Course Number	COE328
Course Title	Digital Systems
Semester/Year	Fall 2023
Instructor	Professor Reza Sedaghat
TA name	Menglu Li
Section Number	06

Lab 6 Report

Report Title	Design of a Simple General-Purpose Processor
Submission Date	2023-12-03
Due Date	2023-12-04

Student Name	Student ID (xxxx1234)
Jason Su	501158090

1. Table of Contents

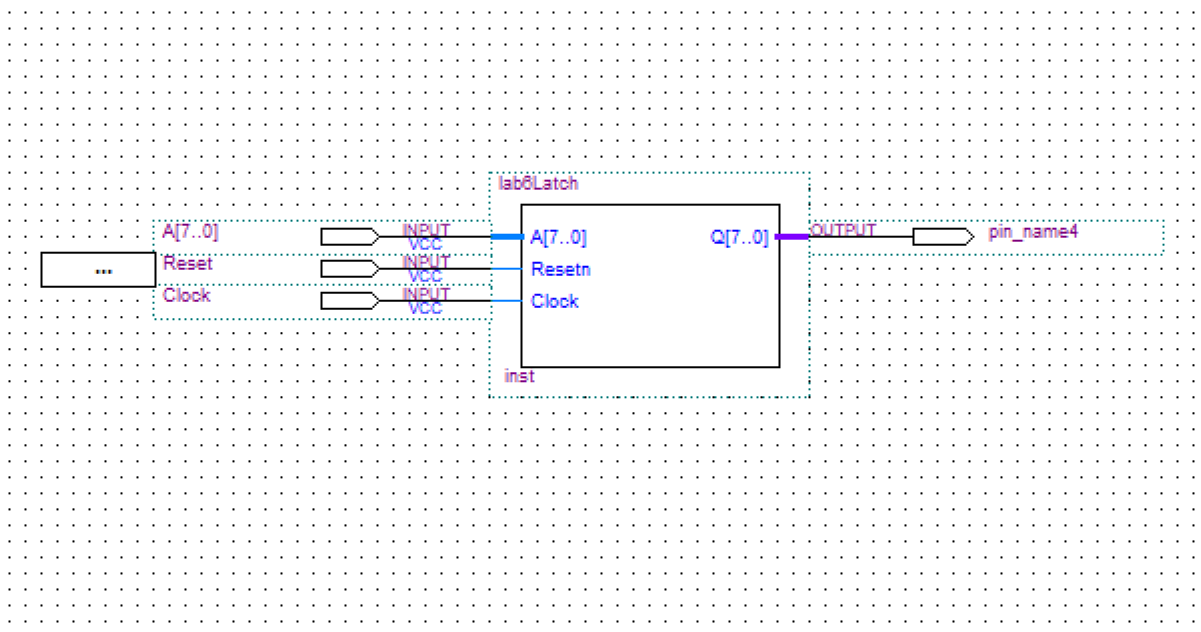
Cover Page……………………………………………………………………………..1
Table of Contents……………………………………………………………………….2
Introduction…………………………………………………………………………...…2
Components……………………………………………………………………………..2
Latches……………………………………………………………………………….3
FSM…………………………………………………………………………………….4
4x16 Decoder…..……………………………………………………………………….6
ALU_1 for Problem Set 1 of the Lab 6 procedure……..……………………………………….7
ALU_2 for Problem Set 2 of the Lab6 procedure..…………..………………………………10
ALU_3 for Problem Set 3 of the Lab6 procedure………...………………………………...…14
Conclusion……………………………...…………...………………………..……….16

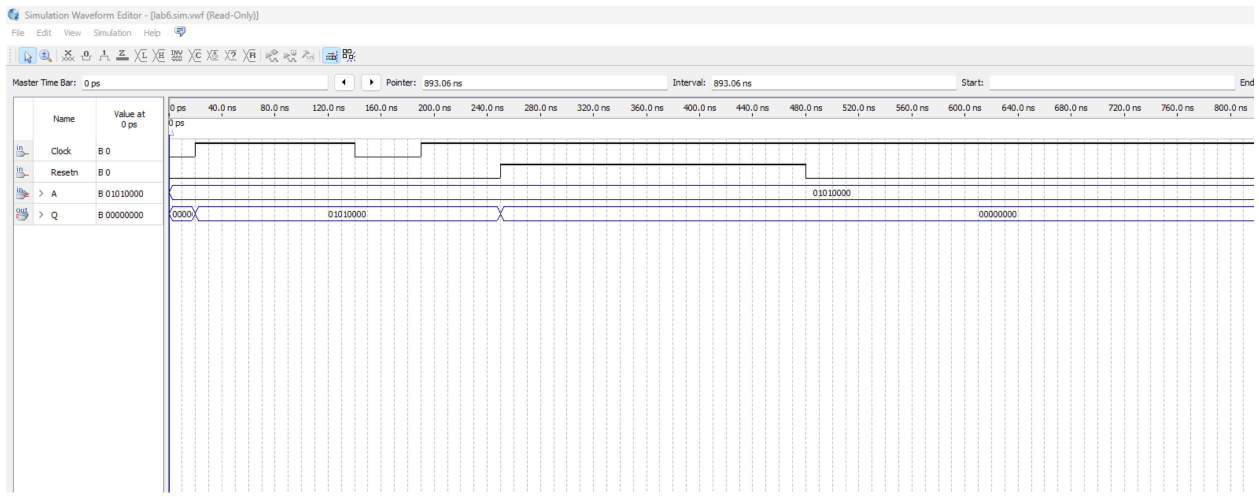
2. Introduction

The Simple General-Purpose Processor designed in our lab consists of 2 latches, 4 seven segment displays, an arithmetic logic unit, a FSM, and a 4x16 decoder. Each latch processes an 8-bit input (A and B) which represents the last four digits of our student number. If the reset is not on, the 8-bit input is sent to the ALU through a positive edge clock input. The FSM is a moore machine that determines and switches to the next depending on the previous state. There are a total of 9 states that switch through chronologically when data_in is on and the clock has a positive edge. Each state has a corresponding student number digit which is sent to a seven segment display while the current state is passed onto the 4x16 decoder. The 4x16 decoder is comprised of two 3x8 decoders and converts the 4-bit state inputs into 16-bit while the enable is on. The ALU receives inputs A and B from the two latches, the state in 16-bit from the decoder, and a clock signal which is synchronous throughout the whole system. The ALU performs a different calculation of A and B depending on the state, with the sign and upper and lower four bit results sent and displayed on a seven segment display.

