Table of Contents

Introduction

The objective of this lab is to design a simple general-purpose processor using a variety of components. Three different Arithmetic and Logic Unit's (ALU) will be created, and they will be used alongside latches, a 4-16 decoder, a Moore State Machine (FSM), and seven-segment displays (SSEG) in order to display the output. Every component will be tested prior to the implementation and will be explained in the report.

Components

Latch

The latch was used to store values of binary numbers and display them. When the latch is on, an output will be generated however when it is turned off (Resetn = 0), a value of null will be displayed, and all memory will be erased. Due to this, there will be a lag and a delay in the waveform. This can be seen at the start of the waveforms and once a cycle is fully complete, the outputs are correctly outputted. All waveforms will behave the same.

Truth Table

Customized Latch

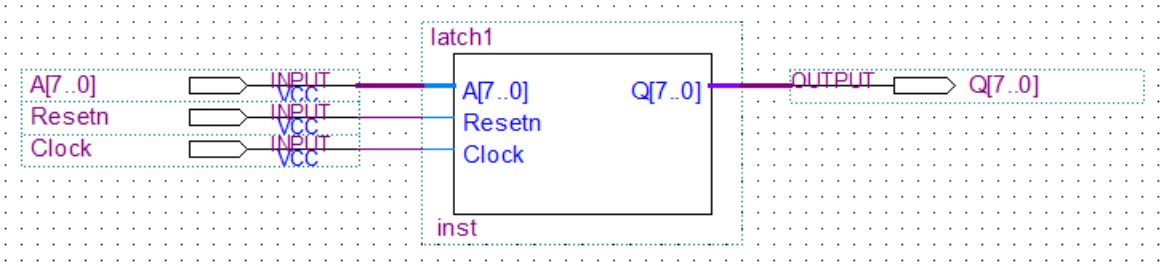
A (Input)	Reset	Q	Action
1	0	0	-
1	1	1	Latch
0	0	0	-
0	1	1	Latch

Latch when Q=A

VHDL

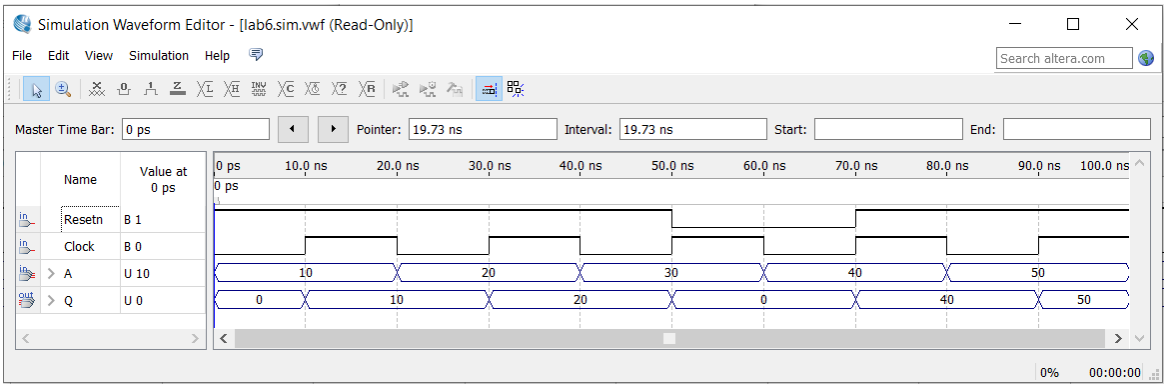
```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY latch1 IS
5      PORT ( A               : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
6             Resetn, Clock   : IN STD_LOGIC;
7             Q               : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
8   END latch1;
9
10  ARCHITECTURE Behavior OF latch1 IS
11     BEGIN
12        PROCESS(Resetn, Clock)
13        BEGIN
14           IF Resetn = '0' THEN
15              Q <= "00000000";
16           ELSIF Clock'EVENT AND Clock='1' THEN
17              Q <= A;
18           END IF;
19        END PROCESS;
20  END Behavior;
```

Schematic



Waveform

The waveform shows how the latch works when Resetn = 1. When Resetn = 0, the output is simply zero which means the latch is off.



4:16 Decoder

The 4:16 decoder used in the lab is a component of a combinational circuit made from different gates which decodes n inputs and produces 2^n outputs. An enable signal allows the decoder to turn off/on. A 3:8 decoder was used to create the 4:16 decoder and the implementation is shown through the VHDL code shown below.

Truth Table

E	w_1	w_2	w_3	w_4	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	1	0	0	0	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	0
1	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

VHDL

The code on the left is of a 3:8 decoder in which a package is created which will be used in the code on the right, which is the code of the 4:16 decoder.

