XORing **A** and **B**	62
8	Produce the summation of **A** and **B**, then decrease it by 4	76
9	Produce all high bits on the output	FF

Table 4: Microcode Table of ALU_2

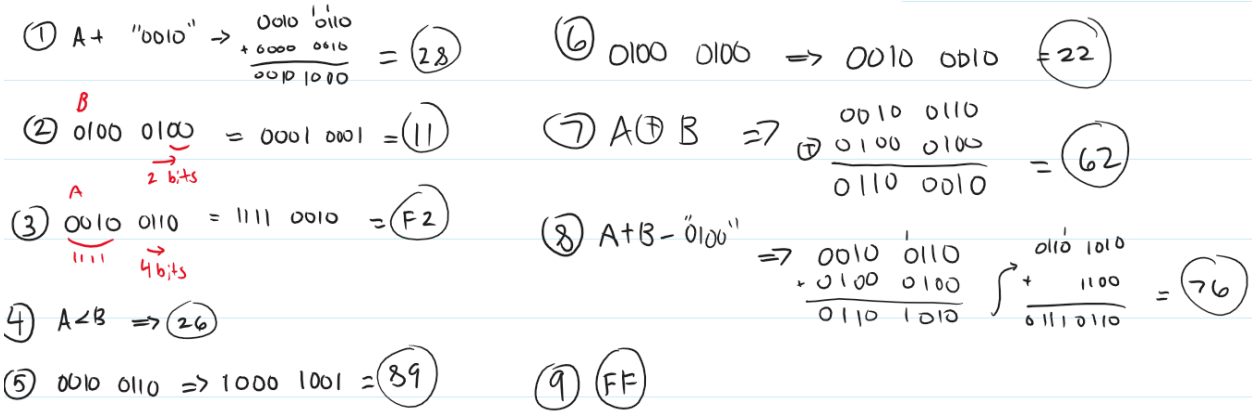

Figure 11: Handwritten results of each function

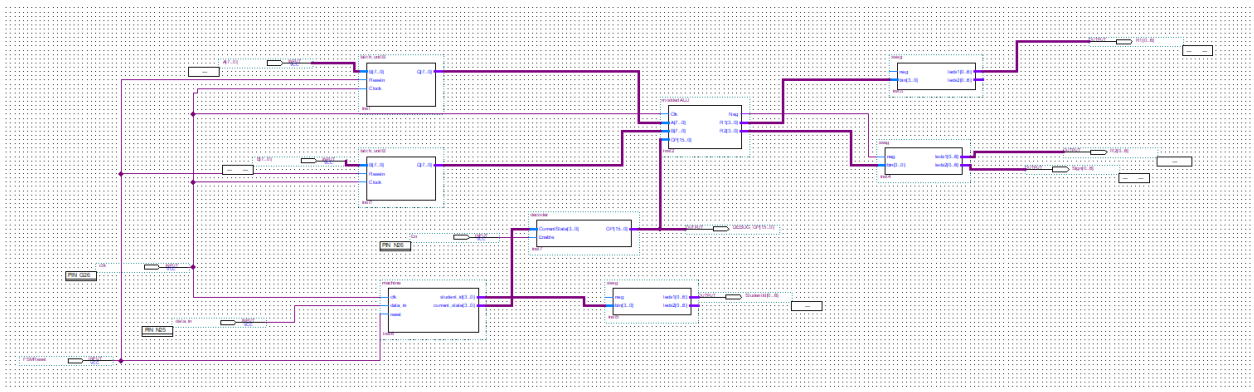

Figure 12: Block diagram of ALU_2 Problem Set 2

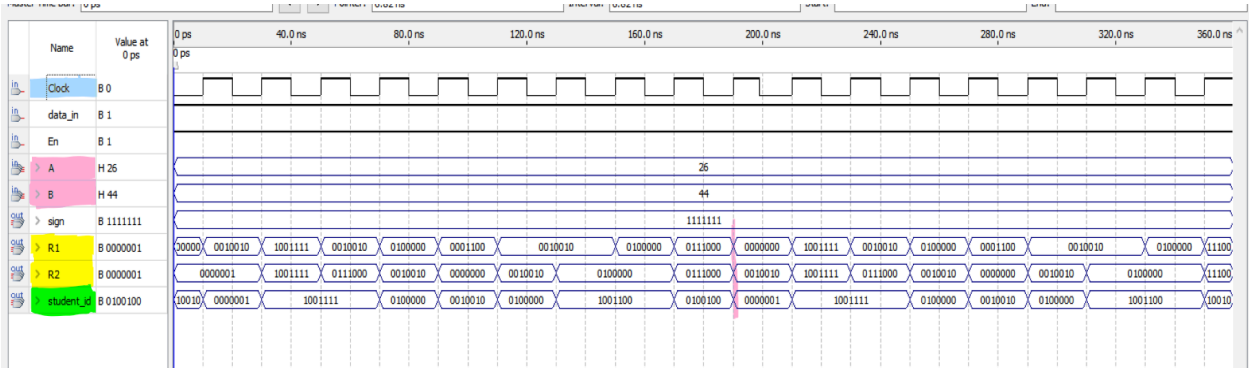

Figure 13: Waveform of ALU_2 Problem Set 2

Similarly, with the previous waveform, it is shown that input “data_in” and “En” are always “1”, the highlighted blue is the clock input where it clocks high and low, highlighted pink are the “A” and “B” inputs, highlighted yellow are the “R1” and “R2” outputs and the highlighted green is my student id. All outputs are in 7-segments. The pink line in the middle is the clock recycle where it now shows the results from operation 1 to 9.

