COE 328 - LAB 6 - FALL 2023

DESIGN OF A SIMPLE GENERAL-PURPOSE PROCESSOR

DOCUMENTATION REPORT

BILAL MALIK (501164708)

SECTION 01

(PROFESSOR: REZA SEDAGHAT)

(TA: AMMAD SHAH)

TABLE OF CONTENTS

- INTRODUCTION -

This laboratory report explores topics learnt in the course to create an algorithmic logic unit (ALU), using the components: Latch1, Latch2, the 4:16 Decoder, and the Finite State Machine (FSM). Latch1 and Latch2, are known for their ability to store bits of information, play a pivotal role in memory and data retention within the circuit. The 4:16 Decoder, another crucial component, demonstrates its capability in decoding inputs into multiple outputs, illustrating the principles of digital logic expansion. The Finite State Machine (FSM), is examined for its dynamic control and decision-making abilities, essential in complex circuit operations.

- COMPONENTS: LATCH1, LATCH2, 4:16 DECODER, FSM -

Latch 1 and Latch 2 are basic digital storage components, each holding a single bit of data and differing in triggering mechanisms and internal configurations. The 4:16 Decoder is a digital logic circuit that translates a 4-bit input into one of 16 outputs, streamlining complex logic designs in tasks like memory address decoding. The Finite State Machine (FSM) is an abstract model used in both software and hardware design, characterized by a finite number of states.

The student number used for this lab is: 501164708. Therefore, A = $(47)_{16}$ and B = $(08)_{16}$. In binary, A and B are translated to A = $(0010\ 1111)_2$ and B = $(0000\ 1000)_2$.

Latch 1 & 2:

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   ENTITY Latch1 IS
4      PORT( A : IN STD_LOGIC_VECTOR (7 DOWNTO 0); -- 8 bit A input
5            Reset, Clock: IN STD_LOGIC; -- 1 bit clock and reset input
6            Q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
7   END Latch1;
8   ARCHITECTURE Behaviour of Latch1 IS
9   BEGIN
10     PROCESS (Reset, Clock)
11     BEGIN
12        IF Reset = '0' THEN
13              Q <= "00000000";
14           ELSIF Clock 'EVENT AND Clock = '1' THEN
15              Q <= A;
16              END IF;
17        END PROCESS;
18  END Behaviour;
```

Figure 1.0: Latch 1 VHDL Code

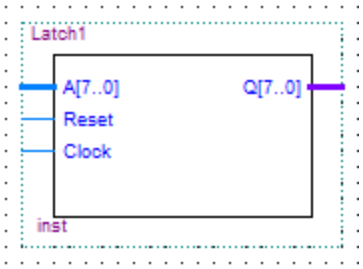


Figure 2.0: Latch 1 Block Symbol

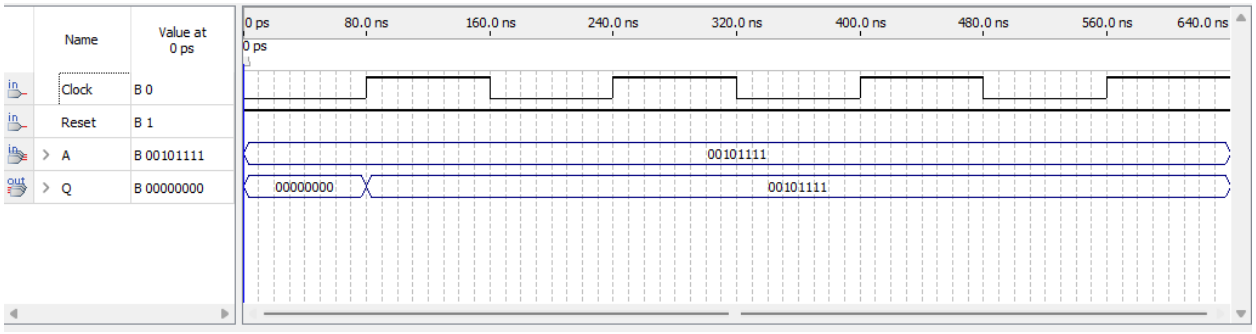


Figure 3.0: Latch 1 Waveform

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   ENTITY latch2 IS
4      PORT( B : IN STD_LOGIC_VECTOR (7 DOWNTO 0); -- 8 bit A input
5            Reset, Clock: IN STD_LOGIC; -- 1 bit clock and reset input
6            Q: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
7   END latch2;
8   ARCHITECTURE Behaviour of Latch2 IS
9   BEGIN
10     PROCESS (Reset, Clock)
11     BEGIN
12        IF Reset = '0' THEN
13              Q <= "00000000";
14           ELSIF Clock 'EVENT AND Clock = '1' THEN
15              Q <= B;
16           END IF;
17        END PROCESS;
18  END Behaviour;
```