3. Components:

Latches



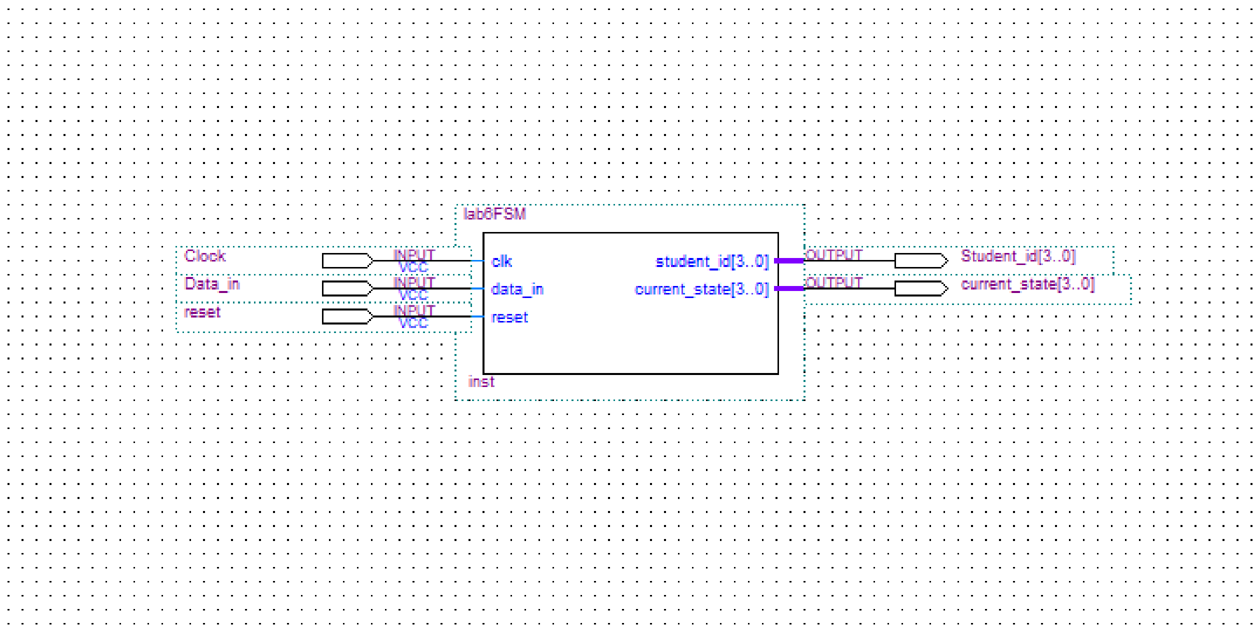


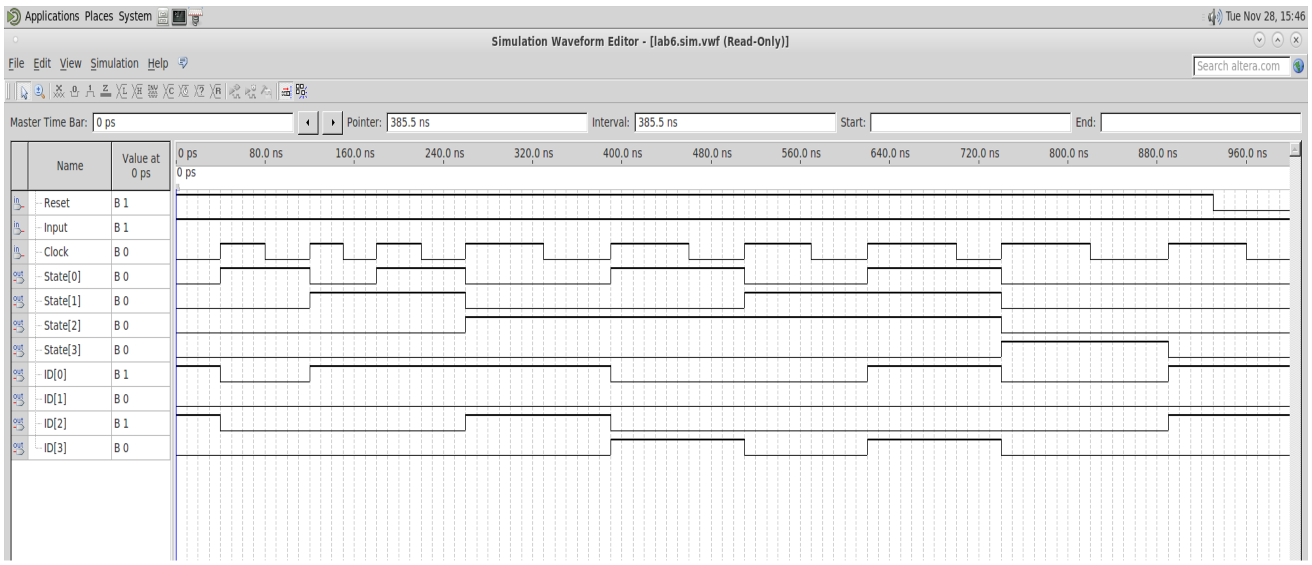
The latches have three inputs, our 8-bit student number, clock, and reset. The latches are positive edge triggered and output the student number for each positive edge if reset is off. If reset is on, the output would instead be “00000000”.

Clock	A	R	Q(t + 1)
0	x	x	Q(t)

1	0	0	Q(t)
1	0	1	0
1	1	0	1
1	1	1	x

FSM





Simulation Waveform Editor - [lab6.sim.vwf (Read-Only)]

Master Time Bar: 0 ps   Pointer: 385.5 ns   Interval: 385.5 ns   Start: End:

Name	Value at 0 ps
Reset	B 1
Input	B 1
Clock	B 0
State[0]	B 0
State[1]	B 0
State[2]	B 0
State[3]	B 0
ID[0]	B 1
ID[1]	B 0
ID[2]	B 1
ID[3]	B 0

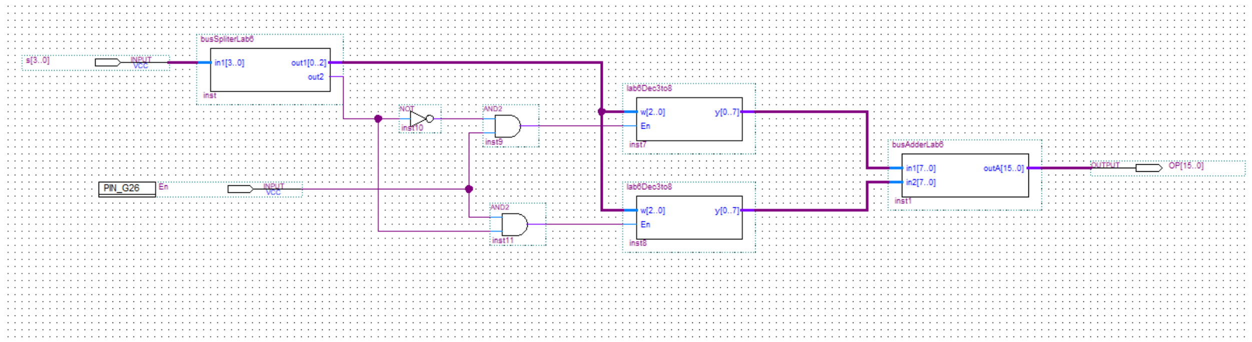
The FSM is a Moore Machine that counts through 9 states in the order of state 0 to 8. Because it is a Moore Machine, each state holds the 4-bit representation of our student number and current state.The student number is sent to a seven segment to be displayed while the 4-bit current state is sent to the 4x16 decoder.