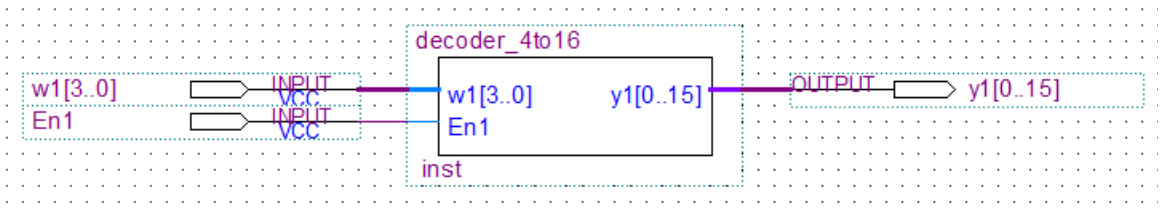
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY decoder IS
   PORT ( w  : IN    STD_LOGIC_VECTOR(2 DOWNTO 0);
          En : IN    STD_LOGIC;
          y  : OUT   STD_LOGIC_VECTOR(0 TO 7));
END decoder;

ARCHITECTURE Behavior OF decoder IS
   SIGNAL Enw : STD_LOGIC_VECTOR(3 DOWNTO 0);

BEGIN
   Enw <= En & w;
   WITH Enw SELECT
      y <= "00000001" WHEN "1000",
           "00000010" WHEN "1001",
           "00000100" WHEN "1010",
           "00001000" WHEN "1011",
           "00010000" WHEN "1100",
           "00100000" WHEN "1101",
           "01000000" WHEN "1110",
           "10000000" WHEN "1111",
           "00000000" WHEN OTHERS;
END Behavior;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
Package decoder_Package is
   Component decoder
      PORT ( w    : IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
             En   : IN  STD_LOGIC;
             y    : OUT    STD_LOGIC_VECTOR(0 TO 7));
   END Component;
end decoder_Package;
```

3:8 Decoder

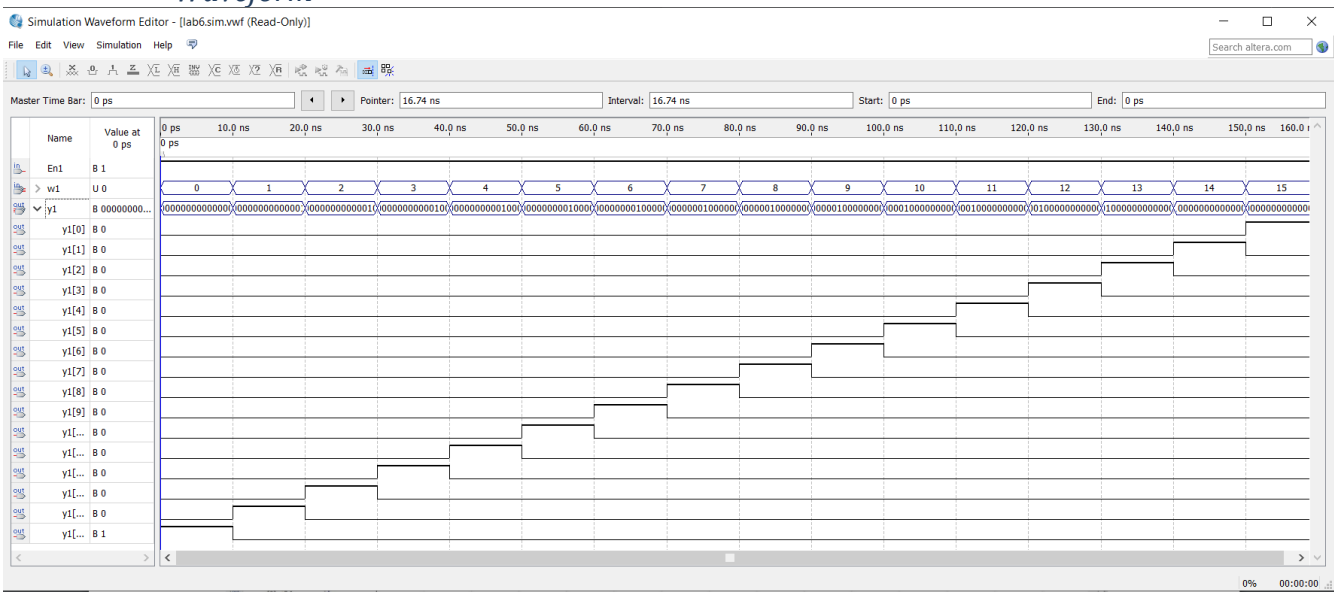
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.decoder_Package.all;

ENTITY decoder_4to16 IS
   PORT ( w1  : IN   STD_LOGIC_VECTOR(3 DOWNTO 0);
          En1 : IN   STD_LOGIC;
          y1  : OUT  STD_LOGIC_VECTOR(0 TO 15));
END decoder_4to16;

ARCHITECTURE Behavior OF decoder_4to16 IS
   SIGNAL w_s  : STD_LOGIC_VECTOR(2 DOWNTO 0);
   SIGNAL y_r  : STD_LOGIC_VECTOR(0 to 7);
   SIGNAL w1_3 : STD_LOGIC;

BEGIN
   w1_3 <= w1(3);
   w_s(2 downto 0) <= w1(2 downto 0);
   stage : decoder PORT MAP (w_s, En1, y_r);

   PROCESS (w1_3)
      BEGIN
         IF (w1_3 = '0') THEN
            y1(8 to 15) <= y_r;
            y1(0 to 3)  <= "0000";
            y1(4 to 7)  <= "0000";
         ELSE
            y1(0 to 7)   <= y_r;
            y1(8 to 11)  <= "0000";
            y1(12 to 15) <= "0000";
         END IF;
   END PROCESS;
END Behavior;
```

4:16 Decoder

Schematic



Waveform



Finite State Machine (FSM)

The FSM is used in the lab to produce outputs of various different states along with the student ID. This happens every time the clock is on a rising edge. Whenever a data input is 1, the next state will be considered for the output.