Figure 4.0: Latch 2 VHDL Code

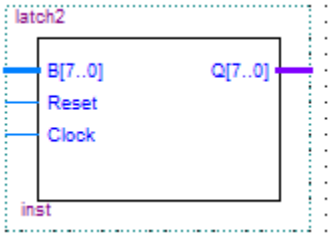


Figure 5.0: Latch 2 Block Symbol

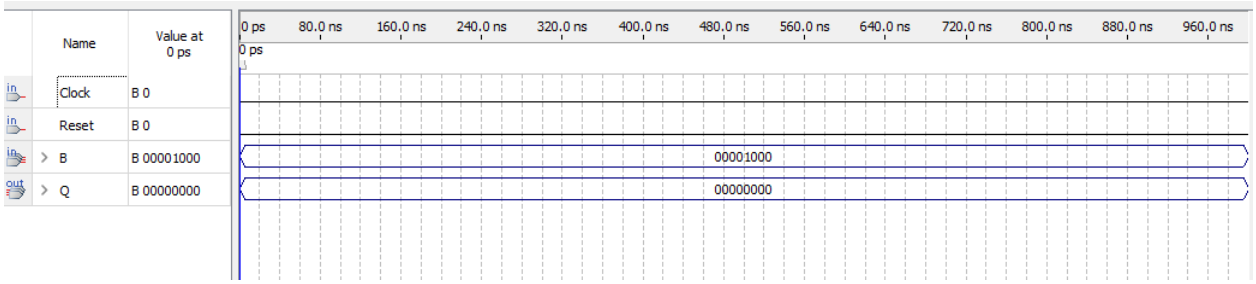


Figure 6.0: Latch 2 Waveform

Table 1.0: Truth Table for Latches 1 & 2

Clk	*D*	*Q(t+1)*
0	*x*	*Q(t)*
1	*0*	*0*
1	*1*	*1*

4:16 Decoder

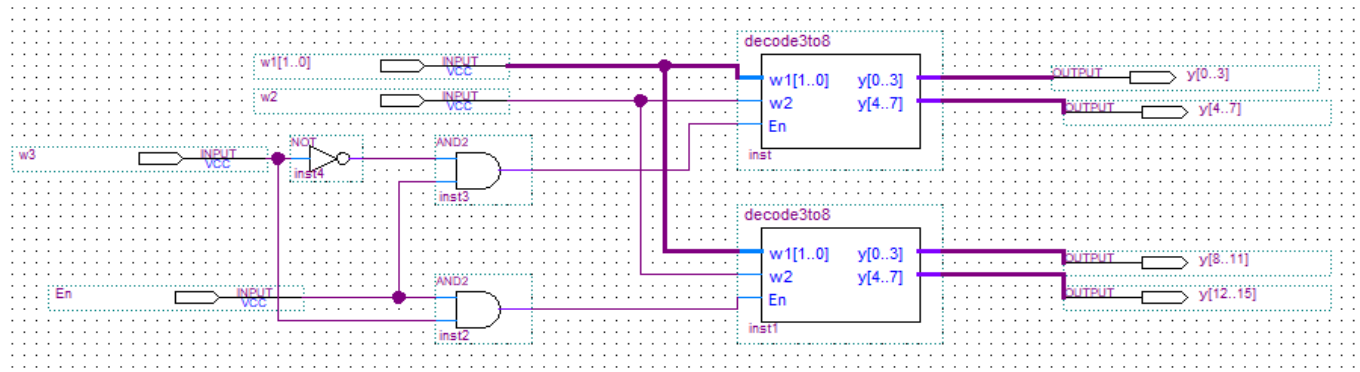


Figure 7.0: 4:16 Decoder Implemented with 2 3:18 Decoders

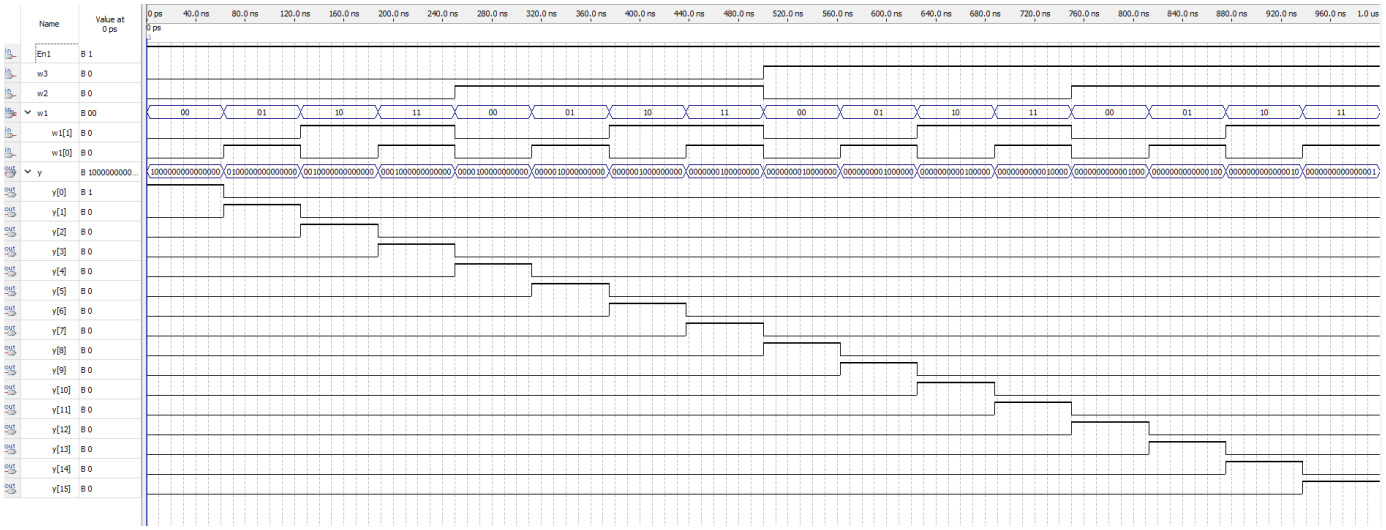


Figure 8.0: 4:16 Decoder Waveform

FSM:

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY moore IS
5   PORT(
6   clk : IN std_logic;
7   data_in : IN std_logic;
8   reset : IN std_logic;
9   student_id : OUT std_logic_vector (3 DOWNTO 0);
10  current_state : OUT std_logic_vector (3 DOWNTO 0)
11  );
12  END ENTITY;===
13
14  ARCHITECTURE fsm OF moore IS
15
16  TYPE state_type IS (s0,s1,s2,s3,s4,s5,s6,s7,s8);
17  SIGNAL yfsm : state_type;
18  SIGNAL data_out : std_logic_vector (3 DOWNTO 0);
19  SIGNAL current_statee : std_logic_vector (3 DOWNTO 0);
20
21  BEGIN
22  PROCESS (clk, reset)
23
24  BEGIN
25  IF reset = '1' THEN
26  yfsm <= s0;
27  ELSIF (clk'EVENT AND clk = '1')THEN
28
29  CASE yfsm is
30
31  WHEN s0 =>
32
33  IF data_in = '1' THEN
34  yfsm <= s1;
35  ELSE
36  yfsm <= s0;
37  END IF;
38
39  WHEN s1 =>
40
41  IF data_in = '1' THEN
42  yfsm <= s2;
43  ELSE
44  yfsm <= s1;
45  END IF;
```

```
47  WHEN s2 =>
48
49  IF data_in = '1' THEN
50  yfsm <= s3;
51  ELSE
52  yfsm <= s2;
53  END IF;
54
55  WHEN s3 =>
56
57  IF data_in = '1' THEN
58  yfsm <= s4;
59  ELSE
60  yfsm <= s3;
61  END IF;
62
63  WHEN s4 =>
64
65  IF data_in = '1' THEN
66  yfsm <= s5;
67  ELSE
68  yfsm <= s4;
69  END IF;
70
71  WHEN s5 =>
72
73  IF data_in = '1' THEN
74  yfsm <= s6;
75  ELSE
76  yfsm <= s5;
77  END IF;
78
79  WHEN s6 =>
80
81  IF data_in = '1' THEN
82  yfsm <= s7;
83  ELSE
84  yfsm <= s6;
85  END IF;
86
87  WHEN s7 =>
88
89  IF data_in = '1' THEN
```

```
89   IF data_in = '1' THEN
90   yfsm <= s8;
91   ELSE
92   yfsm <= s7;
93   END IF;
94
95   WHEN s8 =>
96
97   IF data_in = '1' THEN
98   yfsm <= s0;
99   ELSE
100  yfsm <= s8;
101  END IF;
102
103  END CASE;
104  END IF;
105  END PROCESS;
106  ------------------------------
107  PROCESS (yfsm, data_in)
108  BEGIN
109  CASE yfsm is
110
111  WHEN s0 =>
112  student_id <= "0101"; -- 5
113  current_state <= "0000";
114
115  WHEN s1 =>
116  student_id <= "0000"; -- 0
117  current_state <= "0001";
118
119  WHEN s2 =>
120  student_id <= "0001"; -- 1
121  current_state <= "0010";
122
123  WHEN s3 =>
124  student_id <= "0001"; -- 1
125  current_state <= "0011";
126
127  WHEN s4 =>
128  student_id <= "0110"; -- 6
129  current_state <= "0100";
```

```
127  WHEN s4 =>
128  student_id <= "0110"; -- 6
129  current_state <= "0100";
130
131  WHEN s5 =>
132  student_id <= "0100"; -- 4
133  current_state <= "0101";
134
135  WHEN s6 =>
136  student_id <= "0111"; -- 7
137  current_state <= "0110";
138
139  WHEN s7 =>
140  student_id <= "0000"; -- 0
141  current_state <= "0111";
142
143  WHEN s8 =>
144  student_id <= "1000"; -- 8
145  current_state <= "1000";
146
147  END CASE;
148  END PROCESS;
149  END ARCHITECTURE;
```

Figure 9.0: Moore VHDL Code

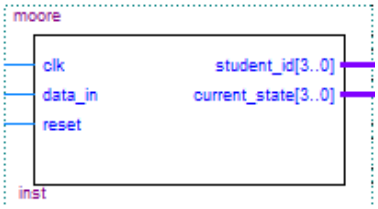


Figure 10.0: Moore FSM Block Symbol

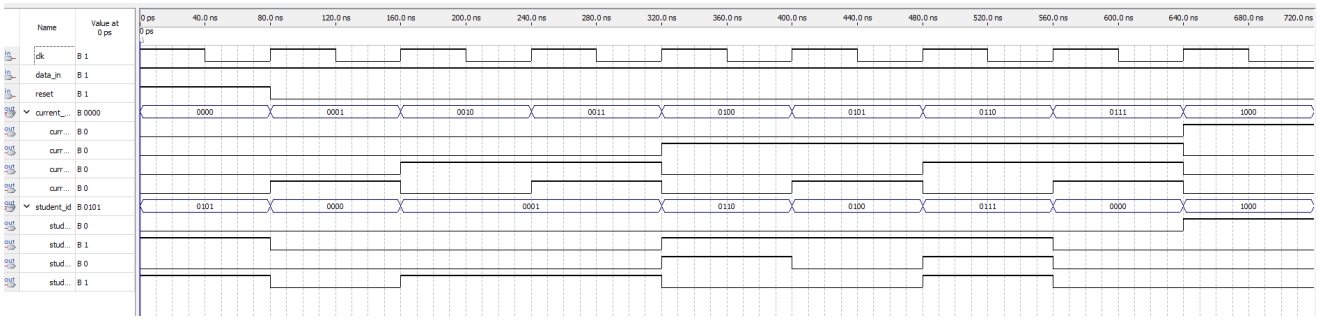


Figure 11.0: FSM Waveform

Table 2.0: Truth Table for Moore Machine

Present State	Next State		Output: Student ID
	Data IN = 0	Data IN = 1	
0000	0000	0001	0101
0001	0001	0010	0000
0010	0010	0011	0001
0011	0011	0100	0001
0100	0100	0101	0110
0101	0101	0110	0100
0110	0110	0111	0110
0111	0111	1000	0000
1000	1000	0000	1000

Seven Segment Display:

The seven segment display is used to display values from 0-15 in hexadecimal code and accounts for a scenario where the resultant output is negative. Multiple 7-segment displays was used as the inputs varied in the student ID due to the FSM and AlU results. The ALU requires two seven segment displays to operate the 8-bit output using two 4-bit inputs.

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY sseg IS
5
6   PORT (bcd: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
7   neg: IN STD_LOGIC;
8   leds: OUT STD_LOGIC_VECTOR(0 TO 6);
9   ledss: OUT STD_LOGIC_VECTOR(0 TO 6) );
10
11  END sseg;
12
13  ARCHITECTURE Behavior OF sseg IS
14  BEGIN
15
16  PROCESS(bcd)
17  BEGIN
18
19  CASE bcd IS -- abcdefg
20
21  WHEN "0000" => leds <= "1111110";
22  WHEN "0001" => leds <= "0110000";
23  WHEN "0010" => leds <= "1101101";
24  WHEN "0011" => leds <= "1111001";
25  WHEN "0100" => leds <= "0110011";
26  WHEN "0101" => leds <= "1011011";
27  WHEN "0110" => leds <= "1011111";
28  WHEN "0111" => leds <= "1110000";
29  WHEN "1000" => leds <= "1111111";
30  WHEN "1001" => leds <= "1111011";
31  -- 10 to 15 in HEX
32  WHEN "1010" => leds <= "1110111"; -- A
33  WHEN "1011" => leds <= "0011111"; -- B
34  WHEN "1100" => leds <= "1001110"; -- C
35  WHEN "1101" => leds <= "0111101"; -- D
36  WHEN "1110" => leds <= "1001111"; -- E
37  WHEN "1111" => leds <= "1000111"; -- F
38
39  END CASE;
40  END PROCESS;
41
42  PROCESS (neg)
43  BEGIN
44  IF (neg = '1') THEN
45  ledss <= "0000001";
46  ELSE
47  ledss <= "0000000";
48
49  END IF;
50  END PROCESS;
51  END Behavior;
```