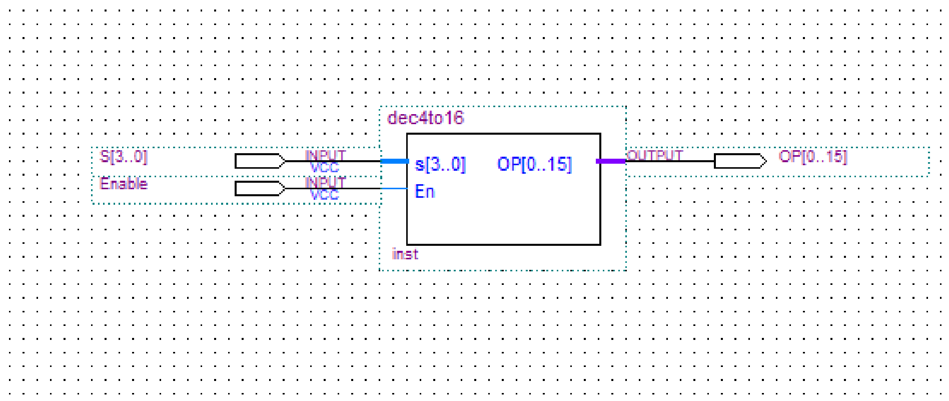
Clock	Data in	Reset	Student ID	Current State
0	1	1	xxxx	xxxx
1	1	0	0000	0000
1	1	1	0101	0000
1	1	1	0000	0001
1	1	1	0001	0010
1	1	1	0001	0011
1	1	1	0101	0100
1	1	1	1000	0101
1	1	1	0000	0110
1	1	1	1001	0111
1	1	1	0000	1000

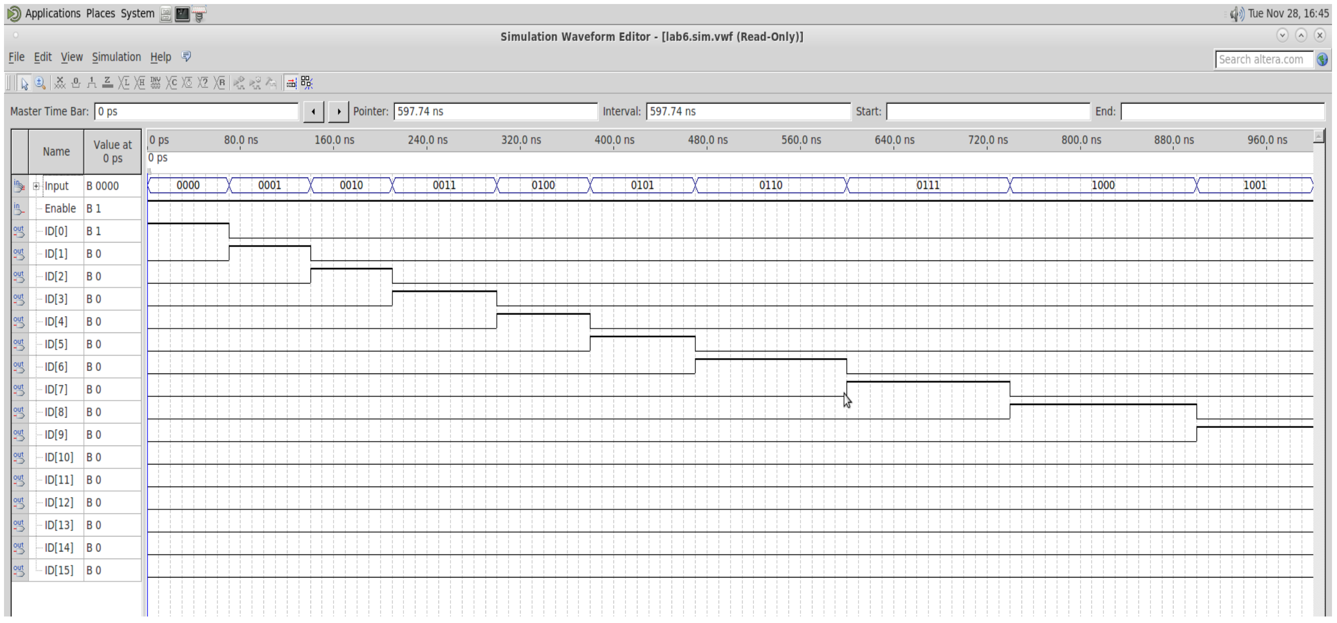
Present State	Next State Clock = 0	Next State Clock = 1	Student ID
0000	0000	0001	0000
0001	0001	0010	0101
0010	0010	0011	0000
0011	0011	0100	0001
0100	0100	0101	0001
0101	0101	0110	0101

0110	0110	0111	1000
0111	0111	1000	0000
1000	1000	0000	1001

4x16 Decoder







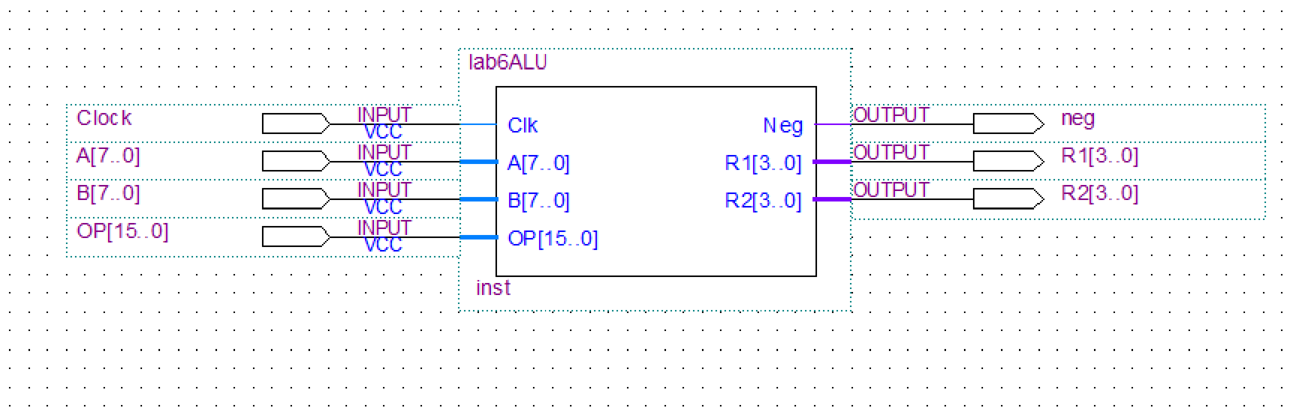
Applications Places System
Tue Nov 28, 16:45
Simulation Waveform Editor - [lab6.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com

