Table of Contents:

Introduction:

The lab “Design of a Simple General Purpose Processor” covers all the concepts and ideas studied contained in this course, ranging from Seven segment display, Decoder, and Finite State machine (FSM) which were all taught and tested in previous labs. The main purpose of this lab is to design and implement an Arithmetic Logic Unit (ALU) that has the ability to perform a multitude of operations, such as logical and arithmetic operations.Which should be able to be displayed on the FPGA board. The other components that are designed and implemented to generate the inputs and produce the outputs for the ALU. The components are the FSM and Decoder which are the control unit, the Latch which is the storage unit, and the Seven segment display which is on the board and is the display.

Components:

Latch 1:

In this experiment, the latches were employed as temporary storage units. This implies latch 1 would temporarily hold input A, which was the final three digits of my student ID. In this instance,$A(72)_{16} = (0111\ 0010)_2$It accepts an 8-bit binary number as input and reads the values on the rising edge of the clock as output on the next rising edge of the clock signal. These outputs are then sent into the ALU component as inputs.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY latch1 IS
   PORT (A : IN STD_LOGIC_VECTOR(7 DOWNTO 0 );
         resetn, clock: IN STD_LOGIC;
         Q1: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END latch1;

ARCHITECTURE Behavior OF latch1 IS
BEGIN
   Process(resetn, Clock)
   BEGIN
      IF resetn = '0' THEN
         Q1 <= "00000000";
      ELSIF CLOCK'EVENT AND Clock = '1' THEN
         Q1 <= A;
      END IF;
   END PROCESS;
   END Behavior;


```

Figure 1: The VHDL Code for Latch 1

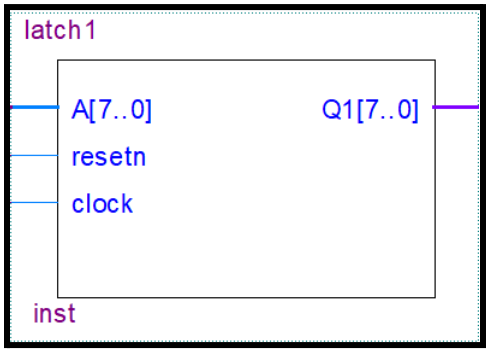


Figure 2: Latch 1 BDF

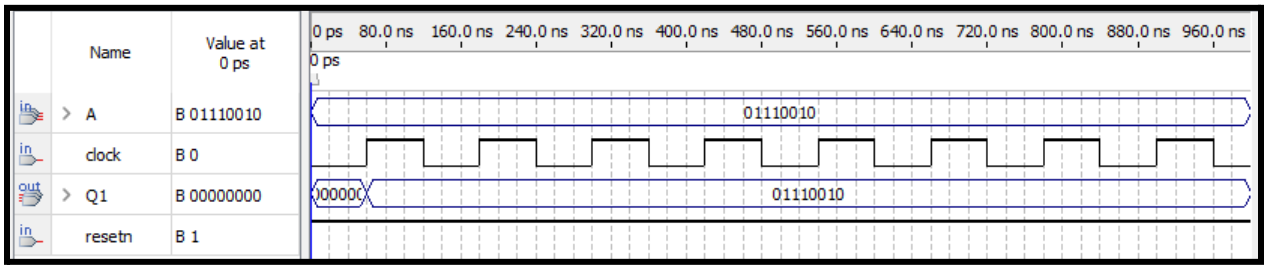


Figure 3: Waveform for Latch(Input A)

Latch 2:

Latch 2 is designed in the same manner as Latch 1, except it will be used to temporarily store input B. This input will be the binary translation of the final two of the student number's last four digits (i.e. $B = (12)_{16} = (0001\ 0010)_2$). The resulting output will once again be transported into the ALU component.

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY latch2 IS
5      PORT (B : IN STD_LOGIC_VECTOR(7 DOWNTO 0 );
6            resetn, clock: IN STD_LOGIC;
7            Q2: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
8   END latch2;
9
10  ARCHITECTURE Behavior OF latch2 IS
11  BEGIN
12     Process(resetn, Clock)
13     BEGIN
14        IF resetn = '0' THEN
15           Q2 <= "00000000";
16        ELSIF CLOCK'EVENT AND Clock = '1' THEN
17           Q2 <= B;
18        END IF;
19     END PROCESS;
20     END Behavior;
```

Figure 4: VHDL Code for Latch 2

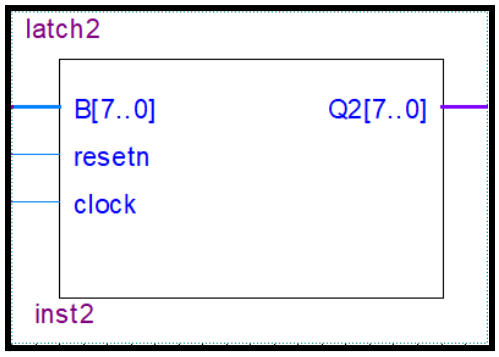


Figure 5: Latch 2 BDF

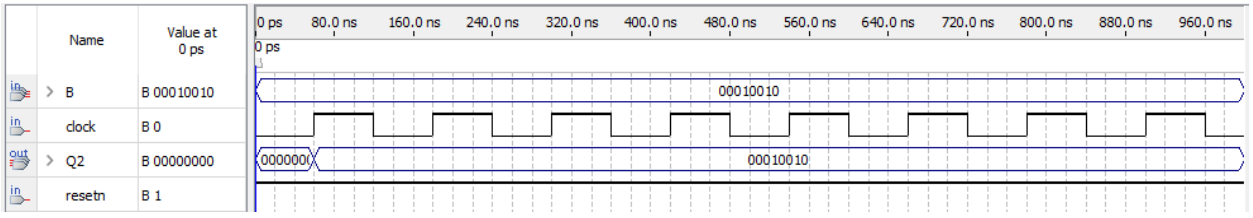


Figure 6: Waveform for Latch(Input B)

4:16 Decoder:

One of the two components that comprise the control unit is the 4:16 Decoder. Its goal is to take a 4-bit input and process it to get a 16-bit result. For the purposes of this lab, the 4:16 Decoder will accept a 4-bit value from the FSM and produce microcode, which will then be sent to the ALU. This microcode is critical in selecting which operation the ALU performs.

w3	w2	w1	w0	y15	y14	y13	y12	y11	y10	y9	y8	y7	y6	y5	y4	y3	y2	y1	y0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1: Truth Table for 4:16 Decoder