Figure 12.0: Seven Segment Display VHDL Code

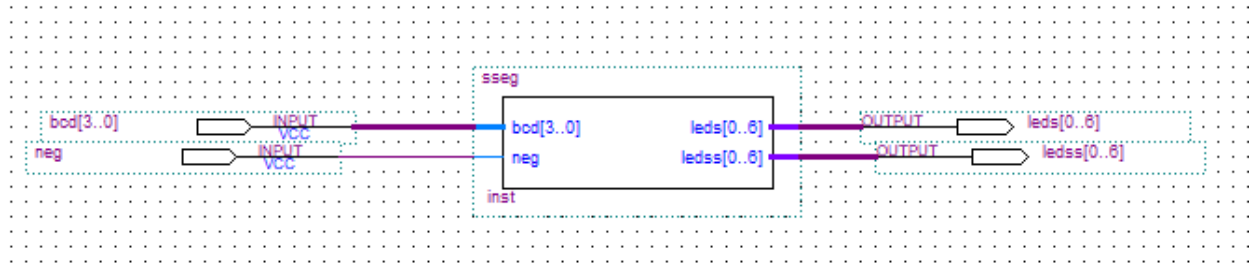


Figure 13.0: Seven Segment Display Block Diagram

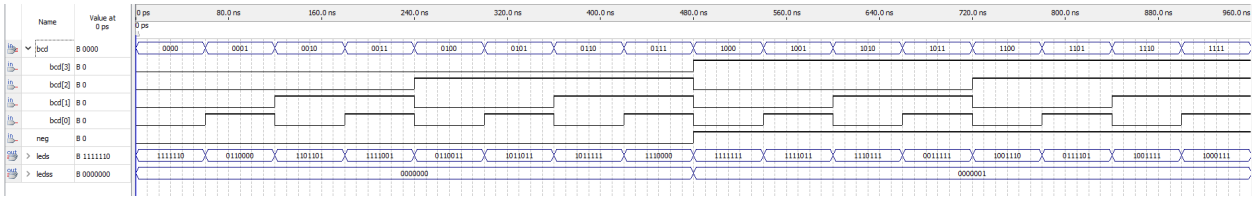


Figure 14.0: Seven Segment Display Waveform

Table 3.0: Truth Table for Seven Segment Display: bcd input

bcd(input)	leds(abcdefg)
0000	1111110
0001	0110000
0010	1101101
0100	1111001
0101	0110011
0110	1011011
0111	1011111
1000	1110000
1001	1111111
1010	1111011
1011	0011111
1100	1001110
1101	0111101
1110	1001111
1111	1000111

Table 4.0: Truth Table for Seven Segment Display: neg input

neg	ledss(abcdefg)
0	0000000
1	0000001

- ALU FOR PROBLEM SET 1 -

The purpose of the ALU component is to perform operations based on the many states of the FSM. The nine operations include addition, subtraction as well as Boolean operations. The ALU component was made by adding to a skeleton code provided in the lab manual, and having 5 inputs: Clock, A, B, student_ID, and OP.

The clock input is shared between all components except the 7-segment display and the 4:16 decoder. The clock alternates between 0 and 1, and when a rising edge occurs, the ALU will check OP, which is the output of the 4:16 decoder. This will correspond to a case statement and perform the operation in inputs A and B. The ALU has 3 outputs (Neg, R1 and R2). Neg refers to a negative output of the seven segment display. As the seven segment display is a 4-bit input but the ALU works in 8-bit, R1 and R2 are used to split the result into two separate 4-bit outputs. Combined, it would become the expected 8-bit output.

```
1   library ieee;
2   USE ieee.STD_LOGIC_1164.ALL;
3   USE ieee.STD_LOGIC_UNSIGNED.ALL;
4   USE ieee.NUMERIC_STD.ALL;
5
6   ENTITY ALU IS
7   PORT(Clk:IN STD_LOGIC;
8   A,B : IN UNSIGNED(7 DOWNTO 0);
9   student_id : IN UNSIGNED(3 DOWNTO 0);
10  OP : IN UNSIGNED(15 DOWNTO 0);
11  Neg : OUT STD_LOGIC;
12  R1: OUT UNSIGNED(3 DOWNTO 0);
13  R2: OUT UNSIGNED(3 DOWNTO 0));
14  END ALU;
15  ARCHITECTURE calculation of ALU IS
16  SIGNAL Reg1,Reg2,Result : UNSIGNED(7 DOWNTO 0) :=(OTHERS=> '0');
17  BEGIN
18  Reg1 <= A;
19  Reg2 <= B;
20  PROCESS(Clk,OP)
21  BEGIN
22  IF(rising_edge(Clk)) THEN
23  CASE OP IS
24
25  WHEN "0000000000000001" => Result <= Reg1 + Reg2;
26
27  WHEN "0000000000000010" =>
28
29  IF (Reg2>Reg1) THEN
30  Result<= (Reg1+(NOT Reg2+1));
31  Neg<='1';
32  ELSE
33  Result<= (Reg1-Reg2);
34  Neg<='0';
35  END IF;
36  WHEN "0000000000000100" => Result <= NOT Reg1;
37   Neg<='0';
38
39  WHEN "0000000000001000" => Result <= (NOT (Reg1 AND Reg2));
40    Neg<='0';
41
```

```
39    WHEN "0000000000001000" => Result <= (NOT (Reg1 AND Reg2));
40      Neg<='0';
41
42    WHEN "0000000000010000" => Result <= (NOT (Reg1 OR Reg2));
43     Neg<='0';
44
45    WHEN "0000000000100000" => Result <= (Reg1 AND Reg2);
46    Neg<='0';
47
48    WHEN "0000000001000000" => Result <= Reg1 OR Reg2;
49    Neg<='0';
50
51    WHEN "0000000010000000" => Result <= (Reg1 XOR Reg2);
52    Neg<='0';
53
54    WHEN "0000000100000000" => Result <= (Reg1 XNOR Reg2);
55    Neg<='0';
56
57    WHEN OTHERS =>
58    Result<= "--------";
59
60  END CASE;
61  END IF;
62  END PROCESS;
63
64  R1 <= Result(3 DOWNTO 0);
65  R2 <= Result(7 DOWNTO 4);
66  END calculation;
```