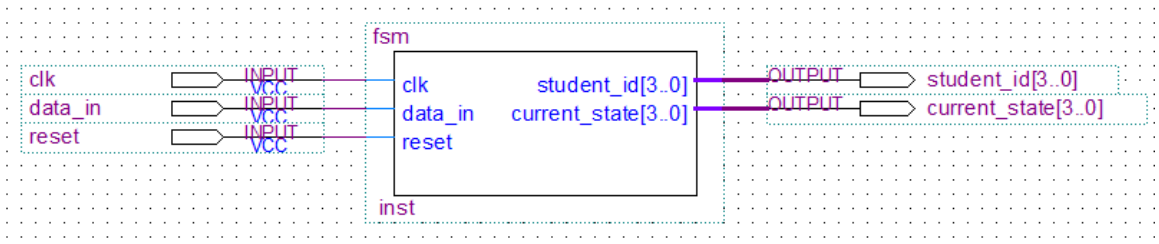
Truth Table

Present State (s_n)	Next State		Output	
	w=0	w=1	Student ID	Current State
s_0	s_0	s_4	0101 [5]	0000 [0]
s_4	s_4	s_3	0000 [0]	0100 [4]
s_3	s_3	s_2	0000 [0]	0011 [3]
s_2	s_2	s_1	1001 [9]	0010 [2]
s_1	s_1	s_8	0101 [5]	0001 [1]
s_8	s_8	s_7	0011 [3]	1000 [8]
s_7	s_7	s_6	0100 [4]	0111 [7]
s_6	s_6	s_5	0111 [7]	0110 [6]
s_5	s_5	s_0	0001 [1]	0101 [5]

VHDL Code

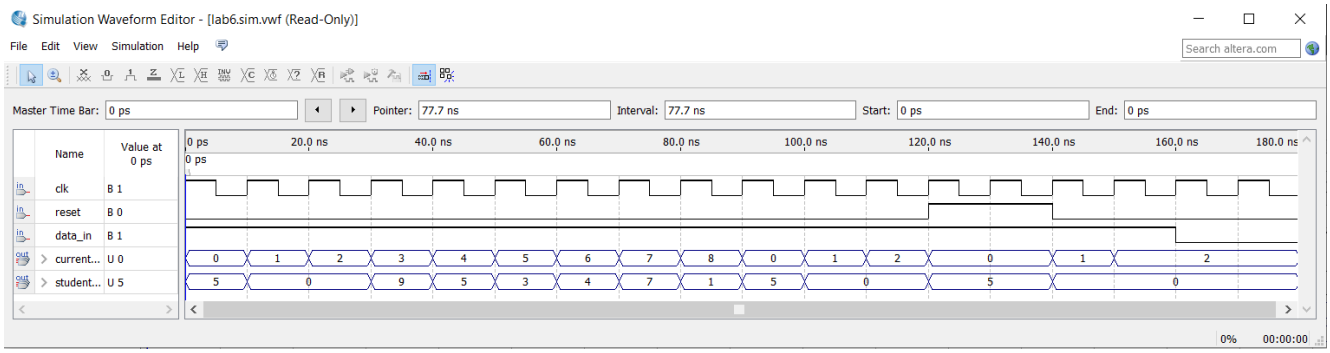
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fsm IS
   PORT ( clk, data_in, 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 Behavior of fsm IS
    type state_type is (s0, s1, s2, s3, s4, s5, s6, s7, s8);
    signal yfsm : state_type;

BEGIN
   p1: process(clk, reset)
       BEGIN
         if reset = '1' then
            yfsm <= s0;
         else if (clk'EVENT AND clk = '1') then
            case yfsm is
               when s0 => if data_in = '1' then yfsm <= s1; else yfsm <= s0; end if;
               when s1 => if data_in = '1' then yfsm <= s2; else yfsm <= s1; end if;
               when s2 => if data_in = '1' then yfsm <= s3; else yfsm <= s2; end if;
               when s3 => if data_in = '1' then yfsm <= s4; else yfsm <= s3; end if;
               when s4 => if data_in = '1' then yfsm <= s5; else yfsm <= s4; end if;
               when s5 => if data_in = '1' then yfsm <= s6; else yfsm <= s5; end if;
               when s6 => if data_in = '1' then yfsm <= s7; else yfsm <= s6; end if;
               when s7 => if data_in = '1' then yfsm <= s8; else yfsm <= s7; end if;
               when s8 => if data_in = '1' then yfsm <= s0; else yfsm <= s8; end if;
            end case;
         end if;
         end if;
    end process p1;