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3   ENTITY decod IS
4   PORT ( w : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
5   En : IN STD_LOGIC ;
6   y : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) ) ;
7
8   END decod ;
9   ARCHITECTURE Behavior OF decod IS
10  SIGNAL Enw : STD_LOGIC_VECTOR(4 DOWNTO 0) ;
11  BEGIN
12     Enw <= En & w ;
13  WITH w SELECT
14  y <=  "1000000000000000" WHEN NOT "0000",
15        "0100000000000000" WHEN NOT "0001",
16        "0010000000000000" WHEN NOT "0010",
17        "0001000000000000" WHEN NOT "0011",
18        "0000100000000000" WHEN NOT "0100",
19        "0000010000000000" WHEN NOT "0101",
20        "0000001000000000" WHEN NOT "0110",
21        "0000000100000000" WHEN NOT "0111",
22        "0000000010000000" WHEN NOT "1000",
23        "0000000001000000" WHEN NOT "1001",
24        "0000000000100000" WHEN NOT "1010",
25        "0000000000010000" WHEN NOT "1011",
26        "0000000000001000" WHEN NOT "1100",
27        "0000000000000100" WHEN NOT "1101",
28        "0000000000000010" WHEN NOT "1110",
29        "0000000000000001" WHEN NOT "1111",
30        "0000000000000000" WHEN OTHERS;
31
32  END Behavior;
```

Figure 7: VHDL Code for 4:16 Decoder

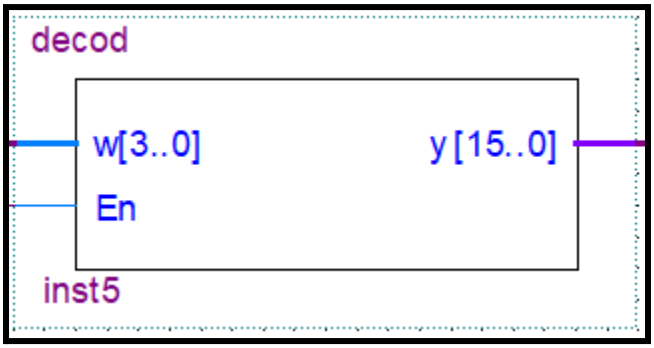


Figure 8: 4:16 Decoder BDF

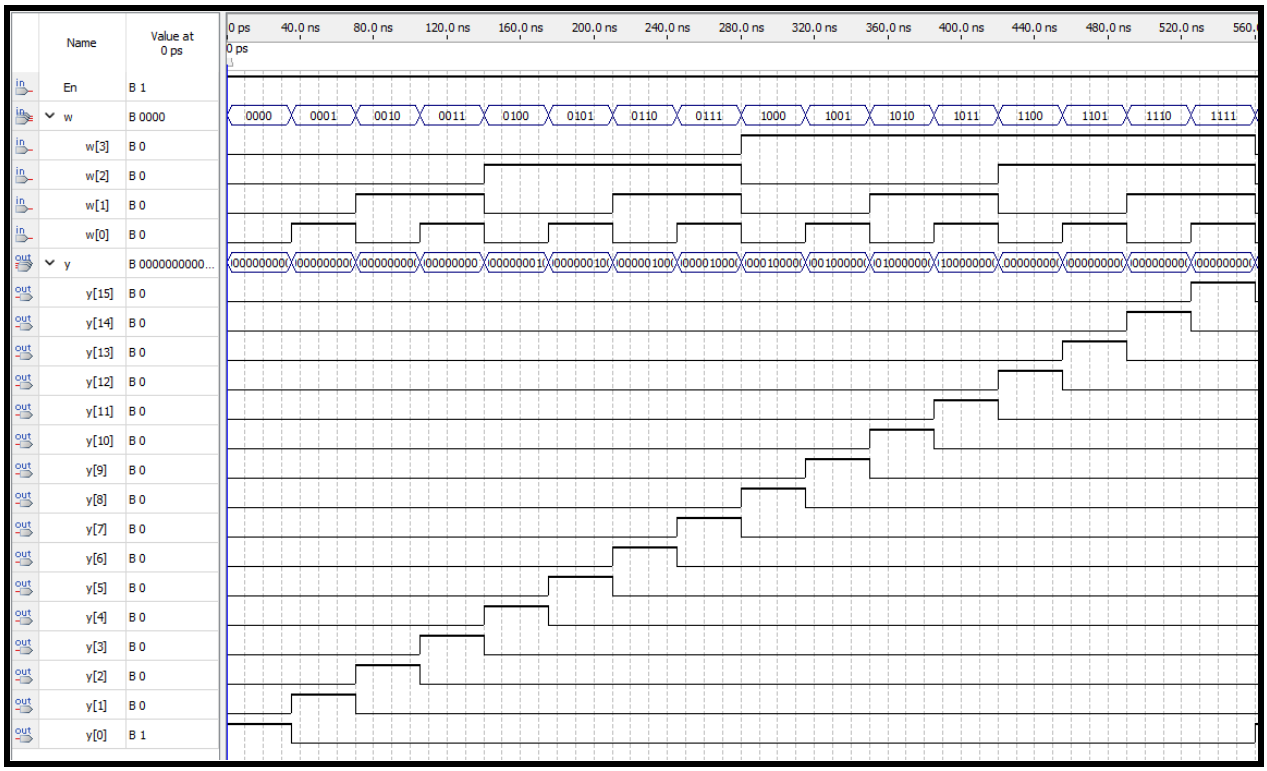


Figure 9: Waveform for 4:16 Decoder

Finite State Machine(FSM):

The finite state machine runs through 9 states in a row, beginning with 0 and ending with 8. The code includes the student number, and the only inputs are data in, clock, and reset. For the FSM to work properly, reset is set to 0 and data in is set to 1. Each digit of the student number corresponds to a different state, which changes during the clock cycle. The current state is a 4-bit number that is sent to the 4:16 decoder.

		Current State				Next State			
Student ID	State	Q_3	Q_2	Q_1	Q_0	Q_3	Q_3	Q_3	Q_3
5	0	0	0	0	0	0	0	0	1
0	1	0	0	0	1	0	0	1	0
1	2	0	0	1	0	0	0	1	1

1	3	0	0	1	1	0	1	0	0
6	4	0	1	0	0	0	1	0	1
7	5	0	1	0	1	0	1	1	0
2	6	0	1	1	0	0	1	1	1
1	7	0	1	1	1	1	0	0	0
2	8	1	0	0	0	0	0	0	0

Table 2: Truth Table for FSM

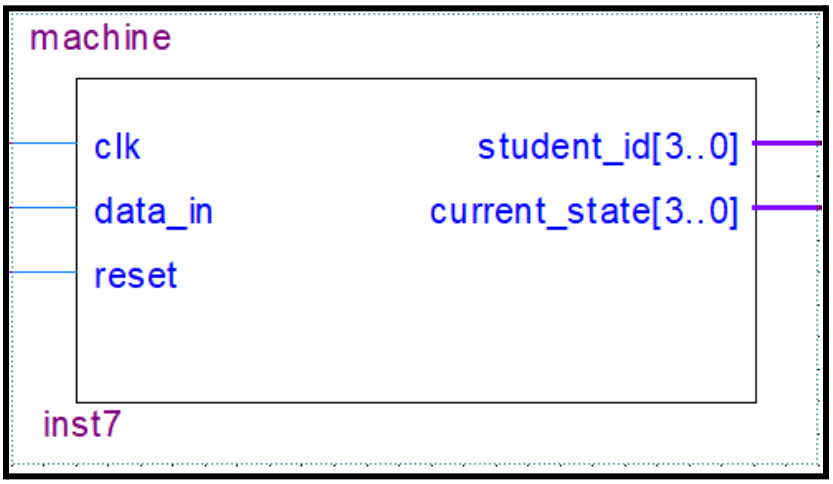


Figure 10: Block Diagram for FSM

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity machine is
    port
    (
        clk : in std_logic;
        data_in : in std_logic;
        reset : in std_logic;
        student_id : out std_logic_vector(3 downto 0);
        current_state: out std_logic_vector (3 downto 0)
    );
end entity;

architecture fsm of machine is

    type state_type is (s0, s1, s2, s3, s4, s5, s6, s7, s8);

    signal yfsm : state_type;

begin
    process(clk, reset)
    begin
        if reset = '1' then
            yfsm <= s0;
        elsif (clk'EVENT AND clk = '1') then

            case yfsm is
                when s0 =>
                    IF (data_in = '0') then
                        yfsm <= s0;
                    ELSE
                        yfsm <= s1;
                    END IF;
                when s1 =>
                    IF (data_in = '0') then
                        yfsm <= s1;
                    ELSE
                        yfsm <= s2;
                    END IF;
                when s2 =>
                    IF (data_in = '0') then
                        yfsm <= s2;
                    ELSE
                        yfsm <= s3;
                    END IF;
                when s3 =>
```