Figure 15.0: ALU VHDL Code

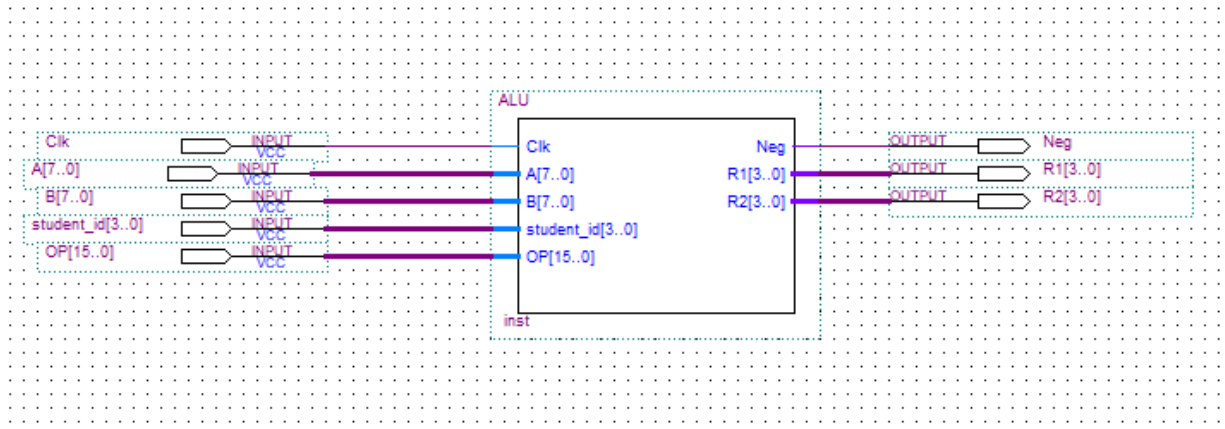


Figure 16.0: ALU Block Diagram

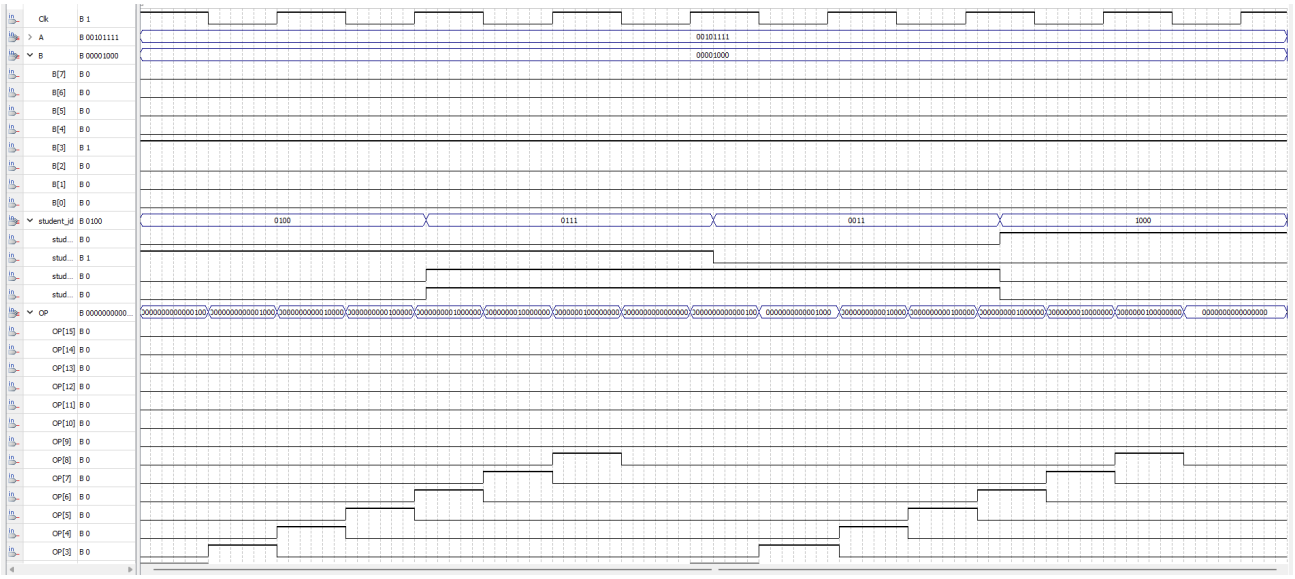


Figure 17.0: ALU Waveform

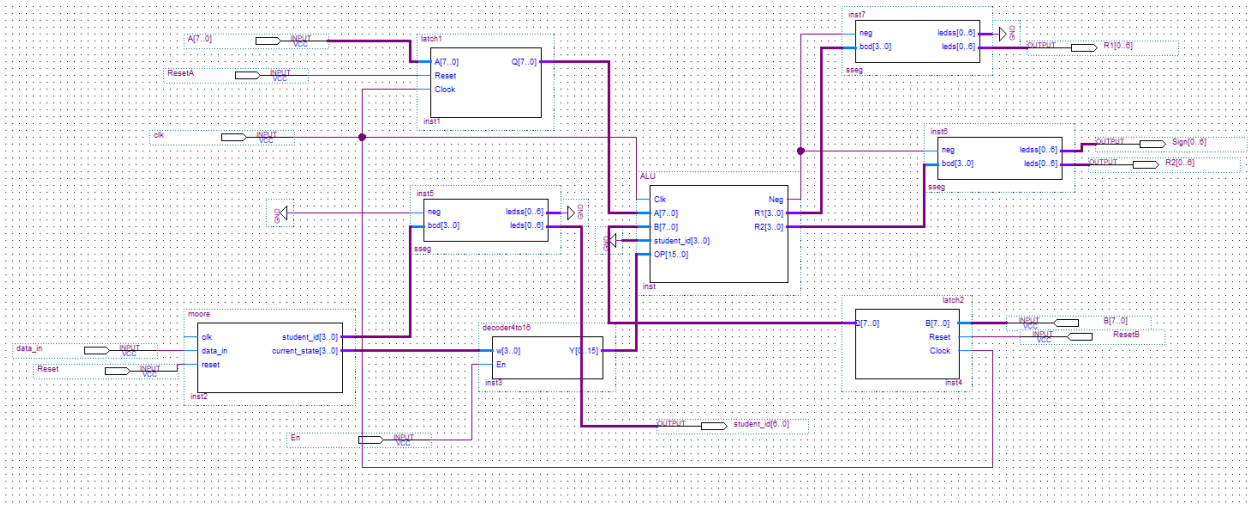


Figure 18.0: Combined ALU Block Diagram

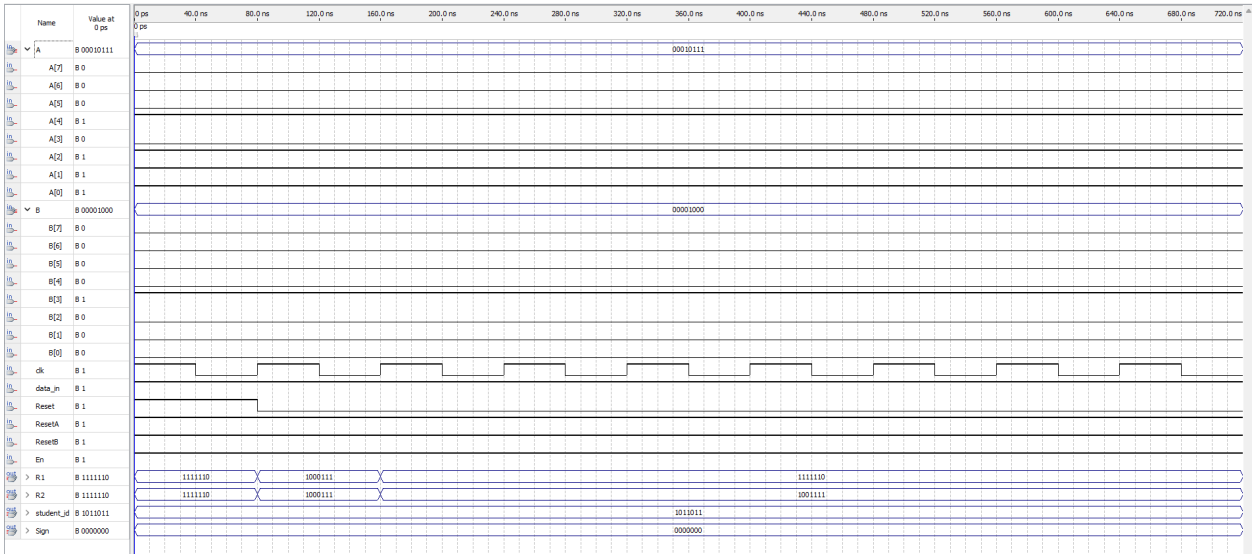
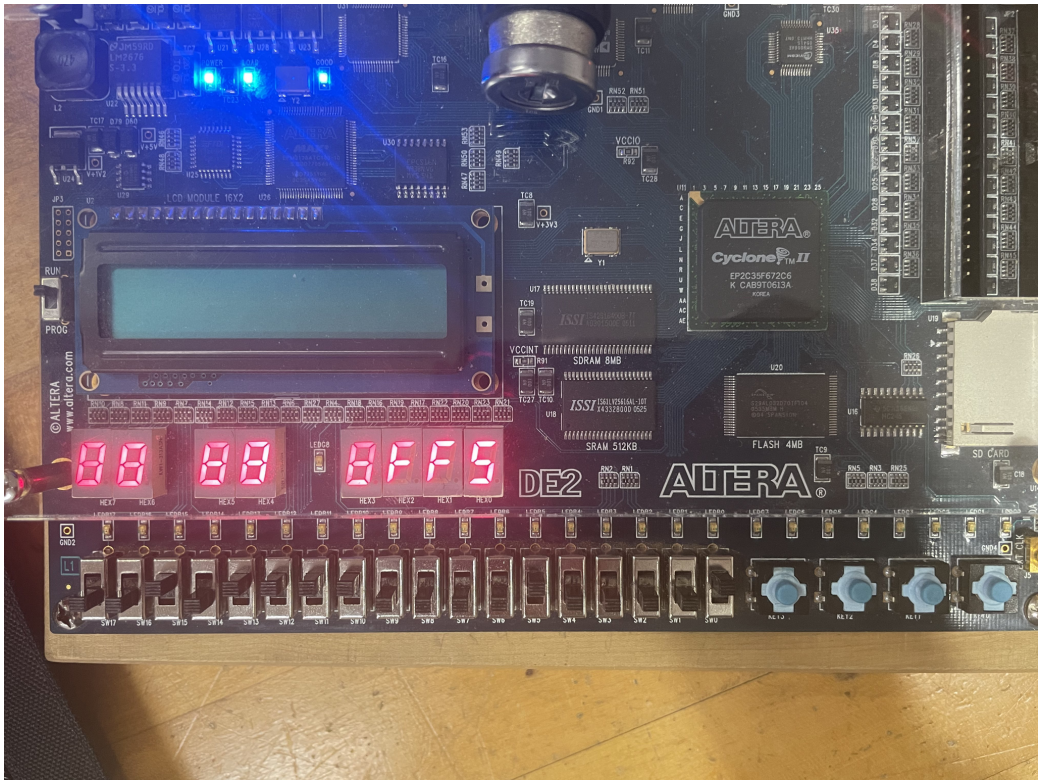