   p2: process(yfsm)
       BEGIN
         case yfsm is
               when s0 => student_id <= "0101";
               current_state <= "0000";
               when s1 => student_id <= "0000";
               current_state <= "0001";
               when s2 => student_id <= "0000";
               current_state <= "0010";
               when s3 => student_id <= "1001";
               current_state <= "0011";
               when s4 => student_id <= "0101";
               current_state <= "0100";
               when s5 => student_id <= "0011";
               current_state <= "0101";
               when s6 => student_id <= "0100";
               current_state <= "0110";
               when s7 => student_id <= "0111";
               current_state <= "0111";
               when s8 => student_id <= "0001";
               current_state <= "1000";
         end case;
    end process p2;
END Behavior;
```

Schematic



Waveform

When the clock is on a rising edge and the data input is 1, the current state changes. When reset is high, the current state goes back to the initial current state and continues from there.



ALU 1 – Problem Set 1

When making ALU 1, we have used two latches will store numbers A and B, which will be 34 and 71. An FSM that works on rising edge clock pulse will output my student ID which is 500953471 one digit a time. The current state output from the FSM will go into the decoder. Then, the outputs of all these components will go into the ALU and a series of functions will be performed and displayed onto seven-segment displays (SSEG). The output will be divided into 2 parts, the first four bits and the last four.

Table

This table shows the microcode's generated after the boolean operations are completed.

Function #	A (34)	B (71)	Boolean Operation	Result
1	0010 0010	0100 0111	Sum (A, B)	0110 1001
2	0010 0010	0100 0111	Diff (A, B)	-010 0101
3	0010 0010	0100 0111	NOT (A)	1101 1101
4	0010 0010	0100 0111	NAND (A, B)	1111 1101
5	0010 0010	0100 0111	NOR (A, B)	0001 1000
6	0010 0010	0100 0111	AND (A, B)	0000 0010
7	0010 0010	0100 0111	OR (A, B)	0110 0111
8	0010 0010	0100 0111	XOR (A, B)	0110 0101
9	0010 0010	0100 0111	XNOR (A, B)	1001 1010

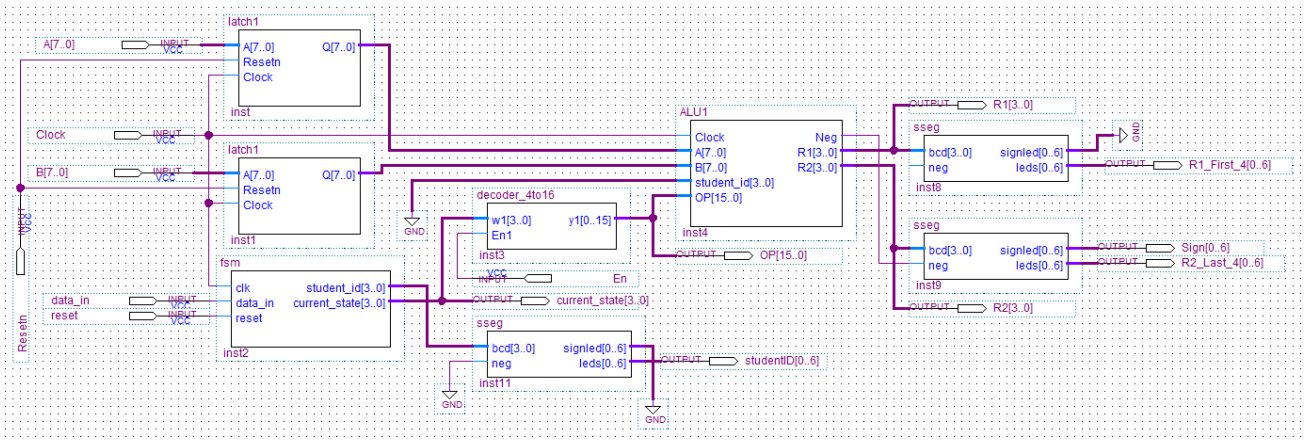
VHDL Code

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ALU1 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);
          R2         : out unsigned(3 downto 0));
end ALU1;

architecture calculation of ALU1 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
               case OP is
                  WHEN "0000000000000001" => Result <= (A+B); Neg <= '0';
                  WHEN "0000000000000010" => if A>B then Result <= (A-B); Neg <= '0'; else Result <=(B-A); Neg <= '1'; end if;
                  WHEN "0000000000000100" => Result <= not(A); Neg <= '0'; if (not A)<0 then neg <= '1'; end if;
                  WHEN "0000000000001000" => Result <= A nand B; if (A nand B)<0 then neg <= '1'; end if;
                  WHEN "0000000000010000" => Result <= A nor B; if (A nor B)<0 then neg <= '1'; end if;
                  WHEN "0000000000100000" => Result <= (A and B); if (A and B)<0 then neg <= '1'; end if;
                  WHEN "0000000001000000" => Result <= (A or B); if (A or B)<0 then neg <= '1'; end if;
                  WHEN "0000000010000000" => Result <= (A xor B); if (A xor B)<0 then neg <= '1'; end if;
                  WHEN "0000000100000000" => Result <= (A xnor B); if (A xnor B)<0 then neg <= '1'; end if;
                  WHEN OTHERS => Result <= Null ;
               end case;
            end if ;
      end process ;
   R1 <= Result(3 downto 0);
   R2 <= Result(7 downto 4);
end calculation;
```