Figure 11: VHDL Code for FSM (Part 1)

```
46              when s3 =>
47                 IF (data_in = '0') then
48                    yfsm <= s3;
49                 ELSE
50                    yfsm <= s4;
51                 END IF;
52              when s4 =>
53                 IF (data_in = '0') then
54                    yfsm <= s4;
55                 ELSE
56                    yfsm <= s5;
57                 END IF;
58              when s5 =>
59                 IF (data_in = '0') then
60                    yfsm <= s5;
61                 ELSE
62                    yfsm <= s6;
63                 END IF;
64              when s6 =>
65                 IF (data_in = '0') then
66                    yfsm <= s6;
67                 ELSE
68                    yfsm <= s7;
69                 END IF;
70              when s7 =>
71                 IF (data_in = '0') then
72                    yfsm <= s7;
73                 ELSE
74                    yfsm <= s8;
75                 END IF;
76              when s8 =>
77                 IF (data_in = '0') then
78                    yfsm <= s8;
79                 ELSE
80                    yfsm <= s0;
81                 END IF;
82              END CASE;
83           END IF;
84        END process;
85
86     process (yfsm, data_in)
87     begin
88        case yfsm is
89           when s0 =>
90              student_id <= "0101";--s#:5
91              current_state <= "0000";
```

Figure 12: VHDL Code for FSM (Part 2)

```
92              when s1 =>
93                 student_id <= "0000";--s#:0
94                 current_state <= "0001";
95              when s2 =>
96                 student_id <= "0001";--s#:1
97                 current_state <= "0010";
98              when s3 =>
99                 student_id <= "0001";--s#:1
100                current_state <= "0011";
101             when s4 =>
102                student_id <= "0110";--s#:6
103                current_state <= "0100";
104             when s5 =>
105                student_id <= "0111";--s#:7
106                current_state <= "0101";
107             when s6 =>
108                student_id <= "0010";--s#:2
109                current_state <= "0110";
110             when s7 =>
111                student_id <= "0001";--s#:1
112                current_state <= "0111";
113             when s8 =>
114                student_id <= "0010";--s#:2
115                current_state <= "1000";
116       END CASE;
117    END process;
118 END fsm;
```

Figure 13: VHDL Code for FSM (Part 3)

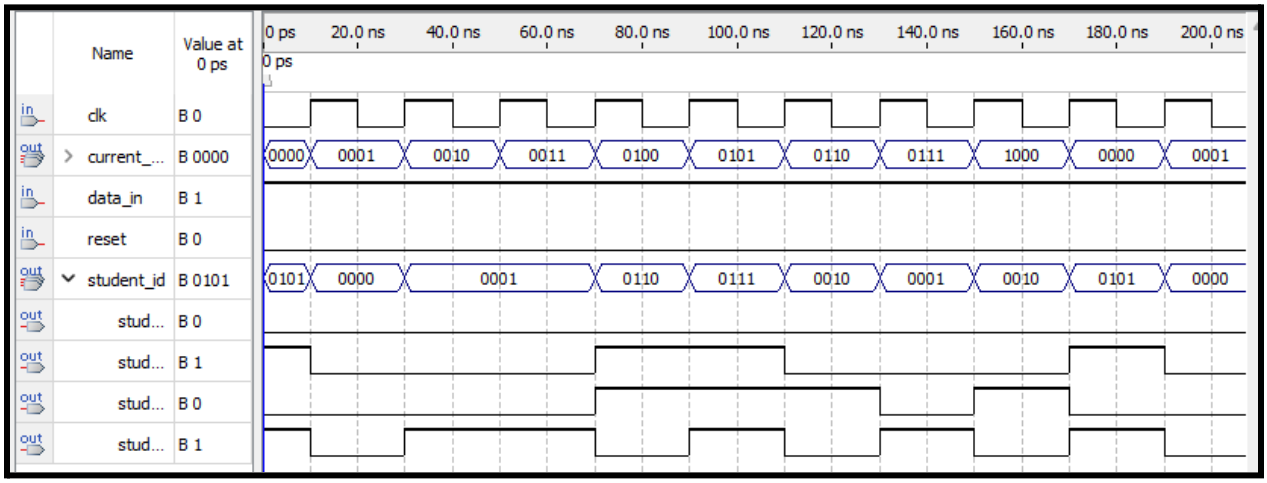


Figure 14: Waveform for FSM

Problem 1;ALU 1:

The ALU, which is the primary component, is used to carry out the operations. There are five inputs: the Student ID number, OP, the A and B values, and the clock. The FSM output provides the student ID, the decoder output provides the OP, and the latches provide the A and B values. The clock is used to transition between functions. The OP has a 16-bit value that comes from the decoder and is used as microcode to tell the ALU what state it is in. Each state, represented by the 16-bit microcode, has a specific purpose.

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	000000010000000	$\overline{A \oplus B}$

Table 3: ALU Operations for Problem Set 1

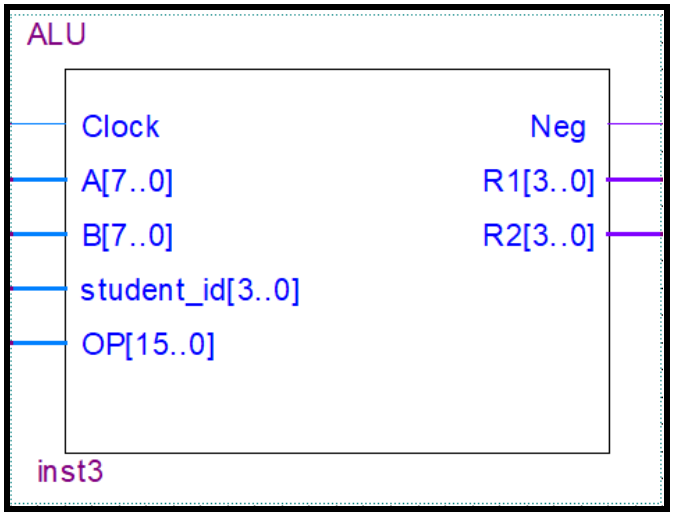


Figure 15: The Block Diagram for ALU 1

```
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(Clock     : in std_logic; --input clock signal
8         A,B         : in unsigned (7 downto 0); -- 8 bit inputs from latches A and B
9         student_id  : in unsigned(3 downto 0); --4 bit student id from FSM
10        OP          : in unsigned (15 downto 0); --16 bit selector for Operation from Decoder
11        Neg         : out std_logic;
12        R1          : out unsigned (3 downto 0);
13        R2          : out unsigned (3 downto 0));
14     end ALU;
15
16     architecture calculation of ALU is
17     signal Reg1,Reg2,Result : unsigned(7 downto 0) := (others => '0');
18     signal Reg4 : unsigned(0 to 7);
19     begin
20     Reg1 <= A;--temp variables to do your operations/signal
21     Reg2 <= B;
22     process(Clock, OP)
23     Begin
24        if(rising_edge(Clock)) THEN
25        neg <= '0';
26        case OP is
27           When "0000000000000001" =>
28           -- Do Addition for Reg1 and Reg2
29           result <= A + B;
30
31           When "0000000000000010" =>
32           -- Do subtraction
33           -- Neg bit set if required
34           if (A < B) then
35           neg <= '1';
36           result <= B - A;
37           else
38           result <= A - B;
39           end if;
40
41           When "0000000000000100" =>
42           -- do inverse
43           result <= NOT A;
44
45           When "0000000000001000" =>
46           -- do boolean nand
```

Figure16 : ALU code for Problem(Part 1)

```
47              result <= NOT (A and B);
48
49              When "0000000000010000" =>
50              -- do boolean Nor
51              result <= NOT (A OR B);
52
53              When "0000000000100000" =>
54              -- do boolean AND
55              result <= A AND B;
56
57              When "0000000001000000" =>
58              -- do boolean xor
59              result <= (A AND (NOT B)) OR ((NOT A) AND B);
60
61              When "0000000010000000" =>
62              -- do boolean or
63              result <= A OR B;
64
65              When "0000000100000000" =>
66              -- do boolean Xnor
67              result <= NOT((A AND (NOT B)) OR ((NOT A) AND B));
68
69              When others =>
70              -- Dont care, do nothing
71
72          end case;
73      end if;
74  end process;
75  R1 <= Result(3 downto 0);
76  R2 <= Result(7 downto 4);
77  end calculation;
78
79
80
```