Figure 19.0: Combined ALU Waveform



Error with the demo. Outputs were inverted

- ALU FOR PROBLEM SET 2 -

The ALU has the same inputs and outputs from the previous part. Inputs A and B are the 8-bit inputs stored in the latches when data_in is 1. The clock input is shared between all sequential circuit components including the FSM, latches, and ALU. When a rising edge occurs (0 to 1), OP determines what operation it will perform, based on the microcode since OP is the output of the 4:16 decoder. R1 and R2 represent the outputs of the ALU. For problem 2, function set a) was selected.

a)

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

Figure 20.0: Function Set for Problem 2

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;
ENTITY ALU2 IS
PORT (Clock : IN STD_LOGIC ;
      A ,B : IN UNSIGNED(7 DOWNTO 0);
      Student_id: IN UNSIGNED(3 DOWNTO 0);
      OP: IN UNSIGNED(15 DOWNTO 0);
      Neg: OUT STD_LOGIC;
      R1: OUT UNSIGNED(3 DOWNTO 0);--first 4bits of 8bits Result
      R2: OUT UNSIGNED(3 DOWNTO 0));--later 4bits of 8bits Result
END ALU2;

ARCHITECTURE Behavior OF ALU2 IS
SIGNAL Reg1, Reg2, Result: UNSIGNED(7 DOWNTO 0) := (OTHERS => '0');
SIGNAL Reg4: UNSIGNED(0 TO 7);

BEGIN
Reg1 <= A;
Reg2 <= B;

PROCESS (Clock,OP,A,B)
BEGIN
   IF (rising_edge(Clock)) THEN
      CASE OP IS
         WHEN "0000000000000001" => --funtion 1 increment A by 2
            neg <= '0';
            Result <= (Reg1 + "00000010");
         WHEN "0000000000000010"  => --function 2 Shift B to the right by 2 bits
            neg <= '0';
            Result(0) <= Reg2(7);
            Result(1) <= Reg2(6);
            Result(2) <= Reg2(0);
            Result(3) <= Reg2(1);
            Result(4) <= Reg2(2);
            Result(5) <= Reg2(3);
            Result(6) <= Reg2(4);
            Result(7) <= Reg2(5);
         WHEN "0000000000000100" =>--Function 3 Shift A to the right by 4 bits
            neg <= '0';
            Result(0) <= Reg1(4);
            Result(1) <= Reg1(5);
            Result(2) <= Reg1(6);
            Result(3) <= Reg1(7);
            Result(4) <= Reg1(0);
            Result(5) <= Reg1(1);
            Result(6) <= Reg1(2);
            Result(7) <= Reg1(3);
         WHEN "0000000000001000" =>--Function 4 Find the smaller value of A and B and pr
            neg <= '0';
            if (Reg1 < Reg2) then
               Result <= Reg1;
            else
               Result <= Reg2;
            end if;
         WHEN "0000000000010000" =>--Function 5 Rotate A to the right by 2 bits
            neg <= '0';
            Result <= (Reg1 ror 2);
         WHEN "0000000000100000" =>--Function 6 Invert the bit-significance order of B
            neg <= '0';
            Result <= Reg2(0)&Reg2(1)&Reg2(2)&Reg2(3)&Reg2(4)&Reg2(5)&Reg2(6)&Reg2(7);
         WHEN "0000000001000000" =>--Function 7 XORing A and B
            neg <= '0';
            Result <= (Reg1 XOR Reg2);
         WHEN "0000000010000000" =>--Function 8 Produce the summation of A and B, then c
            neg <= '0';
            Result <= ((Reg1 + Reg2)-"00000100");
         WHEN "0000000100000000" =>--Function 9 XNOR
            neg <= '0';
            Result <= ("11111111");
         WHEN OTHERS =>-- Don't care
      END CASE;
   END IF;
END PROCESS;

R1 <= Result(3 DOWNTO 0);
R2 <= Result(7 DOWNTO 4);