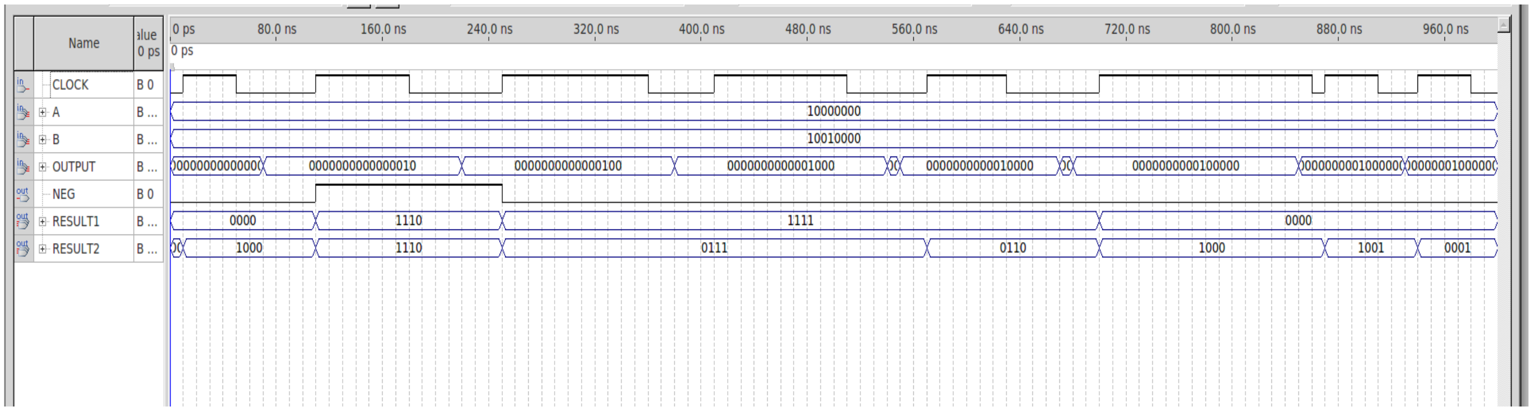
```
Master Time Bar: 0 ps    Pointer: 597.74 ns    Interval: 597.74 ns    Start:    End:

Name      Value at 0 ps
Input     B 0000
Enable    B 1
ID[0]     B 1
ID[1]     B 0
ID[2]     B 0
ID[3]     B 0
ID[4]     B 0
ID[5]     B 0
ID[6]     B 0
ID[7]     B 0
ID[8]     B 0
ID[9]     B 0
ID[10]    B 0
ID[11]    B 0
ID[12]    B 0
ID[13]    B 0
ID[14]    B 0
ID[15]    B 0
```

The 4x16 Decoder utilizes two 3x8 decoders, two AND gates, a NOT gate, and a bus splitter/adder. The bus splitter/adder is needed to split the single 4 variable vector input into two nodes, then combine the two 8 variable vector outputs into a single 15 variable vector. The state input is received from the FSM in 4-bit and is converted into a 16-bit state while the enable is active. The 16-bit state is sent to the ALU as the microcode to determine the state and function.

Enable	Current State	Output
0	x	0000000000000000
1	0000	0000000000000001
1	0001	0000000000000010
1	0010	0000000000000100
1	0011	0000000000001000
1	0100	0000000000010000
1	0101	0000000000100000
1	0110	0000000001000000
1	0111	0000000010000000
1	1000	0000000100000000

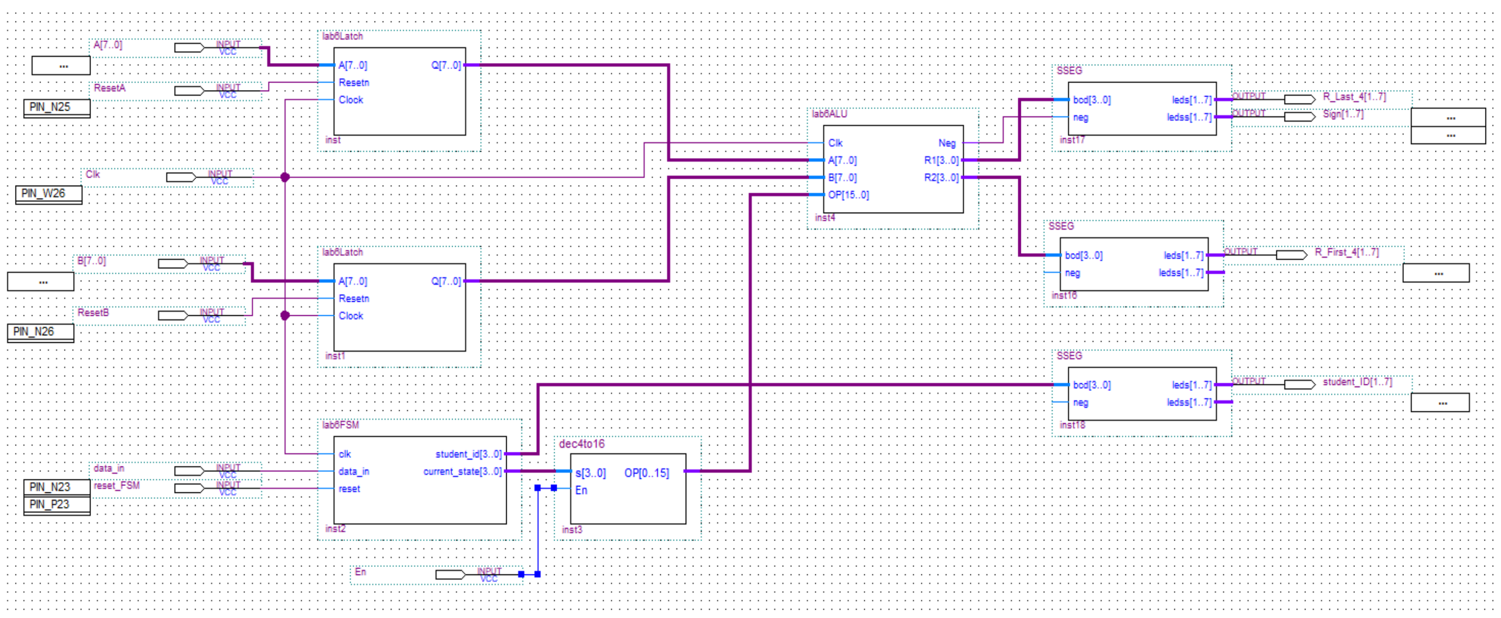
4. ALU_1 for Problem Set 1 of the Lab 6 procedure