Figure 17 : ALU code for Problem(Part 2)

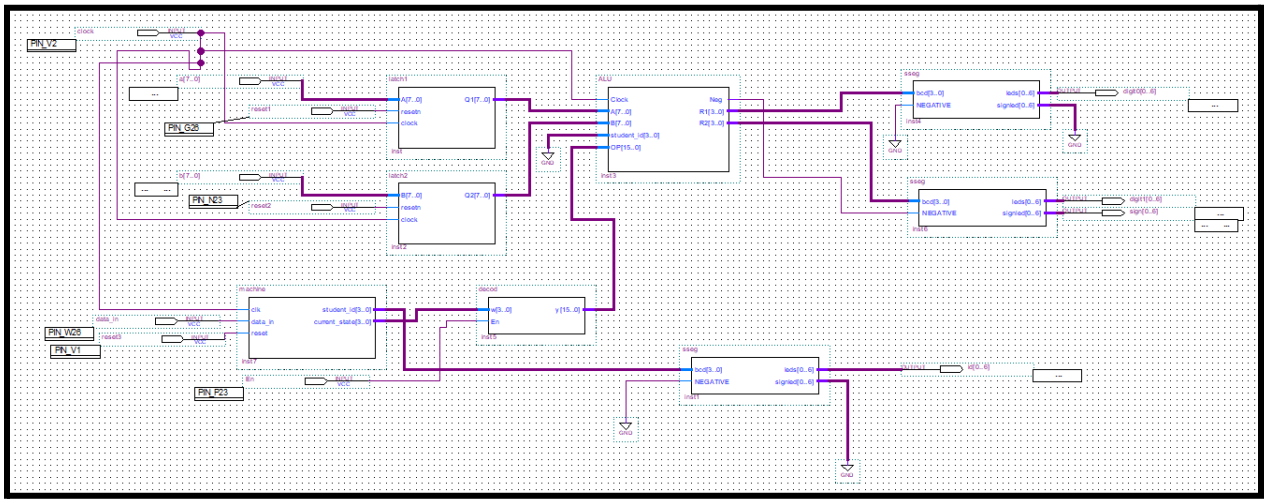


Figure 18: The Block Diagram for Problem 1

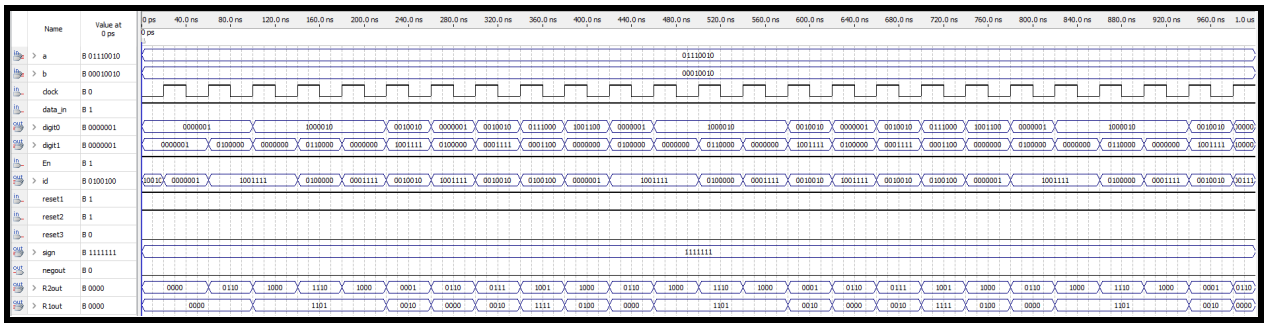


Figure 19: Waveform for Problem 1

According to Figure 19, the first output is "0000," which is not a result of the predicted microcode based on the procedures. This is because there is a delay in the result between the decoder and the ALU. The ALU performs operations on the rising edge of the clock, but the decoder does not. This happens because the ALU performs the operation depending on the FSM's previous state. This is the reason for the "0000" output at the start, as the FSM has no states prior to 0, resulting in the output. This latency is present in ALU 2 and ALU 3.

Hand Computations for Problem 1:

Problem #1:

A: $(01110010)_2 = (72)_{16}$

B: $(00010010)_2 = (12)_{16}$

5
0
1
1
6
7 [- 0111] A
2 [- 0010]
1 [- 0001] B
2 [- 0010]

Function #1: Sum(A,B)

$$\begin{array}{r} 01110010 \\ +\ 00010010 \\ \hline 10000100 \end{array} \Rightarrow (84)_{16}$$

Function #2: diff(A,B)

$$\begin{array}{r} 01110010 \\ -\ 00010010 \\ \hline 01100000 \end{array} \Rightarrow (60)_{16}$$

Function #3: $\bar{A}$

$01110010 \Rightarrow 10001101 \Rightarrow (8D)_{16}$ (not)

Function #4: $\overline{A \cdot B}$

$$\begin{array}{r} 01110010 \\ 00010010 \\ \hline 11101101 \end{array} = (ED)_{16}$$

Function #5: $\overline{A+B}$

$$\begin{array}{r} 01110010 \\ 00010010 \\ \hline 10001101 \end{array} \Rightarrow (8D)_{16}$$

Function #6: $A \cdot B$

$$\begin{array}{r} 01110010 \\ 00010010 \\ \hline 00010010 \end{array} \Rightarrow (12)_{16}$$

Function #7: $A \oplus B$

$$\begin{array}{r} 01110010 \\ 00010010 \\ \hline 01100000 \end{array} = (60)_{16}$$

Function #8: $A + B$ (or-add)

$$\begin{array}{r} 01110010 \\ 00010010 \\ \hline 01110010 \end{array} = (72)_{16}$$

Function #9: $\overline{A \oplus B}$

$$\begin{array}{r} 01110010 \\ 00010010 \\ \hline 10011111 \end{array} = (9F)_{16}$$