END Behavior;
```

Figure 21.0: ALU2 VHDL Code

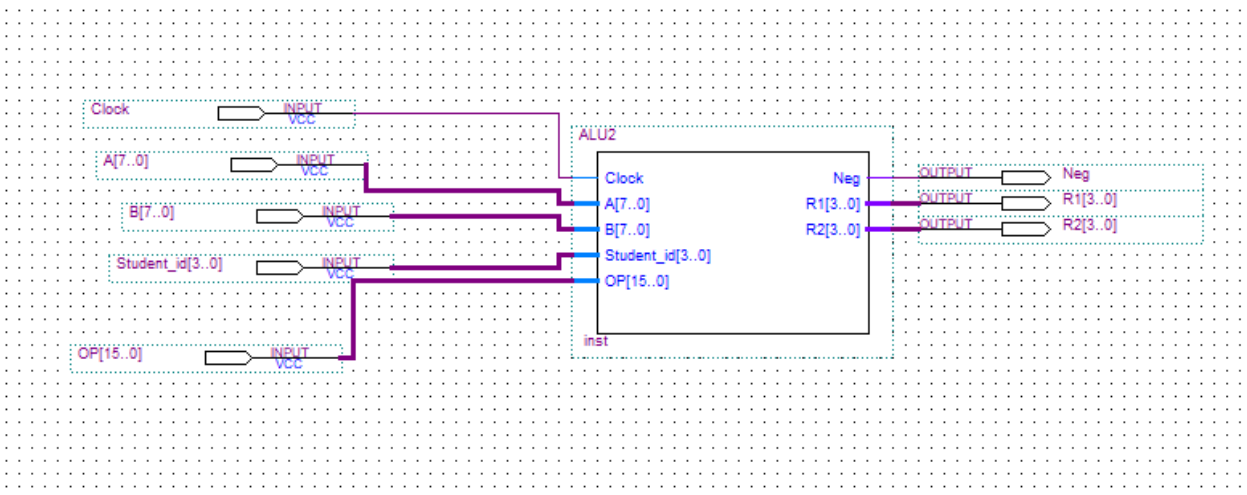


Figure 22.0: ALU2 Block Diagram

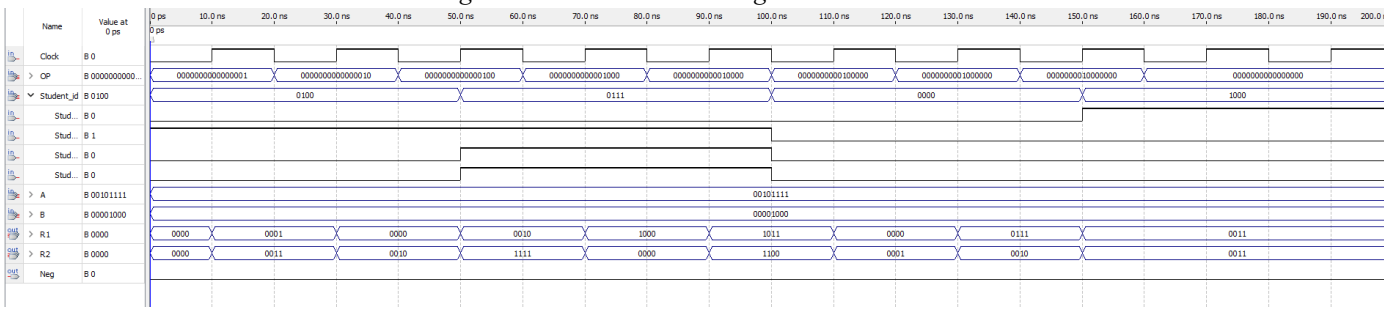


Figure 23.0: Waveform for ALU2

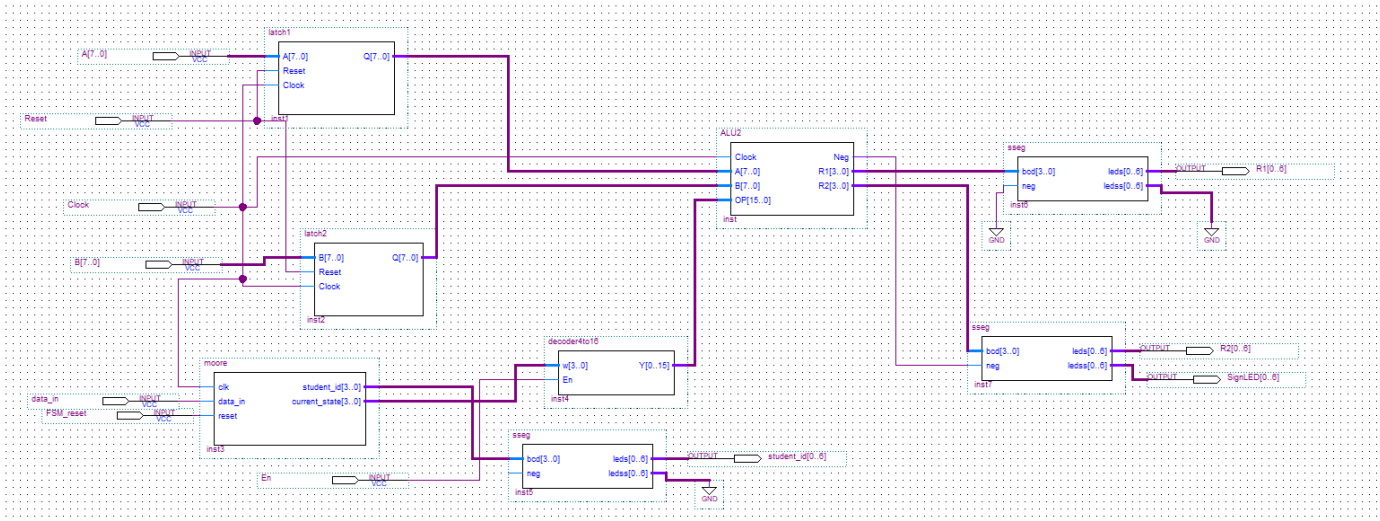


Figure 24.0: ALU2 Combined Block Diagram

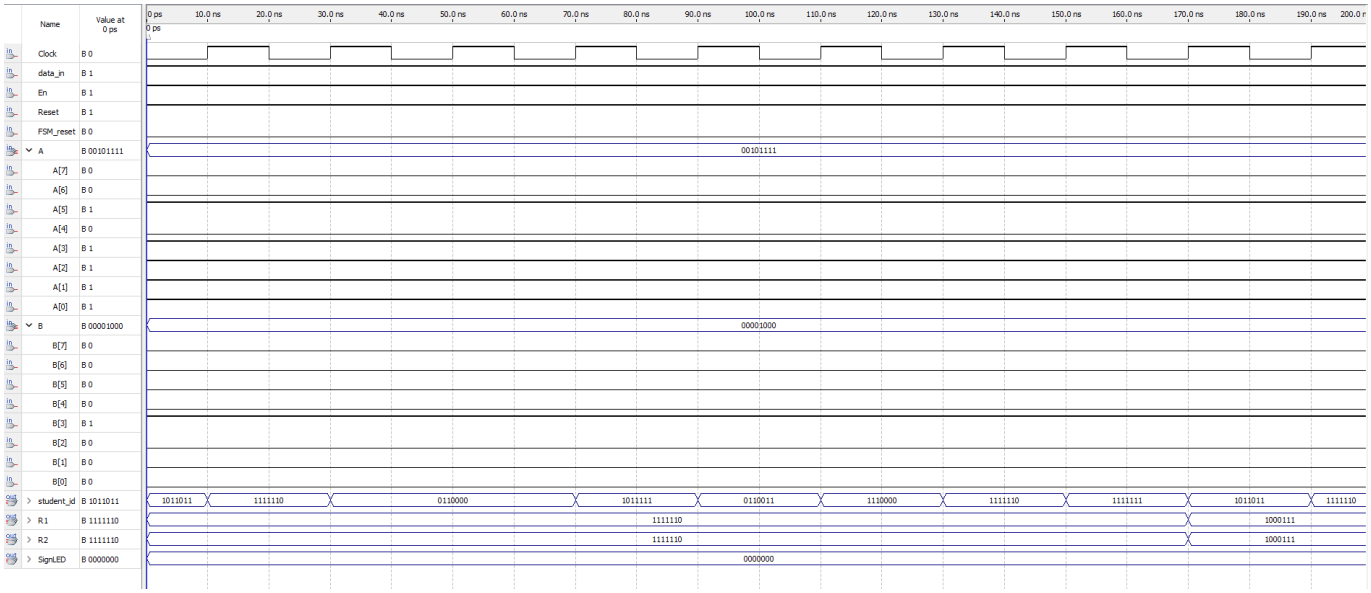
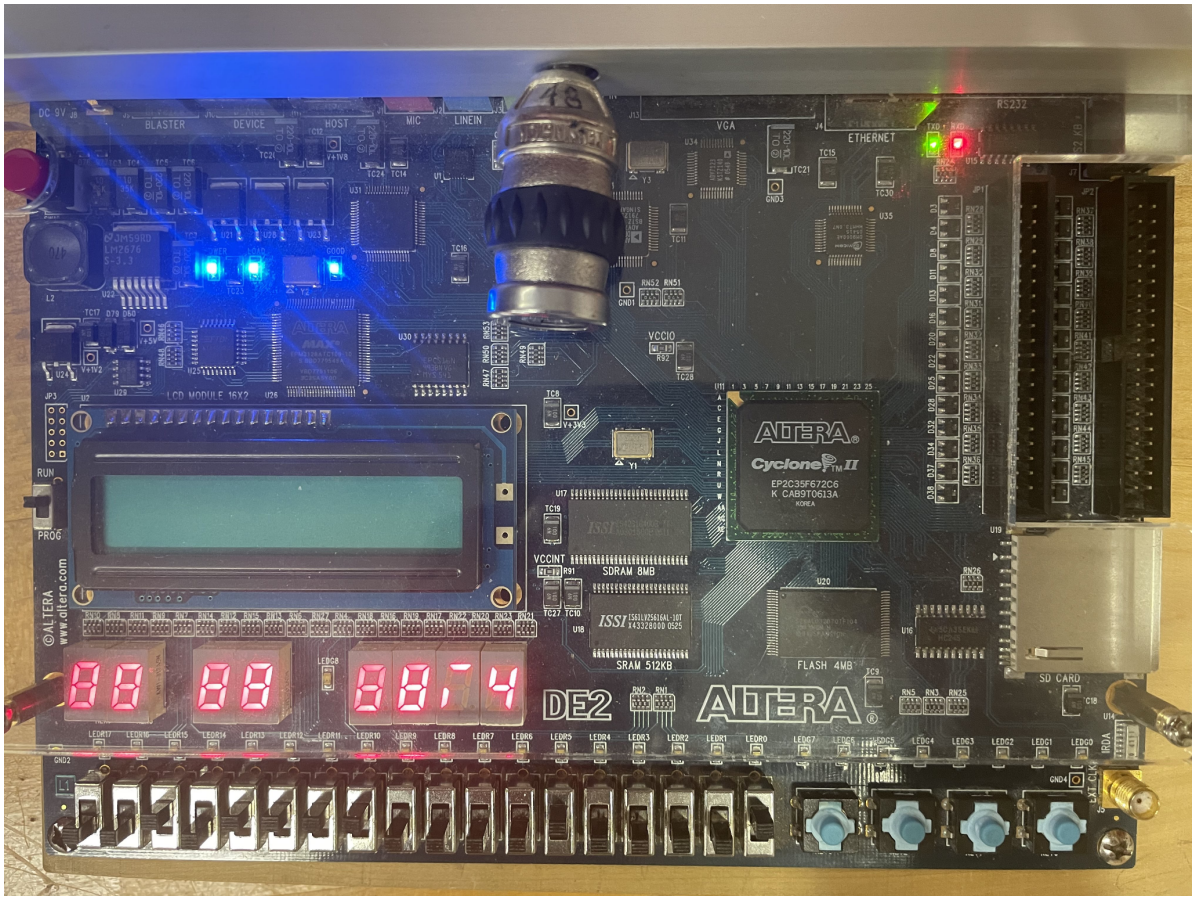


Figure 23.0: Waveform for Combined ALU2



Expected outputs were inverted in the demo.

- ALU FOR PROBLEM SET 3 -

For problem 3, the final outputs will yield a ‘y’ or ‘n’ in the seven segment display depending on the condition chosen. In this case, if two digits from input ‘B’ are greater than the student_id, the result will be ‘y’, otherwise, it would be ‘n’. The seven segment display code was modified in this scenario to output ‘y’ or ‘n’. There will be five inputs in this ALU, clock, A, B, student_id, and OP. Just like in previous parts, the 8-bit inputs of A and B are stored in the latches if data_in is 1. OP is the output of the 4:16 decoder that determines which operation is going to occur based on the states of the moore machine. There is only one output however, as we are no longer using both A and B to perform the calculations, thus not needing a R1 and R2 split to display the 8-bit result. The output of the ALU will either be 1 or 0, which is determined by the condition. Based on this value, the seven segment display will use the output as an input to display either ‘y’ or ‘n’ when it is 1 or 0.

i) For each microcode instruction, 'y' if one of the 2 digits of B are less than FSM output (**student_id**) and 'n' otherwise. Use the microcode instruction from part 1 of the lab.

Figure 24.0: Function Set for Problem 2

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_UNSIGNED.ALL;
4  use IEEE.NUMERIC_STD.ALL;
5  entity ALU3 is
6  port(Clock : in std_logic;
7      A,B : in unsigned(7 downto 0);
8      OP : in unsigned(0 to 15);
9      student_id : in unsigned(3 DOWNTO 0);
10     Neg: out std_logic;
11     R1 : out unsigned(3 downto 0);
12     R2 : out unsigned(3 downto 0));
13 end ALU3;