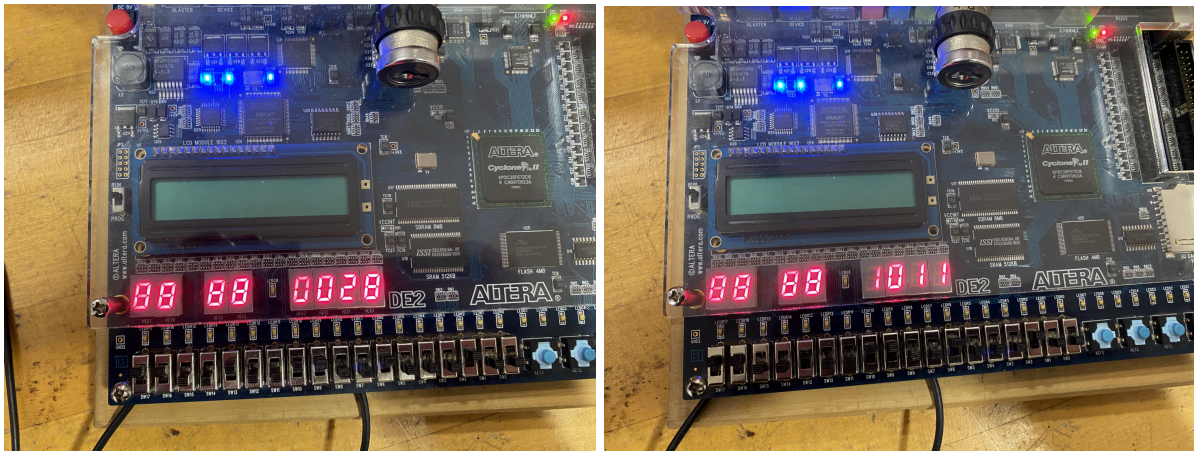

Figure 14: First two Results of ALU_2 Problem Set 2 (same Hex order from ALU_1)

ALU_3 For Problem Set 3

In problem set 3, the ALU and the 7-segment were modified so that a 4-bit student ID input is introduced in the ALU, taken from the FSM. The output “leds” will display “y” if a digit of my student number is odd, and “n” otherwise. This means that there will be only two outputs, the student number and the “leds”. The inputs are the same as the previous problem sets and do the same purpose. This is for part a).

a) For each microcode instruction, display 'y' if the FSM output (**student_id**) is odd and 'n' otherwise

Function #	Microcode	Student_ID	“y” or “n”
1	0000000000000001	5	y
2	0000000000000010	0	n
3	0000000000000100	1	y
4	0000000000001000	1	y
5	0000000000010000	6	n
6	0000000000100000	2	n
7	0000000001000000	6	n
8	0000000010000000	4	n
9	0000000100000000	4	n

Table 5: Microcode Table for ALU_3

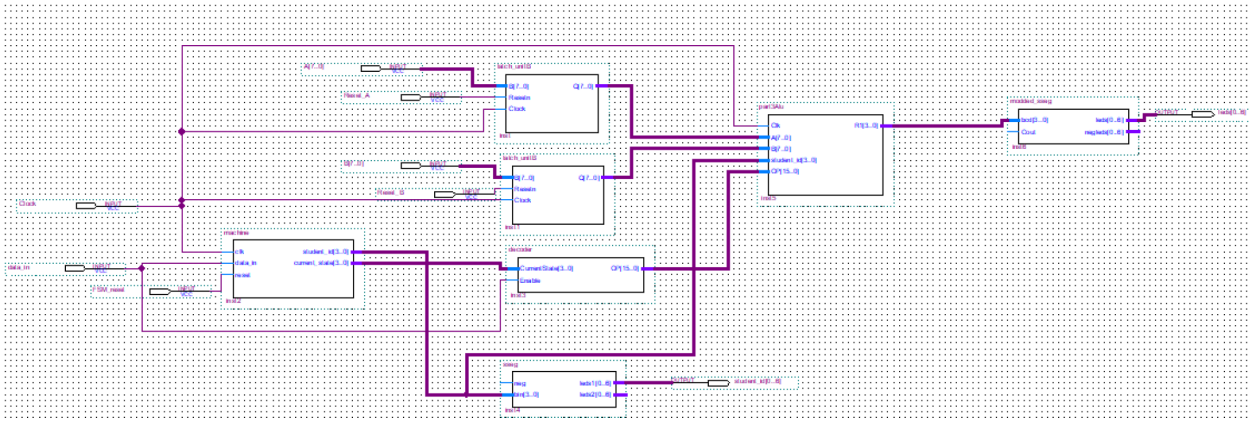

Figure 15: Block diagram of ALU_3 Problem Set 3

I decided to connect “Enable” to data_in since both inputs will always be “1”.