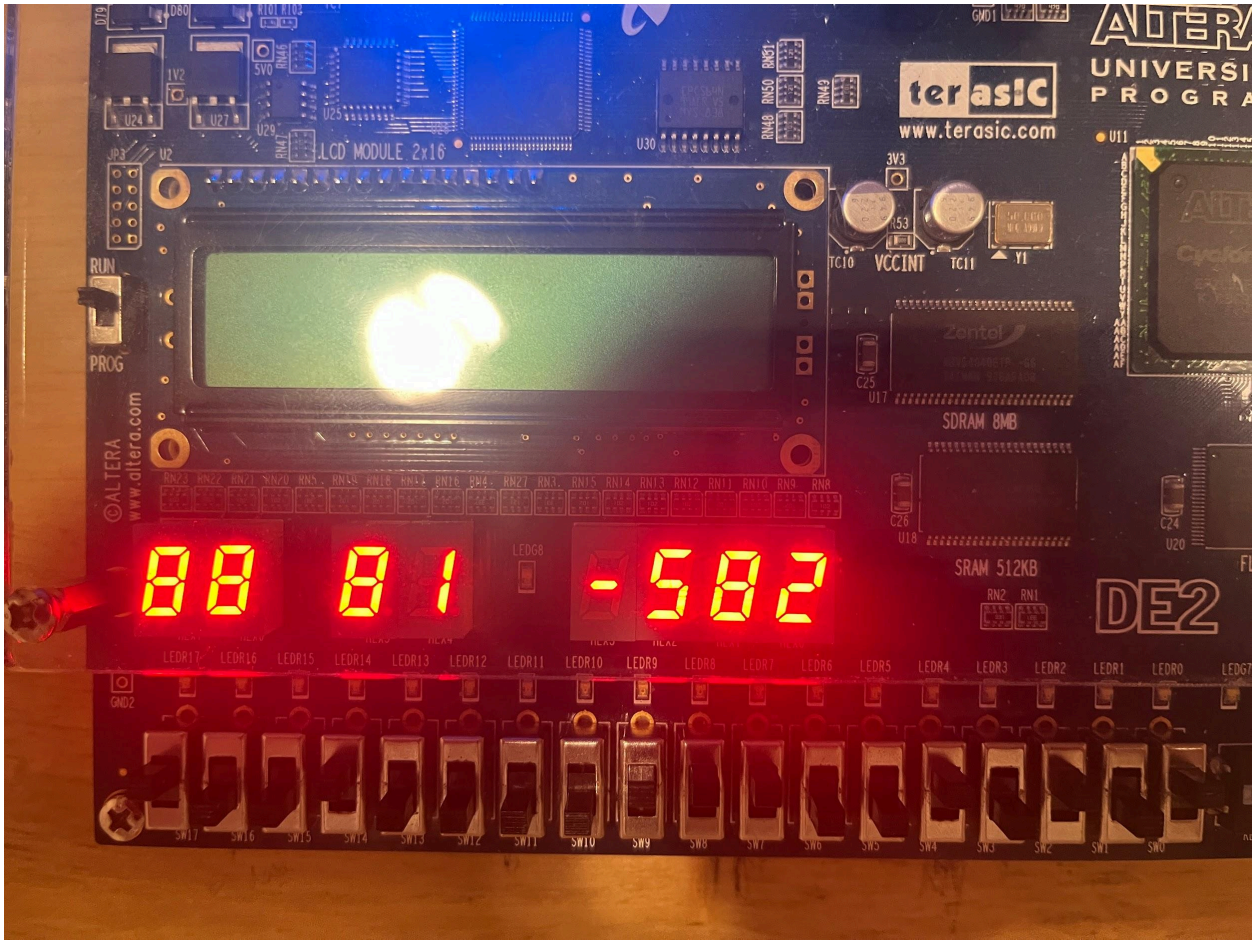


Image 1:Problem 1 outputs for FPGA board

Problem 2;ALU 2; Selection B):

This ALU functioned similarly to ALU 1. It worked precisely the same manner, with the identical variables and code structure. The only change was that new and different logic functions had to be performed. These functions were described in the lab handbook, and problem set B was assigned to this experiment.

Function #	Microcode	Operation / Function
1	0000000000000001	Swap the lower and upper 4 bits of **A**
2	0000000000000010	Produce the result of ORing **A** and **B**
3	0000000000000100	Decrement **B** by 5
4	0000000000001000	Invert all bits of **A**
5	0000000000010000	Invert the bit-significance of **A**
6	0000000000100000	Find the greater value of **A** and **B** and produce the results

		(Max(**A**, **B**))
7	0000000001000000	Produce the difference between **A** and **B**
8	0000000010000000	Produce the result of XNORing **A** and **B**
9	0000000010000000	Rotate **B** to the left by three bits (ROL)

Table 4: ALU Operations for Problem Set 2

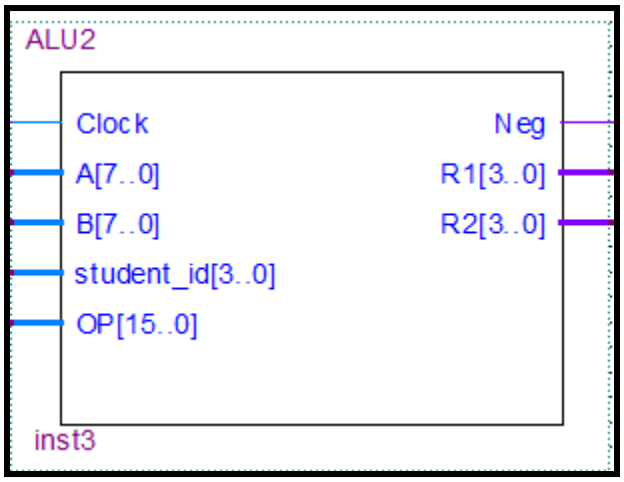


Figure 20: Block Diagram for ALU 2

```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; --input clock signal
      A,B         : in unsigned (7 downto 0); -- 8 bit inputs from latches A and B
      student_id  : in unsigned(3 downto 0); --4 bit student id from FSM
      OP          : in unsigned (15 downto 0); --16 bit selector for Operation from Decoder
      Neg         : out std_logic;
      R1          : out unsigned (3 downto 0);
      R2          : out unsigned (3 downto 0));
   end ALU2;

   architecture calculation 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)
   Begin
      if(rising_edge(Clock)) THEN
      neg <= '0';
      case OP is
         When "0000000000000001" =>
         -- Swap the lower and upper 4 bits of A
         Result(7 downto 4) <= A(3 downto 0);
         Result(3 downto 0) <= A(7 downto 4);

         When "0000000000000010" =>
         -- Produce the result of ORing A and B
         Result <= A OR B;

         When "0000000000000100" =>
         -- Decrement B by 5
         Result <= B - 5;


         When "0000000000001000" =>
         -- Invert all bits of A
         Result <= NOT A;

         When "0000000000010000" =>
         -- Invert the bit-significance order of A
```

Figure 21: ALU Code for Problem 2(Part 1)

```
47          for i in 0 to 7 loop
48             Result(i) <= Reg1(7-i);
49          end loop;
50          Neg <= '0';
51
52          When "0000000000100000" =>
53          -- Find the greater value of A and B and produce the results
54          if(A > B) THEN
55          Result <= A;
56          else
57          Result <= B;
58          end if;
59
60          When "0000000001000000" =>
61          -- Produce the difference between A and B
62          if (A < B) then
63          Result <= B - A;
64          else
65          Result <= A - B;
66          end if;
67
68          When "0000000010000000" =>
69          -- Produce the result of XNORING A and B
70          Result <= NOT((A AND (NOT B)) OR ((NOT A) AND B));
71
72          When "0000000100000000" =>
73          -- Rotate B to the left by three bits(ROL)
74          Result <= (B rol 3);
75
76          When others =>
77          -- Dont care, do nothing
78
79          end case;
80       end if;
81    end process;
82  R1 <= Result(3 downto 0);
83  R2 <= Result(7 downto 4);
84  end calculation;
```

Figure 22:ALU Code for Problem 2 (Part 2)