14 architecture calculation of Alu3 is
15 signal Reg1,Result : unsigned(7 downto 0):=(others => '0');
16 SIGNAL Reg4,Reg5 : UNSIGNED (0 to 3);
17 begin
18      Reg1 <= A;
19      Reg4 <= B(7 DOWNTO 4); -- Higher digit of B
20      Reg5 <= B(3 DOWNTO 0); -- Lower digit of B
21
22 process(Clock, OP)
23 begin
24
25
26 IF(rising_edge(Clock)) THEN
27 CASE OP IS
28
29 WHEN "1000000000000000" =>
30
31   IF (Reg4 < student_id OR Reg5 < student_id) THEN
32   Result <= "00000001"; -- Y
33   ELSE
34   Result <= "00000000"; -- N
35   END IF;
36
37 WHEN "0100000000000000" =>
38
```

```
37 WHEN "0100000000000000" =>
38
39 IF (Reg4 < student_id OR Reg5 < student_id) THEN
40  Result <= "00000001"; -- Y
41  ELSE
42  Result <= "00000000"; -- N
43  END IF;
44
45
46 WHEN "0010000000000000" =>
47
48 IF (Reg4 < student_id OR Reg5 < student_id) THEN
49  Result <= "00000001"; -- Y
50  ELSE
51  Result <= "00000000"; -- N
52  END IF;
53
54 WHEN "0001000000000000" =>
55
56 IF (Reg4 < student_id OR Reg5 < student_id) THEN
57  Result <= "00000001"; -- Y
58  ELSE
59  Result <= "00000000"; -- N
60  END IF;
61
62 WHEN "0000100000000000" =>
63
64 IF (Reg4 < student_id OR Reg5 < student_id) THEN
65  Result <= "00000001"; -- Y
66  ELSE
67  Result <= "00000000"; -- N
68  END IF;
69
70
71 WHEN "0000010000000000" =>
72
73 IF (Reg4 < student_id OR Reg5 < student_id) THEN
74  Result <= "00000001"; -- Y
75  ELSE
```

```
70
71   WHEN "0000010000000000" =>
72
73   IF (Reg4 < student_id OR Reg5 < student_id) THEN
74    Result <= "00000001"; -- Y
75   ELSE
76    Result <= "00000000"; -- N
77    END IF;
78
79
80   WHEN "0000001000000000" =>
81
82   IF (Reg4 < student_id OR Reg5 < student_id) THEN
83    Result <= "00000001"; -- Y
84   ELSE
85    Result <= "00000000"; -- N
86    END IF;
87
88   WHEN "0000000100000000" =>
89
90   IF (Reg4 < student_id OR Reg5 < student_id) THEN
91    Result <= "00000001"; -- Y
92   ELSE
93    Result <= "00000000"; -- N
94    END IF;
95
96   WHEN OTHERS =>
97   Result <="--------"; --n
98
99
100  END CASE;
101  END IF;
102  END PROCESS;
103
104  R1 <= Result(3 DOWNTO 0);
105   END calculation;
```