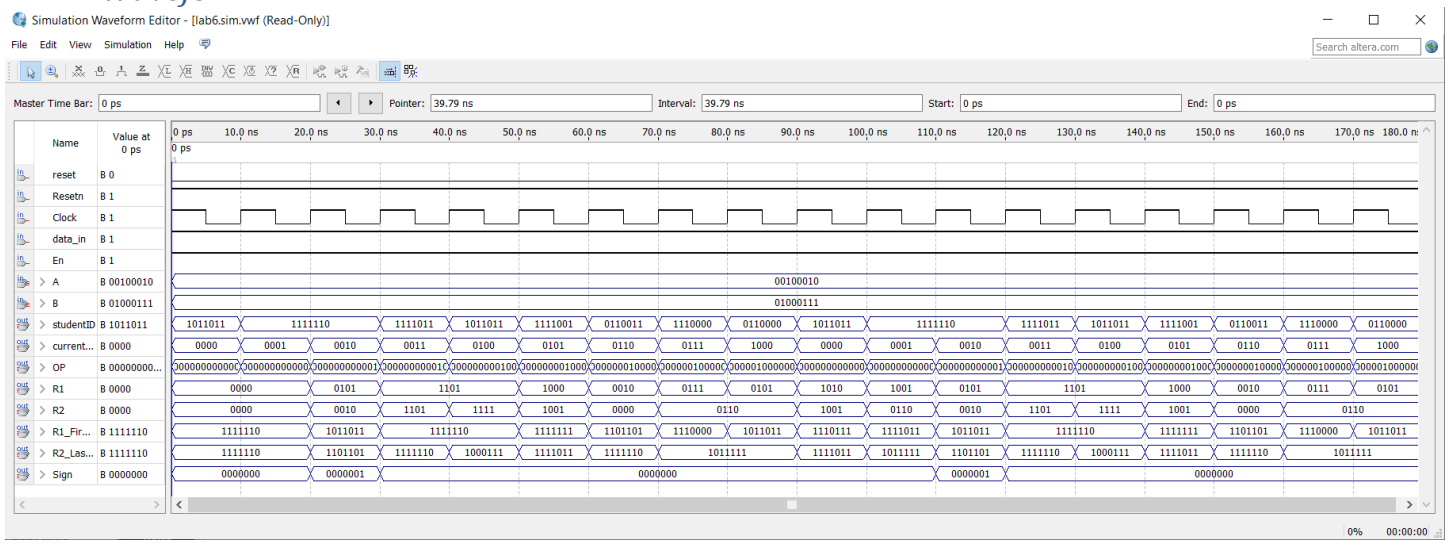
The SSEG that was used is as follows:

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

ENTITY sseg IS
   PORT ( bcd           : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          neg           : IN STD_LOGIC ;
          signled, leds  : OUT STD_LOGIC_VECTOR(0 TO 6) ) ;
END sseg;

ARCHITECTURE Behaviour OF sseg IS
   SIGNAL negvec : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
BEGIN
   p1 : PROCESS ( bcd )
   BEGIN
      CASE bcd IS             -- abcdefg
         WHEN "0000" => leds <= "1111110" ;
         WHEN "0001" => leds <= "0110000" ;
         WHEN "0010" => leds <= "1101101" ;
         WHEN "0011" => leds <= "1111001" ;
         WHEN "0100" => leds <= "0110011" ;
         WHEN "0101" => leds <= "1011011" ;
         WHEN "0110" => leds <= "1011111" ;
         WHEN "0111" => leds <= "1110000" ;
         WHEN "1000" => leds <= "1111111" ;
         WHEN "1001" => leds <= "1111011" ;
         WHEN "1010" => leds <= "1110111" ;
         WHEN "1011" => leds <= "1111111" ;
         WHEN "1100" => leds <= "1001110" ;
         WHEN "1101" => leds <= "1111110" ;
         WHEN "1110" => leds <= "1001111" ;
         WHEN "1111" => leds <= "1000111" ;
      END CASE;
      END PROCESS p1;