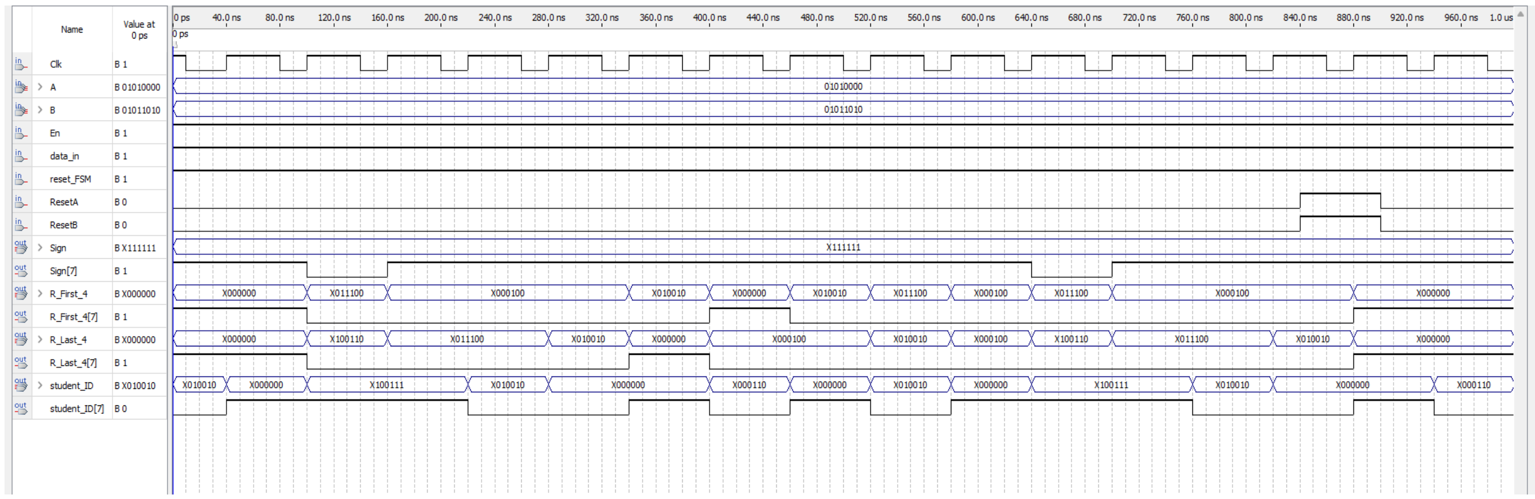




Above is the block diagram and waveform for the ALU. The ALU has 4 inputs: Clock, A, B, and OP (microcode). The ALU switches state with each positive clock edge. There are also 3 outputs: neg, R1, and R2. Neg determines whether the output is negative or not and will display a “-”. R1 represents the lower 4-bits of the result while R2 represents the upper 4-bits. The outputs are sent to a seven segment to be displayed. The function of the ALU is to perform an operation/calculation of the inputs A and B depending on the current state. The current state is determined by the microcode and the operations for each state is included in the table below:

Function #	**Microcode**	**Operation / Function**
1	0000000000000001	sum(A,B)
2	0000000000000010	diff(A,B)
3	0000000000000100	$\overline{A}$
4	0000000000001000	$\overline{A \cdot B}$
5	0000000000010000	$\overline{A + B}$
6	0000000000100000	$A \cdot B$
7	0000000001000000	$A \oplus B$
8	0000000010000000	$A + B$
9	0000000100000000	$\overline{A \oplus B}$



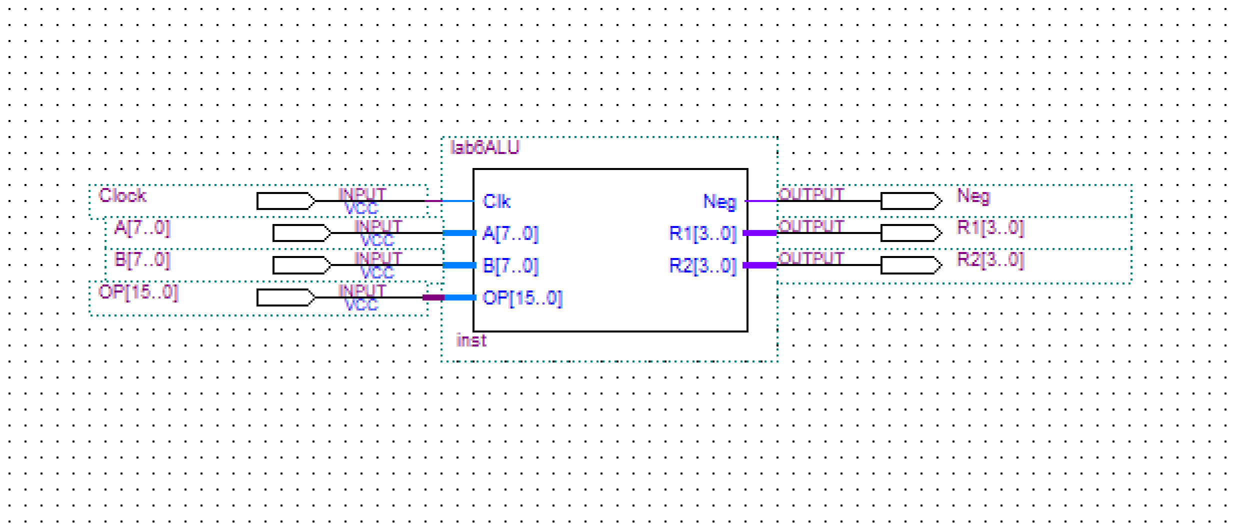


Above is the complete waveform and block diagram. R_First_4 represents the upper 4-bits of the result while R_Last_4 represents the lower 4-bits. In the waveform, the outputs also display the 7th variable separate from the variable group; however, the values are still correct. The outputs are the 8-bit representation of the seven segment which are inverted since the seven segments take the inversion of the abcdef values. The reset_FSM is also inverted as it was programmed to a button; 1 - reset is off, 0 - reset is on.