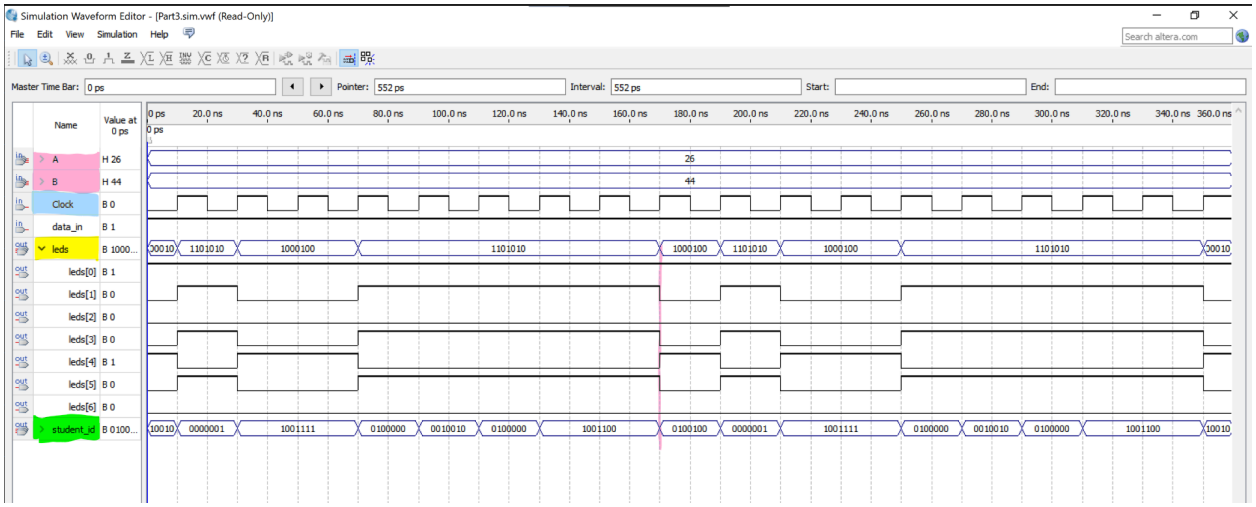

Figure 16: Waveform of ALU_3 Problem Set 3

Similarly, for the previous waveforms of the problem sets, the highlighted pink are input “A” and “B”, data_in with enable connected is always “1”, the highlighted yellow is the “leds” display showing “y” (1000100) or “n” (1101010), and the highlighted green is the student_id. The pink line is the recycle that goes back to the beginning of my student ID.

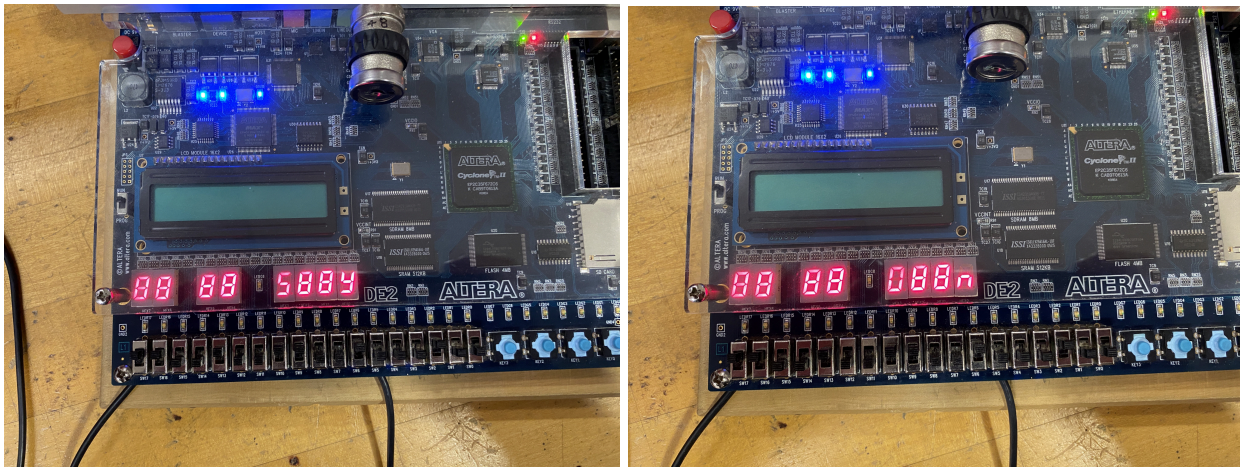

Figure 1: First two Results of ALU_3 Problem Set 3

On Hex[3], my student number is displayed and Hex[0] shows the “leds” output displaying either “y” or “n”.

Conclusion

In conclusion, this laboratory experiment demonstrates the design and utilization of an ALU on an FPGA board, offering flexibility by allowing easy circuit alteration through ALU modifications. The ALU's versatility enables diverse functions, while in conjunction with the latch and FSM, it stores values and transitions to the next state. Following the function execution and result retrieval on the FPGA board, the output is showcased using a 7-segment display.