   p2 : PROCESS (neg)
   BEGIN
   negvec(0) <= neg ;
      CASE negvec IS
         WHEN "0001" => signled <= "0000001" ;
         WHEN OTHERS => signled <= "0000000" ;
      END CASE;
      END PROCESS p2;
END Behaviour ;
```

Schematic



Waveform



ALU 2 – Problem Set 2

This is similar to ALU 1 as it is modified to implement the functions in problem set 2.

Table

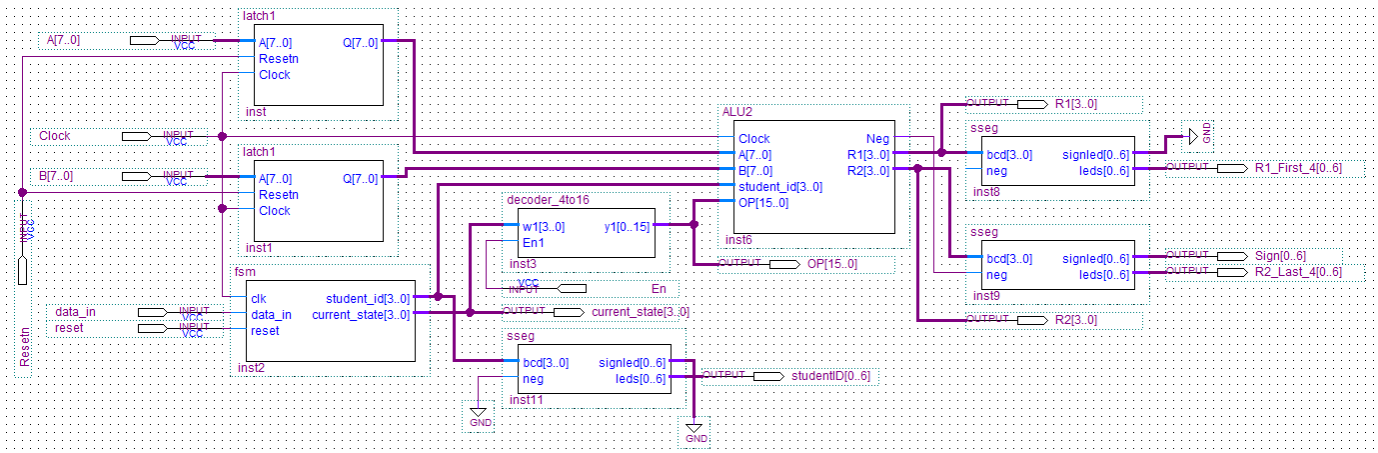
Function #	A (34)	B (71)	Operation/Function	Result
1	0010 0010	0100 0111	Replace odd bits of A with odd bits of B	0000 0010
2	0010 0010	0100 0111	Produce the result of NANDing A and B	1111 1101
3	0010 0010	0100 0111	Calculate the summation of A and B and decrease it by 5	0110 0100
4	0010 0010	0100 0111	Produce the 2's complement of B	1011 1001
5	0010 0010	0100 0111	Invert the even bits of B	0001 0011
6	0010 0010	0100 0111	Shift A to left by 2 bits, input bit= 1 (SHL)	1000 1011
7	0010 0010	0100 0111	Produce null on the output	0000 0000
8	0010 0010	0100 0111	Produce 2's complement of A	1101 1110
9	0010 0010	0100 0111	Rotate B to right by 2 bits (ROR)	1101 0001

VHDL Code

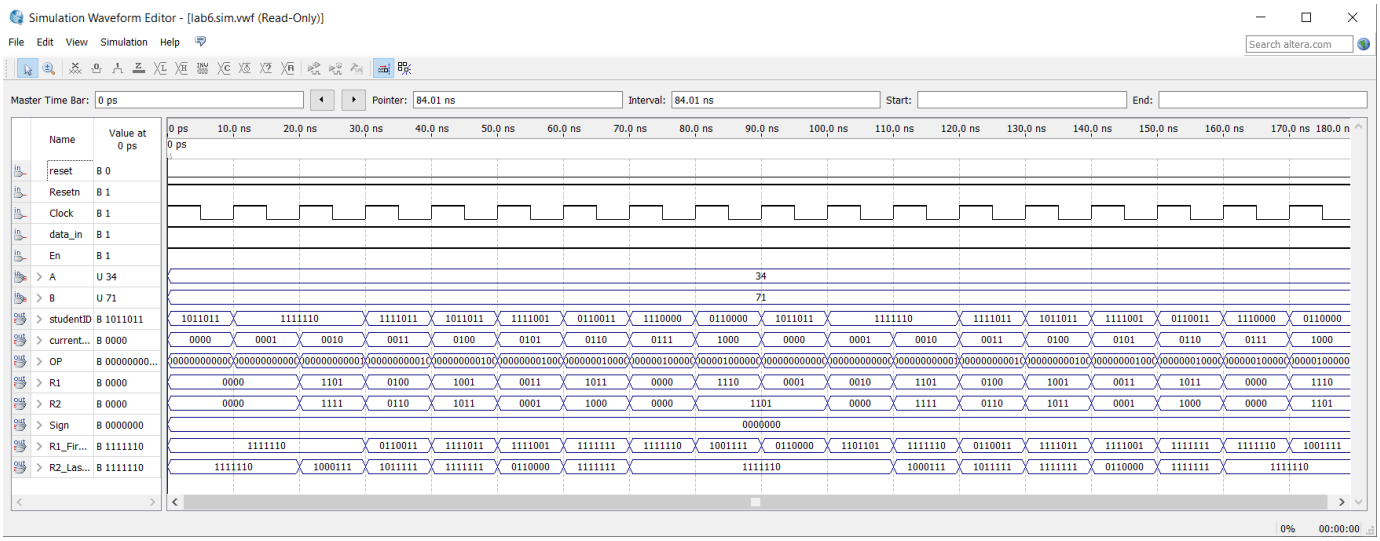
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

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);
          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
               case OP is
                  WHEN "0000000000000001" =>
                     Result(0) <= A(0);
                     Result(1) <= B(1);
                     Result(2) <= A(2);
                     Result(3) <= B(3);
                     Result(4) <= A(4);
                     Result(5) <= B(5);
                     Result(6) <= A(6);
                     Result(7) <= B(7);
                  WHEN "0000000000000010" =>
                     Result <= (A nand B);
                  WHEN "0000000000000100" =>
                     Result <= (A + B) - 5;
                  WHEN "0000000000001000" =>
                     Result <= not(B) + "00000001";
                  WHEN "0000000000010000" =>
                     Result(0) <= B(0);
                     Result(1) <= B(1);
                     Result(2) <= NOT B(2);
                     Result(3) <= B(3);
                     Result(4) <= NOT B(4);
                     Result(5) <= B(5);
                     Result(6) <= NOT B(6);
                     Result(7) <= B(7);
                  WHEN "0000000000100000" =>
                     Result(0) <= '1';
                     Result(1) <= '1';
                     Result(2) <= A(0);
                     Result(3) <= A(1);
                     Result(4) <= A(2);
                     Result(5) <= A(3);
                     Result(6) <= A(4);
                     Result(7) <= A(5);
                  WHEN "0000000001000000" =>
                     Result <= "00000000";
                  WHEN "0000000010000000" =>
                     Result <= not(A) + "00000001";
                  WHEN "0000000100000000" =>
                     Result(0) <= B(2);
                     Result(1) <= B(3);
                     Result(2) <= B(4);
                     Result(3) <= B(5);
                     Result(4) <= B(6);
                     Result(5) <= B(7);
                     Result(6) <= B(0);
                     Result(7) <= B(1);
                  WHEN OTHERS => Result <= Null ;
               end case;
            end if ;
      end process ;
   R1 <= Result(3 downto 0);
   R2 <= Result(7 downto 4);
end calculation;
```