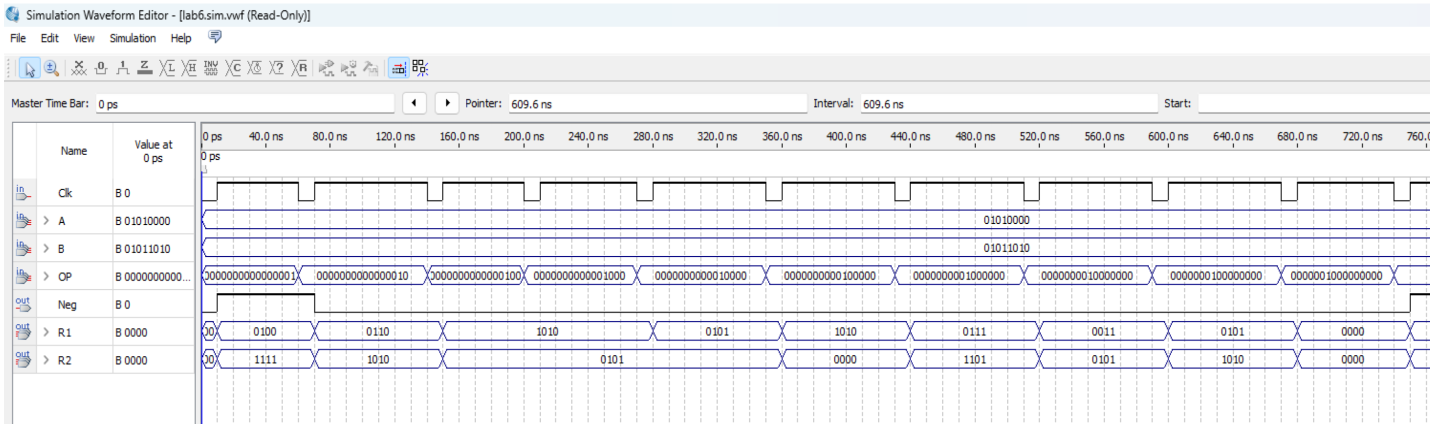
5. ALU_2 for Problem Set 2 of the Lab6 procedure

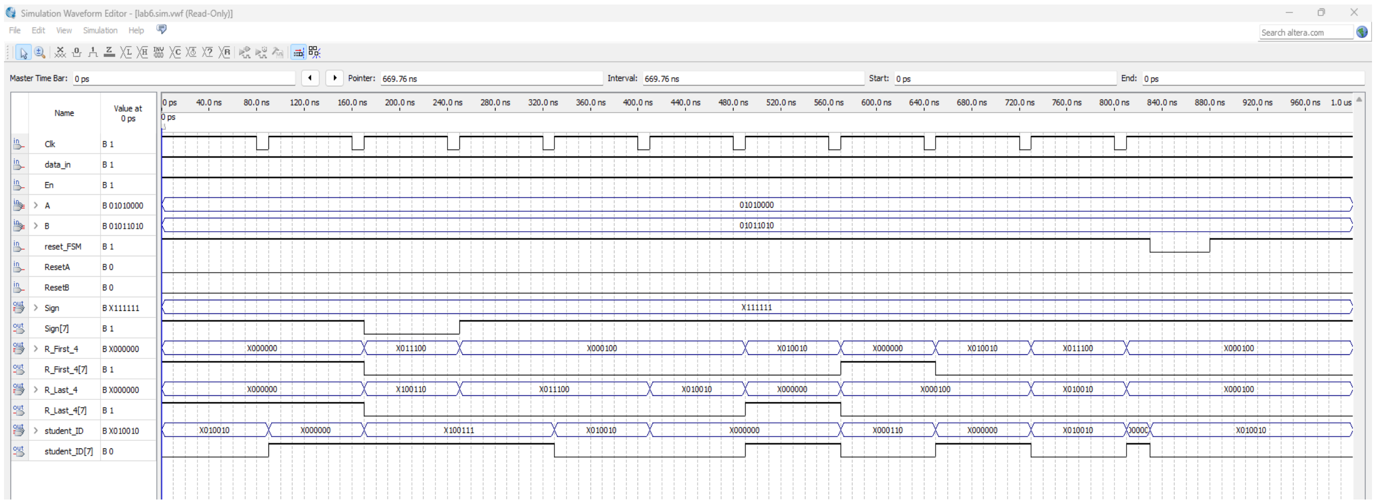
I was assigned modification ***c*** with the functions in the table below:

Function #	Microcode	Operation / Function
1	0000000000000001	Produce the difference between A and B
2	0000000000000010	2 Produce the 2's complement of B
3	0000000000000100	Swap the lower 4 bits of A with lower 4 bits of B
4	0000000000001000	Produce null on the output
5	0000000000010000	Decrement B by 5
6	0000000000100000	Invert the bit-significance order of A
7	0000000001000000	Shift B to left by three bits, input bit = 1 (SHL)
8	0000000010000000	Increment A by 3
9	0000000100000000	Invert all bits of B



Part 2 follows the same design as Part 1 with modifications in the ALU core. Each state performs a different operation which was changed in the VHDL code for the ALU. The same inputs/outputs and decoder microcodes are used as in part 1.





The first waveform above shows the outputs of all 9 of the functions with R2 being the upper 4-bits and R1 being the lower 4-bits. As seen in the waveform, for each positive edge, the output changes depending on the current state (OP). The values for A and B are my student number in binary 8-bit. The second waveform is the complete waveform which instead shows the 8-bit output values for the seven segment. Just like in part 1, reset_FSM and the 8-bit values are inverted.

Below is the modified VHDL code for the Switch/Case statement in the ALU:

```
case OP is
            WHEN "0000000000000001" =>
                --Produce the difference between A and B
                if(Reg2 > Reg1)then
                    Result <= Reg1 + (NOT(Reg2+1));
                    Neg <= '1';
                else
                    Result <= Reg1 - Reg2;
                    Neg <= '0';
```

```
          end if;

WHEN "0000000000000010" =>
          --Produce the 2’s complement of B
          for i in 0 to 7 loop

                    if(j = 0)then--same if j=0, inverts if j=1
                              if(Reg2(i) = '1') then --Result = B until first bit = 1
                                        Result(i) <= '1';
                                        j := 1;
                              else
                                        Result(i) <= '0';
                              end if;

                    elsif(j = 1)then--inverts
                              if(Reg2(i) = '1') then
                                        Result(i) <= '0';
                              else
                                        Result(i) <= '1';
                              end if;

                    end if;

          end loop;
          Neg <= '0';

WHEN "0000000000000100" =>
          --Swap the lower 4 bits of A with lower 4 bits of B
          Result(7 downto 4) <= Reg1(7 downto 4);
          Result(3 downto 0) <= Reg2(3 downto 0);
          Neg <= '0';

WHEN "0000000000001000" =>
          --Produce null on the output
          Result <= null;
          Neg <= '0';

WHEN "0000000000010000" =>
          --Decrement B by 5
          Result <= Reg2 - 5;
```