Figure 25.0: ALU3 VHDL Code

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4
5   ENTITY ssegmodified IS
6   PORT(
7    bcd: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
8   leds: OUT  STD_LOGIC_VECTOR(0 TO 6));
9    END ssegmodified;
10
11  ARCHITECTURE Behaviour OF ssegmodified IS
12  BEGIN
13  PROCESS(bcd)
14   BEGIN
15  CASE bcd IS -- abcdefg
16   WHEN "0000" => leds <= "0010101"; -- n
17   WHEN "0001" => leds <= "0111011"; -- y
18   WHEN OTHERS => leds <= "--------";
19  END CASE;
20   END PROCESS;
21
22   END Behaviour;
```

Figure 26.0: Modified Seven Segment Display VHDL Code

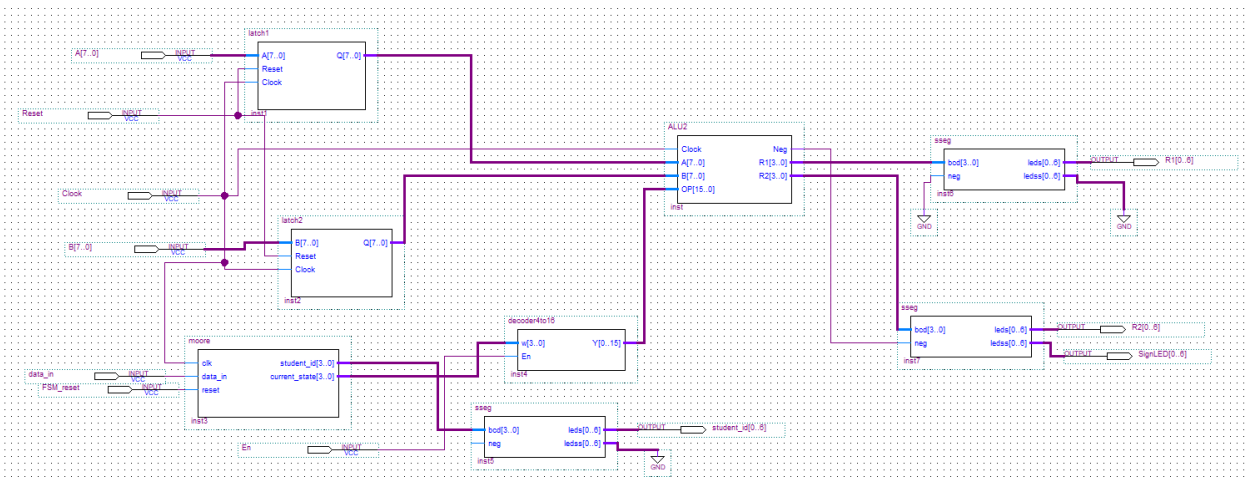


Figure 27.0: ALU3 Combined Block Diagram

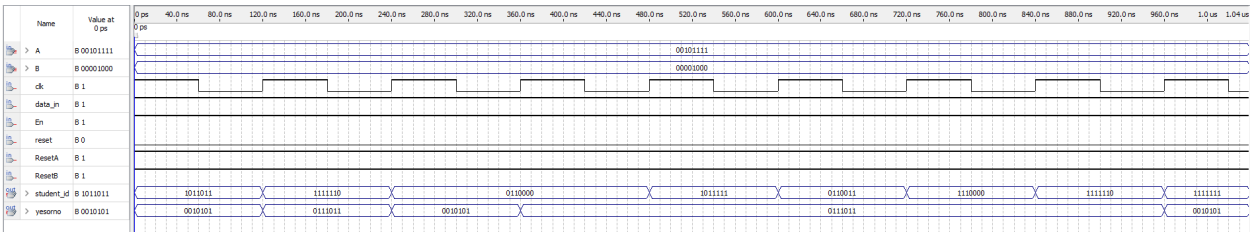
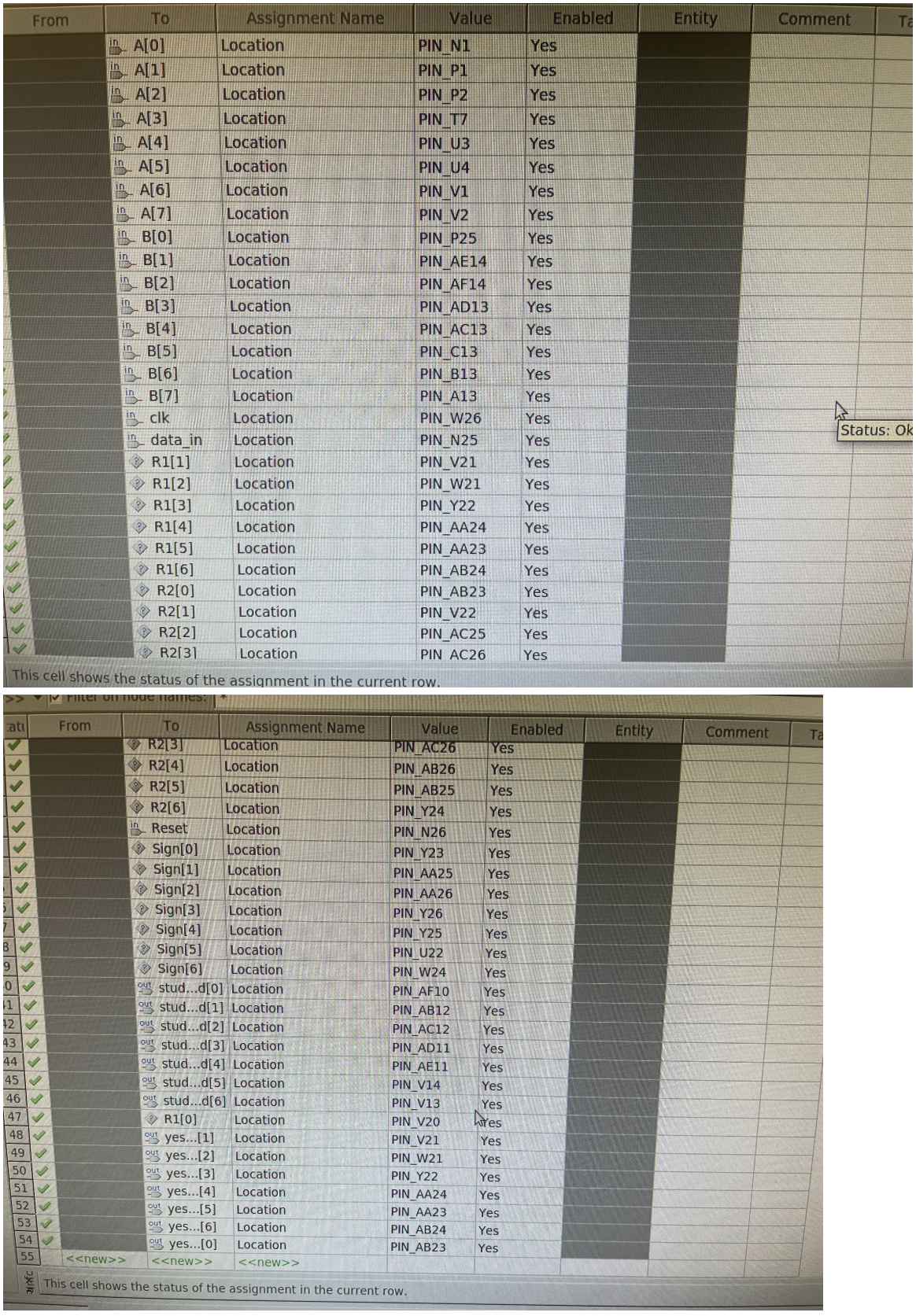


Figure 28.0: Waveform for ALU3



From	To	Assignment Name	Value	Enabled	Entity	Comment
	A[0]	Location	PIN_N1	Yes		
	A[1]	Location	PIN_P1	Yes		
	A[2]	Location	PIN_P2	Yes		
	A[3]	Location	PIN_T7	Yes		
	A[4]	Location	PIN_U3	Yes		
	A[5]	Location	PIN_U4	Yes		
	A[6]	Location	PIN_V1	Yes		
	A[7]	Location	PIN_V2	Yes		
	B[0]	Location	PIN_P25	Yes		
	B[1]	Location	PIN_AE14	Yes		
	B[2]	Location	PIN_AF14	Yes		
	B[3]	Location	PIN_AD13	Yes		
	B[4]	Location	PIN_AC13	Yes		
	B[5]	Location	PIN_C13	Yes		
	B[6]	Location	PIN_B13	Yes		
	B[7]	Location	PIN_A13	Yes		
	clk	Location	PIN_W26	Yes		
	data_in	Location	PIN_N25	Yes		
	R1[1]	Location	PIN_V21	Yes		
	R1[2]	Location	PIN_W21	Yes		
	R1[3]	Location	PIN_Y22	Yes		
	R1[4]	Location	PIN_AA24	Yes		
	R1[5]	Location	PIN_AA23	Yes		
	R1[6]	Location	PIN_AB24	Yes		
	R2[0]	Location	PIN_AB23	Yes		
	R2[1]	Location	PIN_V22	Yes		
	R2[2]	Location	PIN_AC25	Yes		
	R2[3]	Location	PIN_AC26	Yes		

This cell shows the status of the assignment in the current row.

From	To	Assignment Name	Value	Enabled	Entity	Comment
	R2[3]	Location	PIN_AC26	Yes		
	R2[4]	Location	PIN_AB26	Yes		
	R2[5]	Location	PIN_AB25	Yes		
	R2[6]	Location	PIN_Y24	Yes		
	Reset	Location	PIN_N26	Yes		
	Sign[0]	Location	PIN_Y23	Yes		
	Sign[1]	Location	PIN_AA25	Yes		
	Sign[2]	Location	PIN_AA26	Yes		
	Sign[3]	Location	PIN_Y26	Yes		
	Sign[4]	Location	PIN_Y25	Yes		
	Sign[5]	Location	PIN_U22	Yes		
	Sign[6]	Location	PIN_W24	Yes		
	stud...d[0]	Location	PIN_AF10	Yes		
	stud...d[1]	Location	PIN_AB12	Yes		
	stud...d[2]	Location	PIN_AC12	Yes		
	stud...d[3]	Location	PIN_AD11	Yes		
	stud...d[4]	Location	PIN_AE11	Yes		
	stud...d[5]	Location	PIN_V14	Yes		
	stud...d[6]	Location	PIN_V13	Yes		
	R1[0]	Location	PIN_V20	Yes		
	yes...[1]	Location	PIN_V21	Yes		
	yes...[2]	Location	PIN_W21	Yes		
	yes...[3]	Location	PIN_Y22	Yes		
	yes...[4]	Location	PIN_AA24	Yes		
	yes...[5]	Location	PIN_AA23	Yes		
	yes...[6]	Location	PIN_AB24	Yes		
	yes...[0]	Location	PIN_AB23	Yes		
<<new>>	<<new>>	<<new>>				

This cell shows the status of the assignment in the current row.

Figure 29.0: Pin Assignments for entire lab

- CONCLUSION -

Lab 6's primary goal was to build three cases of an arithmetic logical unit (ALU). With the usage of VHDL code and knowledge of sequential circuits, the entire project was completed. Every component functions to create the processor. D flip-flops are used as latches to serve as storage elements for the 8-bit inputs of A and B. The Moore logic produced the Finite State Machine (FSM) output which had its states transformed into the ALU microcodes by the 4:16 decoder. These two parts are the control units that obtain instructions and send it into the ALU to perform the tasks required. The decoder as well as the seven segment display are both combinational circuits, hence, they do not need these inputs. The FSM, ALU, and latches however, are sequential circuits, thus they all have a clock input that alternates between one and 0. Every component served its purpose to create the three different ALU's, proving that a completely functional general-purpose processor is produced once all these parts are put together.