Note: Same SSEG that was used in problem set 1 was used here.

Schematic



Waveform



ALU 3 – Problem Set 3

ALU 1 is again modified again to implement the function assigned in problem set 3. I was assigned function C which was:

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

Table

Student_ID	Number of 1s	Parity	Display
0101 [5]	2	Even	1110110 [n]
0000 [0]	0	Even	1110110 [n]
0000 [0]	0	Even	1110110 [n]
1001 [9]	2	Even	1110110 [n]
0101 [5]	2	Even	1110110 [n]
0011 [3]	2	Even	1110110 [n]
0100 [4]	1	Odd	0111011 [y]

0111 [7]	3	Odd	0111011 [y]
0001 [1]	1	Odd	0111011 [y]

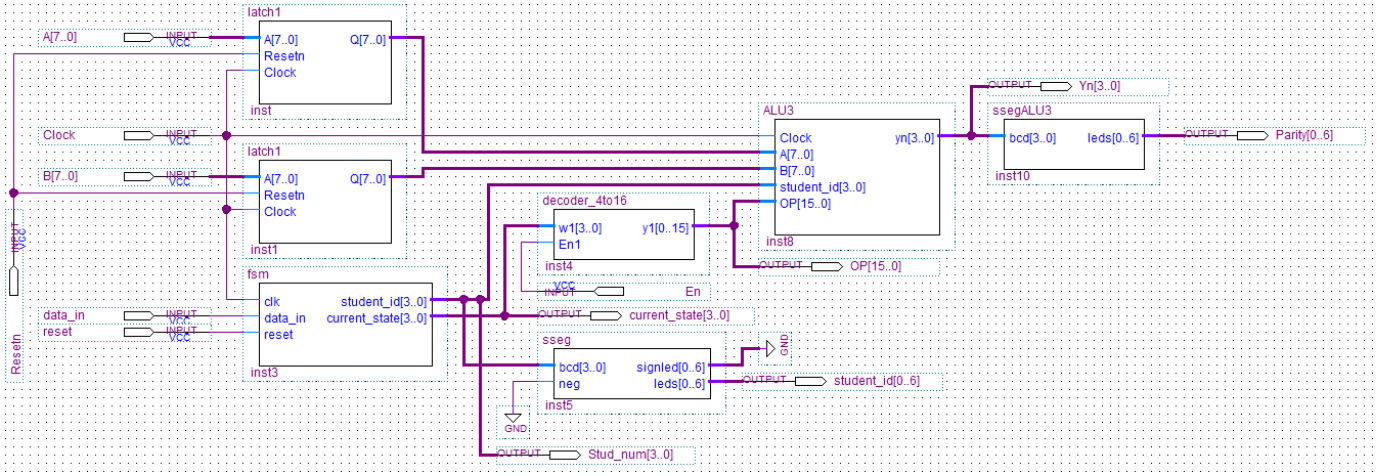
VHDL Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ALU3 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);
          yn         : out std_logic_vector(3 downto 0));
end ALU3;

architecture calculation of ALU3 is
    signal Reg1, Reg2         : unsigned(7 downto 0);
    signal parity             : std_logic;
    signal student            : std_logic_vector(3 downto 0);