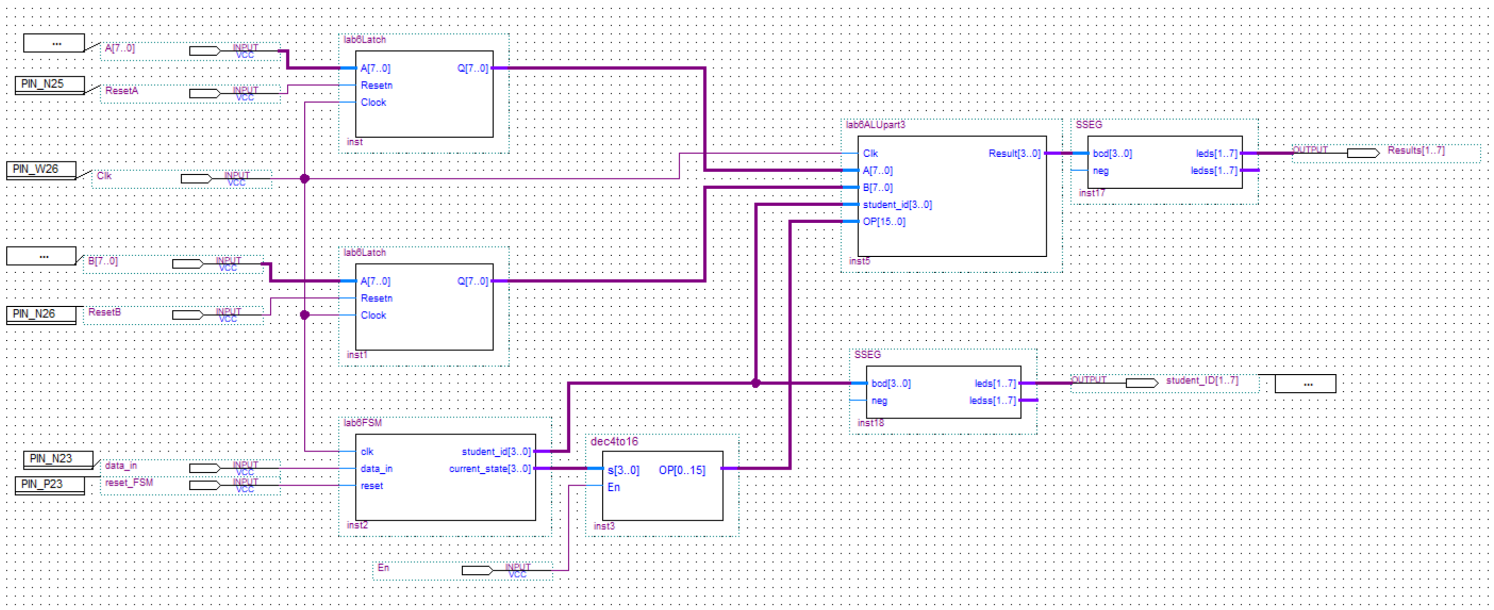
```
        Neg <= '0';

WHEN "0000000000100000" =>
        --Invert the bit-significance order of A
        Result(7) <= Reg1(0);
        Result(6) <= Reg1(1);
        Result(5) <= Reg1(2);
        Result(4) <= Reg1(3);
        Result(3) <= Reg1(4);
        Result(2) <= Reg1(5);
        Result(1) <= Reg1(6);
        Result(0) <= Reg1(7);
        Neg <= '0';

WHEN "0000000001000000" =>
        --Shift B to left by three bits, input bit = 1 (SHL)
        Result <= shift_left(unsigned(Reg2), 3);
        Result(2) <= '1';
        Result(1) <= '1';
        Result(0) <= '1';
        Neg <= '0';

WHEN "0000000010000000" =>
        --Increment A by 3
        Result <= Reg1 + 3;
        Neg <= '0';

WHEN "0000000100000000" =>
        --Invert all bits of B
        for i in 0 to 7 loop
                if(Reg2(i) = '1') then
                        Result(i) <= '0';
                else
                        Result(i) <= '1';
                end if;
        end loop;
        Neg <= '0';

WHEN OTHERS =>
        --Dont care; do nothing
```

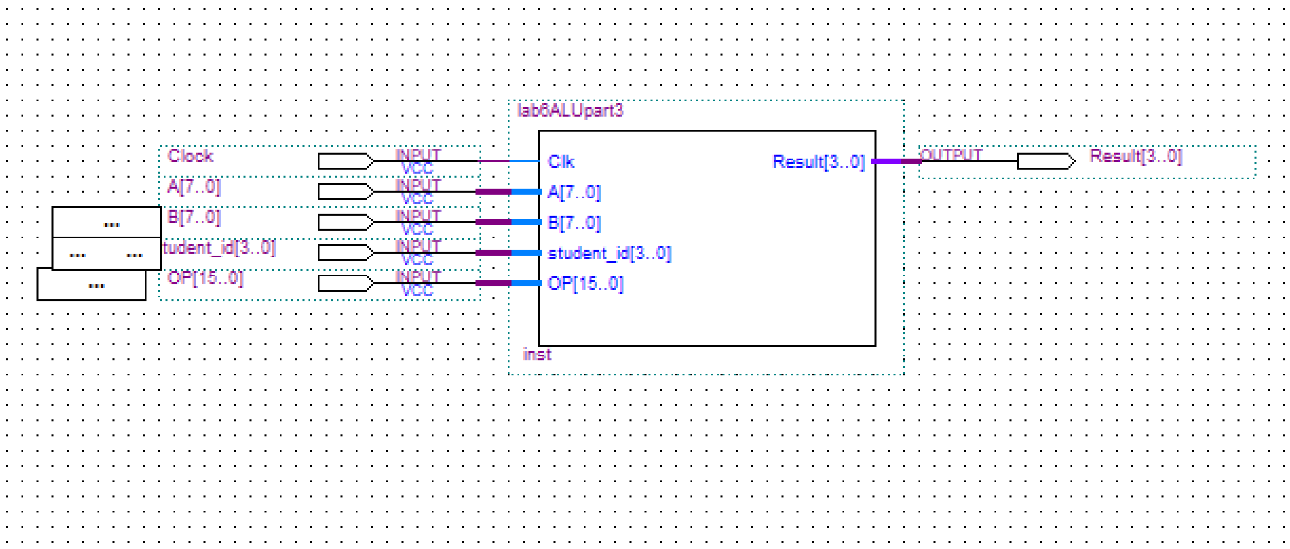
```
                    Result <= "--------";
                    Neg <= '0';

        end case;
```