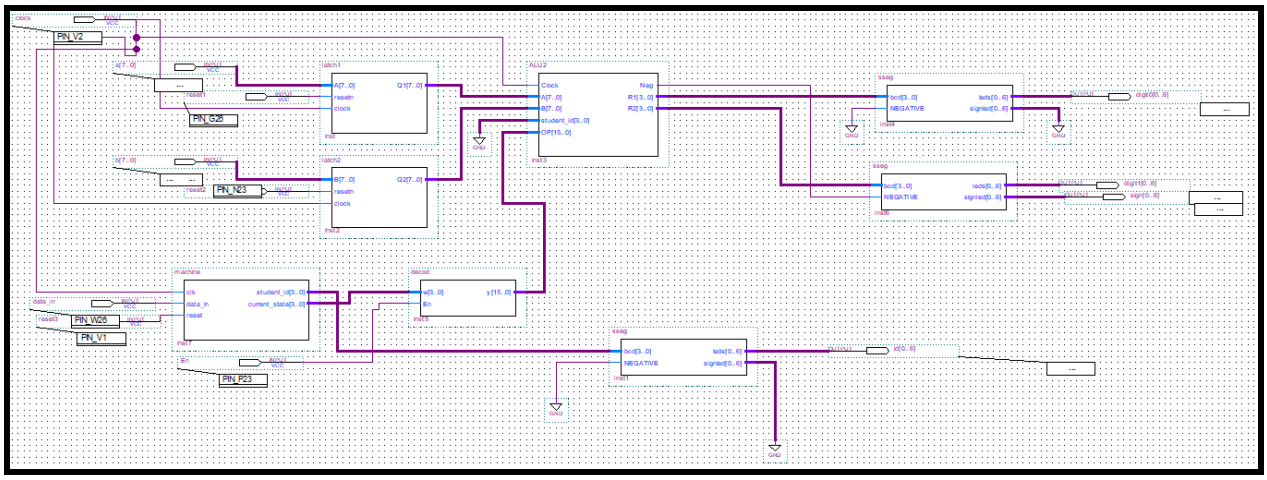


Figure 23: Block Diagram for Problem 2

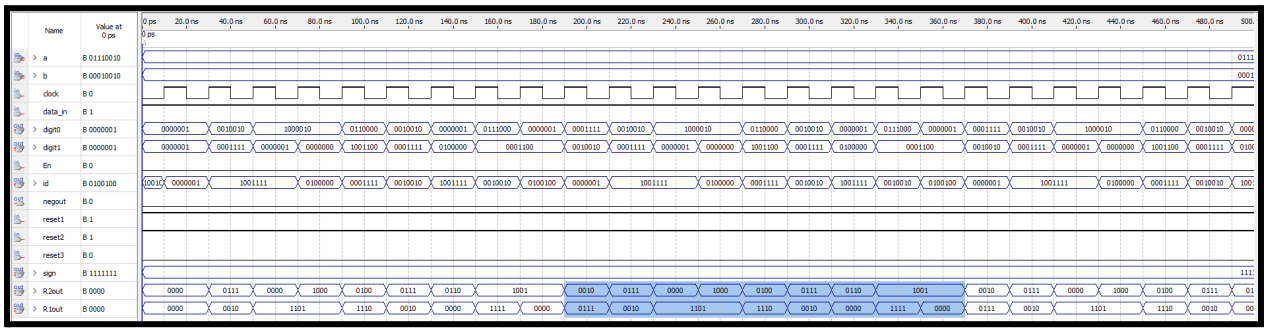


Figure 24: Waveform for Problem 2

According to Figure 19, the first output is "0000," which is not a result of the predicted microcode based on the procedures. This is because there is a delay in the result between the decoder and the ALU. The ALU performs operations on the rising edge of the clock, but the decoder does not. This happens because the ALU performs the operation depending on the FSM's previous state. This is the reason for the "0000" output at the start, as the FSM has no states prior to 0, resulting in the output. As explained in Problem 1, the student number and the operations are not synchronized.

Hand Computations for Problem 2:

7 6 5 4 3 2 1 0
Lower ← b

Problem #2 A: $(01111010)_2 = (72)_{16}$
B: $(00010010)_2 = (12)_{16}$

Function #1: Swap the Lower and upper 4 bits of A

A = 01111010 → $(10100111)_2 = (27)_{16}$

Function #2: Produce the result of oR(all)ing A and B

01111010
00010010
$(01111010)_2 \Rightarrow (72)_{16}$

Function #3: Decrement B by 5

00010010
- 00000101
00001101 $\Rightarrow (0D)_{16}$

Function #4: Invert all bits of A

$\overline{A}$ = 01111010 $\Rightarrow$ 10000101 $\Rightarrow (85)_{16}$
(not)

Function #5: Invert all bit-significance of A

A = 01111010 $\Rightarrow (10000110)_2 = (86)_{16}$

Function #6: Find the greater value of A and B and produce the result (Max(A,B))

A = 01111010 > B = 00010010

Max(A,B) = $(01111010)_2 = (72)_{16}$

Function #7: Produce the difference between A and B

01111010
- 00010010
01100000 $\Rightarrow (60)_{16}$

Function #8: Produce the result of XNoRing A and B

01111010
00010010
10001111 $\Rightarrow (8F)_{16}$

Function #9: Rotate B to left by three bits (ROL)

B = 00010010 $\Rightarrow (10010000)_2 \Rightarrow (90)_{16}$

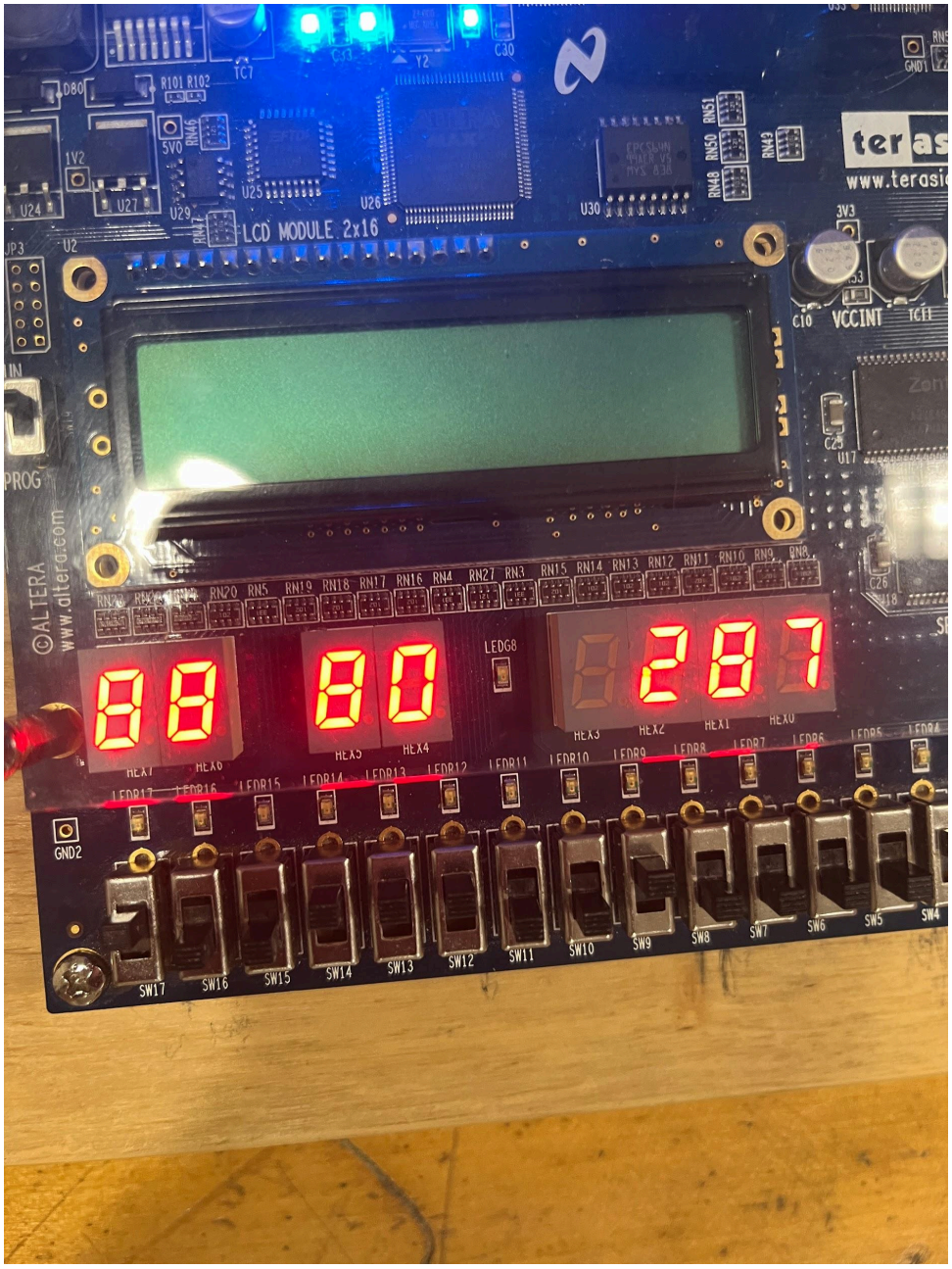


Image 2: Problem 2 outputs for FPGA board

Problem 3 ;ALU 3; Selection B):

This ALU worked in a different way than the preceding ones. This is because it is much simpler to operate because it has fewer functions to perform. For this section, problem set B was assigned once again. ALU 3 had to determine whether the digits of the student number were even or odd. The code will set the output variable "result" to "0000" if it is even, and "0001" if it is odd. This output was then fed into a modified SSEG. This new SSEG will output a "y" on the FPGA board if the input value is "0000" and a "n" if the input value is "0001".