begin
    Reg1 <= A;
    Reg2 <= B;
    student <= std_logic_vector(student_id);
    parity <= student(0) xor student(1) xor student(2) xor student(3) ;
      process(Clock, OP)
         begin
            if(rising_edge(Clock)) THEN
               case OP is
                  WHEN "0000000000000001" =>
                     if parity = '1'
                        then yn <= "0001" ;
                     else yn <= "0000" ;
                     end if;
                  WHEN "0000000000000010" =>
                     if parity = '1'
                        then yn <= "0001" ;
                     else yn <= "0000" ;
                     end if;
                  WHEN "0000000000000100" =>
                     if parity = '1'
                        then yn <= "0001" ;
                     else yn <= "0000" ;
                     end if;
                  WHEN "0000000000001000" =>
                     if parity = '1'
                        then yn <= "0001" ;
                     else yn <= "0000" ;
                     end if;
                  WHEN "0000000000010000" =>
                     if parity = '1'
                        then yn <= "0001" ;
                     else yn <= "0000" ;
                     end if;
                  WHEN "0000000000100000" =>
                     if parity = '1'
                        then yn <= "0001" ;
                     else yn <= "0000" ;
                     end if;
                  WHEN "0000000001000000" =>
                     if parity = '1'
                        then yn <= "0001" ;
                     else yn <= "0000" ;
                     end if;
                  WHEN "0000000010000000" =>
                     if parity = '1'
                        then yn <= "0001" ;
                     else yn <= "0000" ;
                     end if;
                  WHEN "0000000100000000" =>
                     if parity = '1'
                        then yn <= "0001" ;
                     else yn <= "0000" ;
                     end if;
                  WHEN OTHERS => yn <= Null ;
               end case;
            end if ;
      end process ;
end calculation;
```

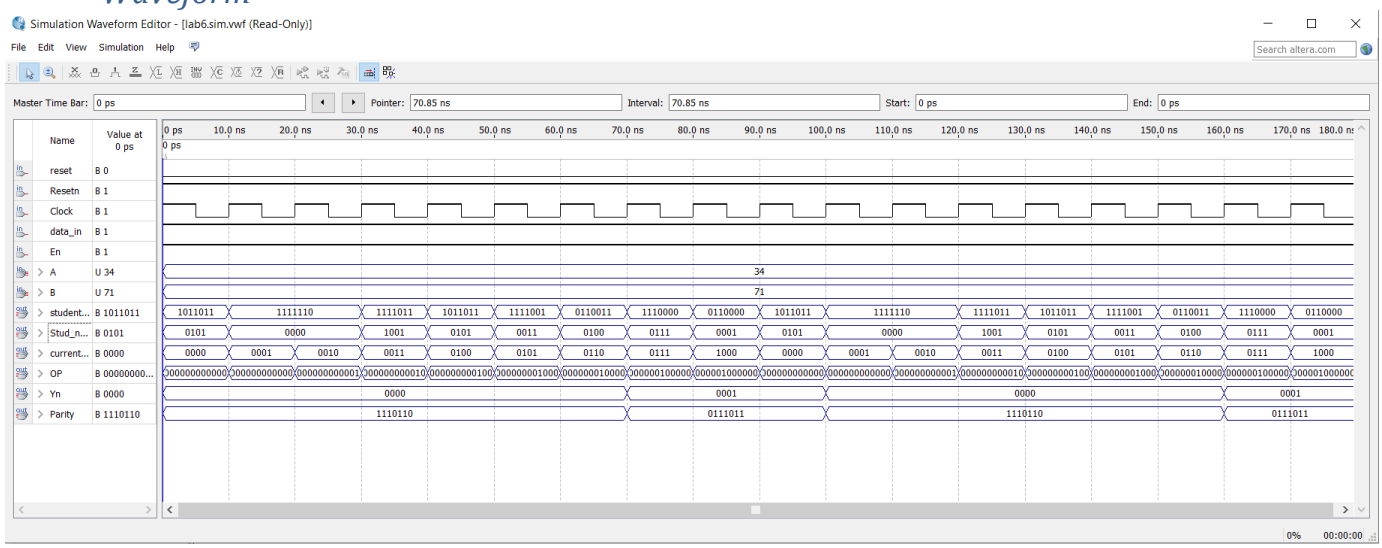
Note: Same SSEG as ALU1 was used in addition to a modified version of the SSEG was used and the code of the modified one is as follows:

```
1  LIBRARY ieee ;
2  USE ieee.std_logic_1164.all ;
3
4  ENTITY ssegALU3 IS
5     PORT ( bcd          : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
6            leds  : OUT STD_LOGIC_VECTOR(0 TO 6) ) ;
7  END ssegALU3;
8
9  ARCHITECTURE Behaviour OF ssegALU3 IS
10 BEGIN
11    p1 : PROCESS ( bcd )
12    BEGIN
13       CASE bcd IS             -- abcdefg
14          WHEN "0000" => leds <= "1110110" ;
15          WHEN "0001" => leds <= "0111011" ;
16          WHEN OTHERS => leds <= null ;
17       END CASE;
18       END PROCESS p1;
19
20
21 END Behaviour ;
```

Schematic



Waveform



Conclusion

Upon the completion of the lab, I have learned how to create a simple general processing unit using many different components. In addition, the ability to test each component separately was also gained as we had to match the output of the first component to the input of the second component to ensure its functionality. This lab provides an overall understanding of the entire course and allowed us to use all previous knowledge gained and apply it to this specific lab. Finally, we have been taught how to create a simple processor which can further be used for multiple devices.