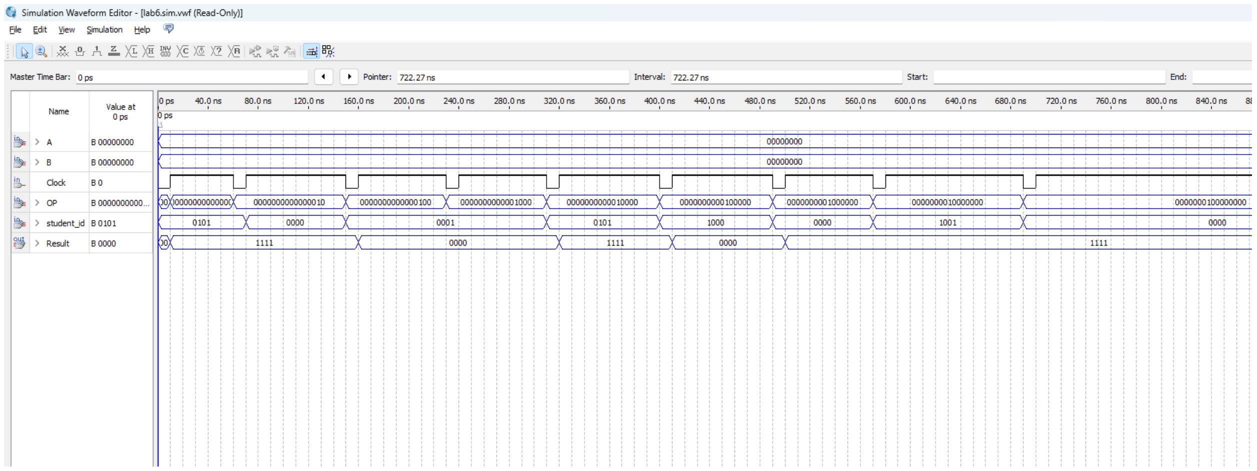
6. ALU_3 for Problem Set 3 of the Lab6 procedure

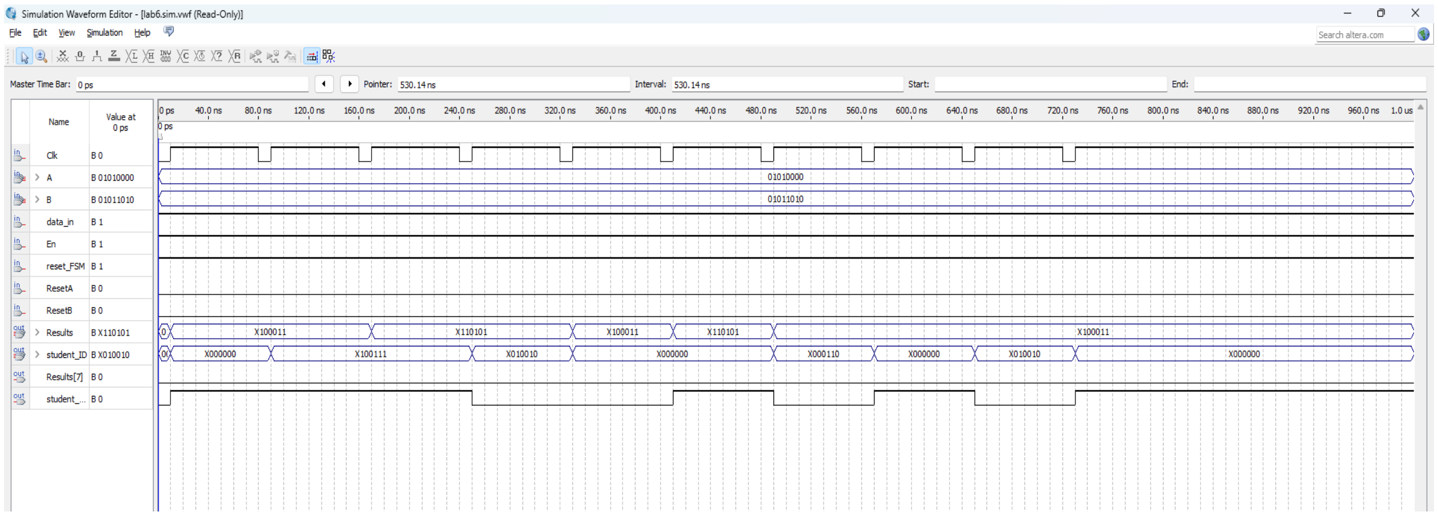
I was assigned modification ***c*** with the following instructions:

For each microcode instruction, display 'y' if the FSM output (student_id) had an odd parity and 'n' otherwise.









Part 3 asks for a modified Control Unit and ALU which will take the student number as an input and display “y” or “n” depending on its parity. Above is the waveform for the modified ALU showing the result as “1111” for “y” and “0000” for “n”. The second waveform is the complete waveform with the 8-bit seven segment displays as the output.

```
for i in 0 to 3 loop        --counts number of 1's
        if(id(i) = '1')then
                j := j + 1;
        end if;
end loop;

if(j mod 2 = 0) then --if even number of 1's; odd parity = true
        Result <= "1111"; -- y
else
```

```
        Result <= "0000"; -- n
    end if;
    j := 0;
```

Here is the VHDL code used to determine whether there is an odd parity bit or not. The loop counts through all values of the 4-bit student number and increments “j” if there is a “1”. If there is an even number of “1”s that means the odd parity bit would be “1”, and “0” if there’s an odd number of “1”s. If the result is “1111” the seven segment will display “y” with the bits “1000110” (The seven segment displays take inverted inputs). If the result is “0000” the seven segment will instead display “n” with the bits “1101010”.

7. Conclusion

In conclusion, our Simple General-Purpose Processor, featuring two latches, four seven-segment displays, an arithmetic logic unit (ALU), a finite state machine (FSM), and a 4x16 decoder, has the ability to perform various calculations and operations with user input. The processor's utilization of eight-bit inputs, representing the last four digits of our student number, as well as an individual part 2 and 3, ensures a personalized system that is unique to our group. The system is able to transfer data to the ALU synchronously through each positive edge clock input when the reset is inactive. The Moore machine-based FSM is able to sequence through nine states, each linked to a different student number digit. Our system also has a visual output representation through the seven-segment display, enhancing accessibility of processed data. The ALU showcases the versatility and uniqueness of our system as it computes different calculations based on state and assigned modifications. This project has served as a valuable learning experience as we were able to implement components from each lab to create a functional general-purpose processor.