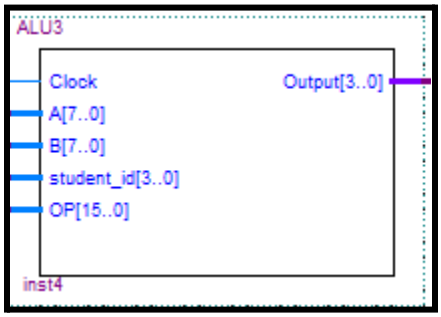
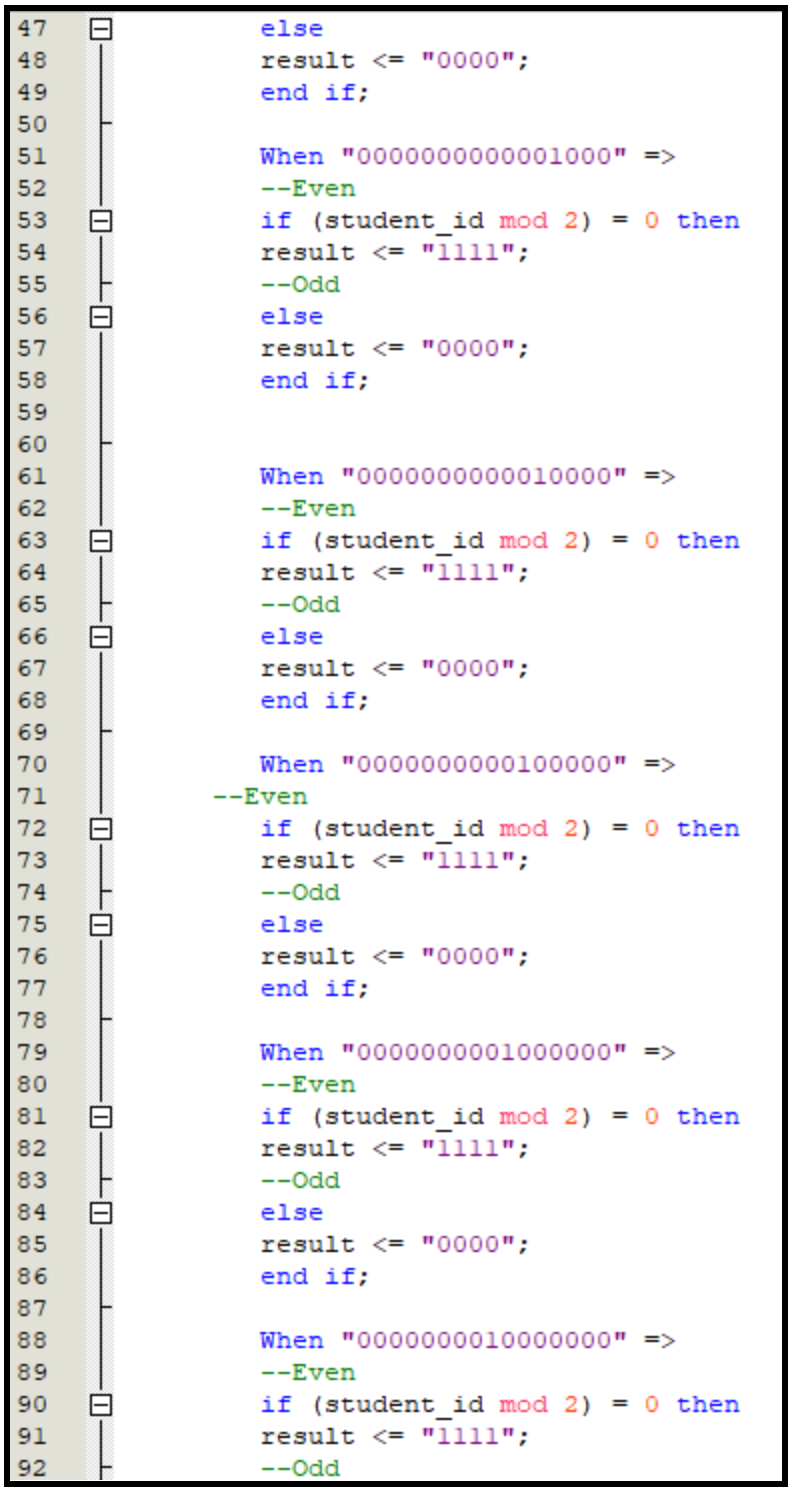


Figure 25: Symbol Block for ALU 3

```
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 ALU3 is
7      port(Clock     : in std_logic; --input clock signal
8         A,B         : in unsigned (7 downto 0); -- 8 bit inputs from latches A and B
9         student_id  : in unsigned(3 downto 0); --4 bit student id from FSM
10        OP          : in unsigned (15 downto 0); --16 bit selector for Operation from Decoder
11        Output      : out unsigned(3 downto 0));
12     end ALU3;
13
14     architecture calculation of ALU3 is
15     signal Result : unsigned(3 downto 0) := (others => '0');
16     signal Reg4 : unsigned(0 to 7);
17     begin
18
19     process(Clock, OP)
20     Begin
21        if(rising_edge(Clock)) THEN
22        case OP is
23           When "0000000000000001" =>
24           --Even
25           if (student_id mod 2) = 0 then
26           result <= "1111";
27           --Odd
28           else
29           result <= "0000";
30           end if;
31
32           When "0000000000000010" =>
33           --Even
34           --Even
35           if (student_id mod 2) = 0 then
36           result <= "1111";
37           --Odd
38           else
39           result <= "0000";
40           end if;
41
42           When "0000000000000100" =>
43           --Even
44           if (student_id mod 2) = 0 then
45           result <= "1111";
46           --Odd
```

Figure 26:ALU Code for problem 3(part 1)



```
47          else
48          result <= "0000";
49          end if;
50
51          When "0000000000001000" =>
52          --Even
53          if (student_id mod 2) = 0 then
54          result <= "1111";
55          --Odd
56          else
57          result <= "0000";
58          end if;
59
60
61          When "0000000000010000" =>
62          --Even
63          if (student_id mod 2) = 0 then
64          result <= "1111";
65          --Odd
66          else
67          result <= "0000";
68          end if;
69
70          When "0000000000100000" =>
71        --Even
72          if (student_id mod 2) = 0 then
73          result <= "1111";
74          --Odd
75          else
76          result <= "0000";
77          end if;
78
79          When "0000000001000000" =>
80          --Even
81          if (student_id mod 2) = 0 then
82          result <= "1111";
83          --Odd
84          else
85          result <= "0000";
86          end if;
87
88          When "0000000010000000" =>
89          --Even
90          if (student_id mod 2) = 0 then
91          result <= "1111";
92          --Odd
```

Figure 27: ALU Code for problem 3(part 2)

```
 94            result <= "0000";
 95            end if;
 96
 97            When "0000000100000000" =>
 98          --Even
 99            if (student_id mod 2) = 0 then
100            result <= "1111";
101            --Odd
102            else
103            result <= "0000";
104            end if;
105
106
107            When others =>
108            -- Dont care, do nothing
109
110            end case;
111         end if;
112      end process;
113      Output <= result(3 downto 0);
114   end calculation;
115
```

Figure 28:ALU Code for problem 3(part 3)

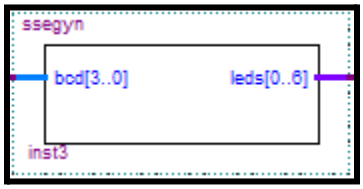


Figure 29: Symbol Block for Modified Seven Segment Display

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3   ENTITY ssegyn IS
4       PORT ( bcd : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
5           leds : OUT STD_LOGIC_VECTOR(0 TO 6) ) ;
6   END ssegyn ;
7
8   ARCHITECTURE Behavior OF ssegyn IS
9   BEGIN
10      PROCESS(bcd)
11          BEGIN
12
13
14          CASE bcd IS                 --abcdefg
15              WHEN "0000" => leds <= NOT "0010101" ; --N
16              WHEN "1111" => leds <= NOT "0111011" ; --Y
17
18              WHEN OTHERS => leds <= NOT "0000001" ;
19          END CASE ;
20      END PROCESS ;
21  END Behavior ;
```

Figure 30: Code for Modified Seven Segment Display

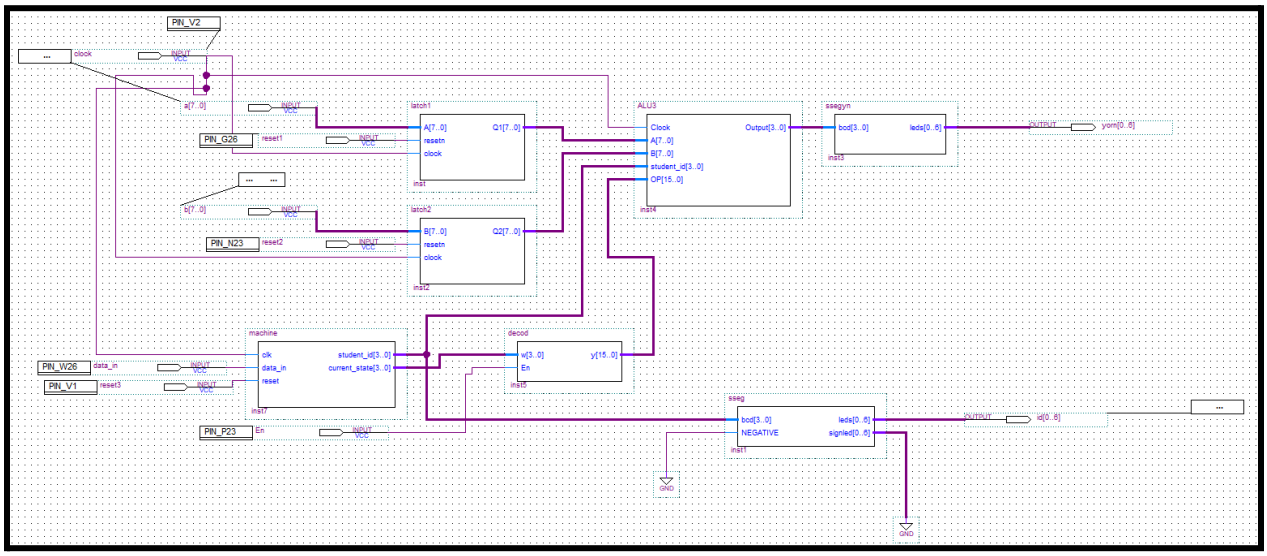


Figure 31: Block Diagram for Problem 3

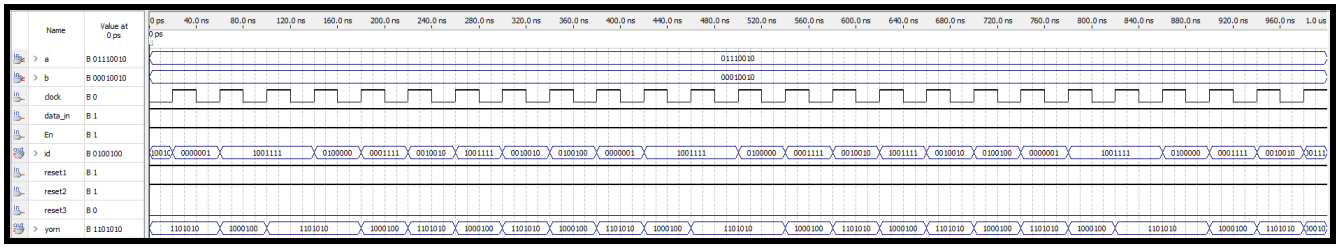


Figure 32: Waveform for Problem 3

The student number and the operations are not synced, as indicated in Problem 1. The procedure that shows 'y' for even numerals and 'n' for odd digits is not synchronized in the figures below. It shows the outcome for the preceding student digit.

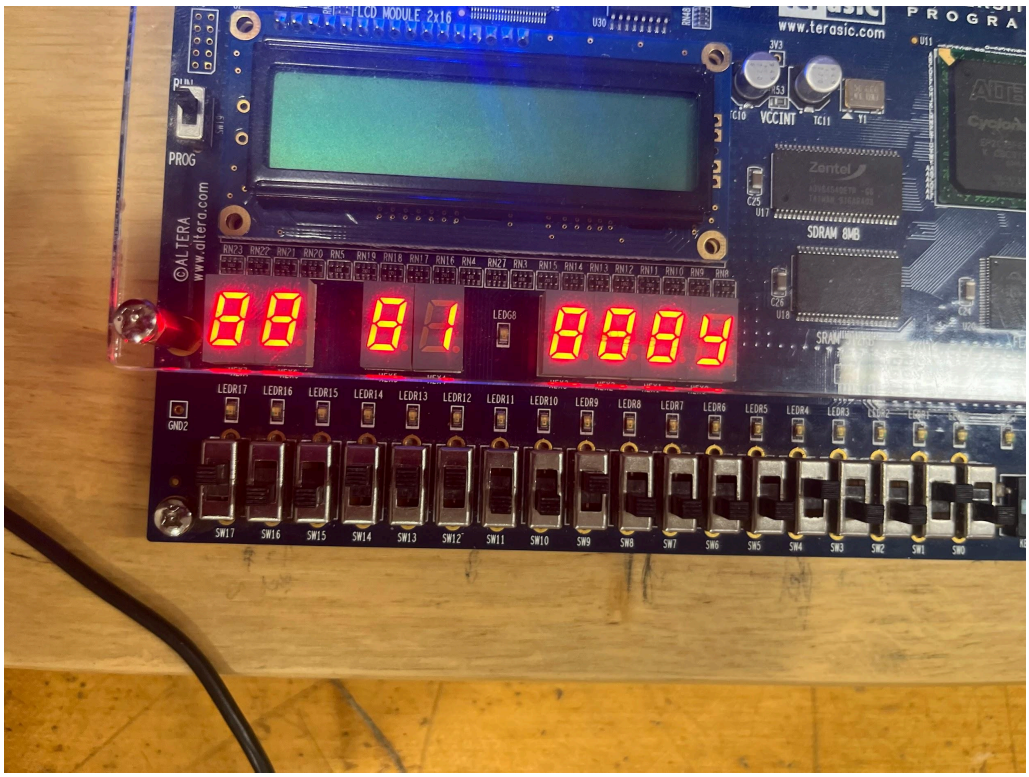


Image 2: Problem 2 outputs for FPGA board

Conclusion:

The primary goal of this lab was to develop and create an ALU component in the VHDL environment that performed varied operations based on the state, inputs, and written code. This task was accurately completed as a General-Purpose Processor (GPP)was designed, implemented, compiled, and tested. The GPP was a combination of control and storage units, Arithmetic and logic units, and the Seven segment display. The general purpose processor was built on the FPGA board once all of the code was written and compiled. Handwritten solutions were compared to device data to check that the lab was completed correctly. After assembling these components, the implementation into the FPGA board was successful.