Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	DIGITAL SYSTEMS
Course Number:	COE328 - 162
Semester/Year (e.g.F2016)	F2020

Instructor:	Dr. Prathap Siddavaatam

Assignment/Lab Number:	6
Assignment/Lab Title:	Simple General Purpose Processor

Submission Date:	Dec 4 2020
Due Date:	Dec 4 2020

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
BHAVSAR	SMIT	500957721	16	s.b.

Table of Contents

Introduction

The main goal of this lab was to tie everything we've done in these last few lab sessions together into one simple general purpose processor. Namely, understanding the ALU, its function, how to build it and how to function it so it outputs the desired result. This was done in a series of steps and different forms, ranging from building block diagrams to programming VHDL code to waveforms, but perhaps the initial step everyone first had to figure out is what each component is, its function, and how it'll be used in the lab to help the ALU reach the desired output. In my case, since my student number is 50095**7721**, taking the last four digits and storing them in the two input latches, my A = $(77)_{10}$ and my B = $(21)_{10}$. The components consisted of the Latches, 4:16 Decoder and the Moore FSM and I'll get into more specific detail on each component later on for each individual component.

Components: Latches, 4:16 Decoder, FSM

Component Descriptions:

______**Latches:** The purpose of the latches was to help store information. Since these take the form of the a D latch, they only have 1 input each which are A = $(77)_{10}$ and B = $(21)_{10}$.

4:16 Decoder: The 4:16 decoder is a type of logic circuit that reads in 4 inputs and outputs 16 that in a sense "decodes" the initial "encoded" information.

Moore FSM: Finally, the Moore FSM is a sequential circuit that uses various latches and flip flops for it to function. Since it is synchronous, the output is only the present state and it is completely independent of the input, it only depends on the clock

Component Truth/State Tables:

Latch Truth Table:

Clock	A/B	Z	Z’
0	0	d	d
0	1	d	d
1	0	0	1
1	1	1	0

4:16 Decoder Truth Table:

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

Moore FSM State Table:

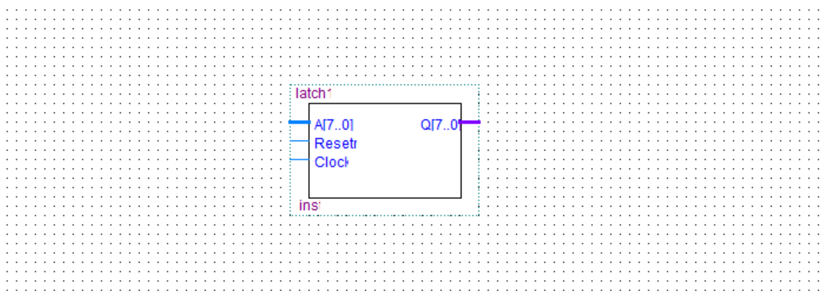
Present State	**Next State Data in (w = 0)**	**Next State Data in (w = 1)**	**Output (z) Current State**	**Output (z) Student #**
S0	S0	S1	C1 = 0000 [0]	D1 = 0101 [5]
S1	S1	S2	C2 = 0001 [1]	D2 = 0000 [0]
S2	S2	S3	C3 = 0010 [2]	D3 = 0000 [0]
S3	S3	S4	C4 = 0011 [3]	D4 = 1001 [9]
S4	S4	S5	C5 = 0100 [4]	D5 = 0101 [5]
S5	S5	S6	C6 = 0101 [5]	D6 = 0111 [7]
S6	S6	S7	C7 = 0110 [6]	D7 = 0111 [7]
S7	S7	S8	C8 = 0111 [7]	D8 = 0010 [2]
S8	S8	S0	C9 = 1000 [8]	D9 = 0001 [1]

Moore FSM State Assigned Table:

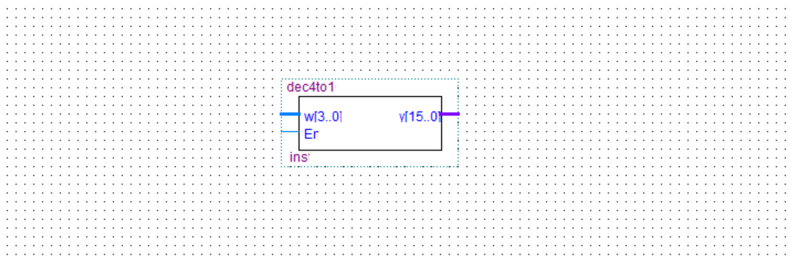
Present State	**Next State Data in (w = 0)**	**Next State Data in (w = 1)**	**Output (z) Current State**	**Output (z) Student #**
0000	0000	0001	C1 = 0000 [0]	D1 = 0101 [5]
0001	0001	0010	C2 = 0001 [1]	D2 = 0000 [0]
0010	0010	0011	C3 = 0010 [2]	D3 = 0000 [0]
0011	0011	0100	C4 = 0011 [3]	D4 = 1001 [9]
0100	0100	0101	C5 = 0100 [4]	D5 = 0101 [5]
0101	0101	0110	C6 = 0101 [5]	D6 = 0111 [7]
0110	0110	0111	C7 = 0110 [6]	D7 = 0111 [7]
0111	0111	1000	C8 = 0111 [7]	D8 = 0010 [2]
1000	1000	0000	C9 = 1000 [8]	D9 = 0001 [1]

Component Block Diagrams:

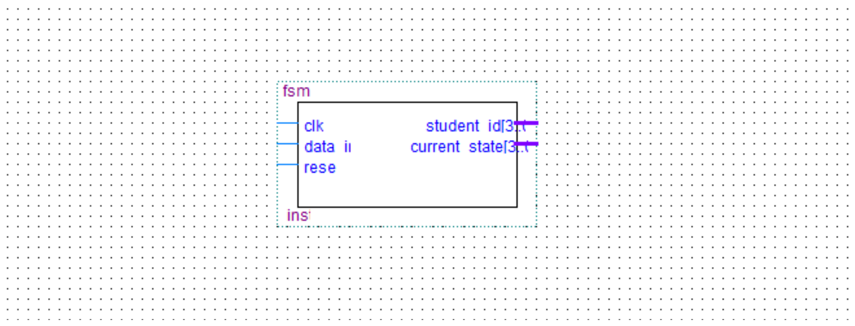
_______Latch Block Diagram:



_______4:16 Decoder Block Diagram:

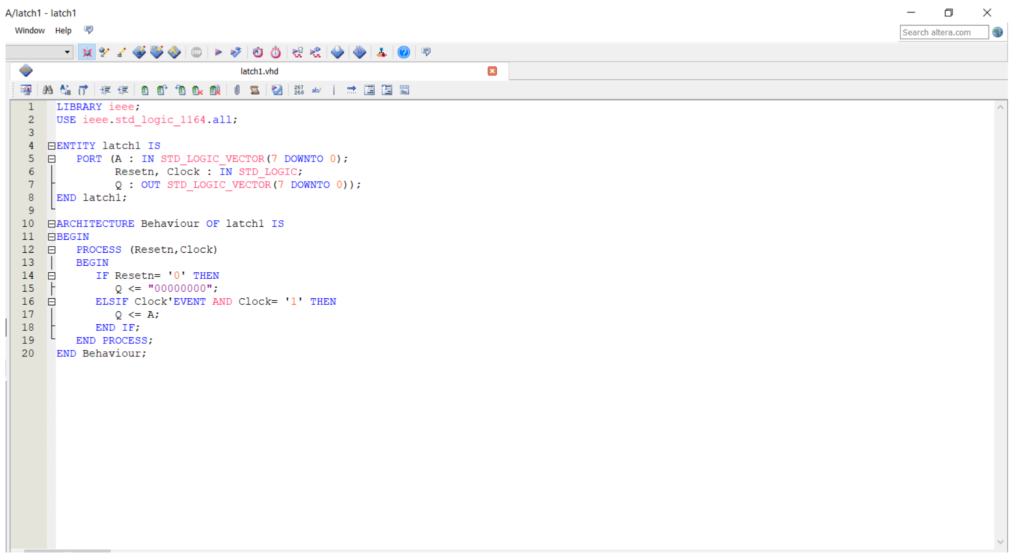


Moore FSM Block Diagram:



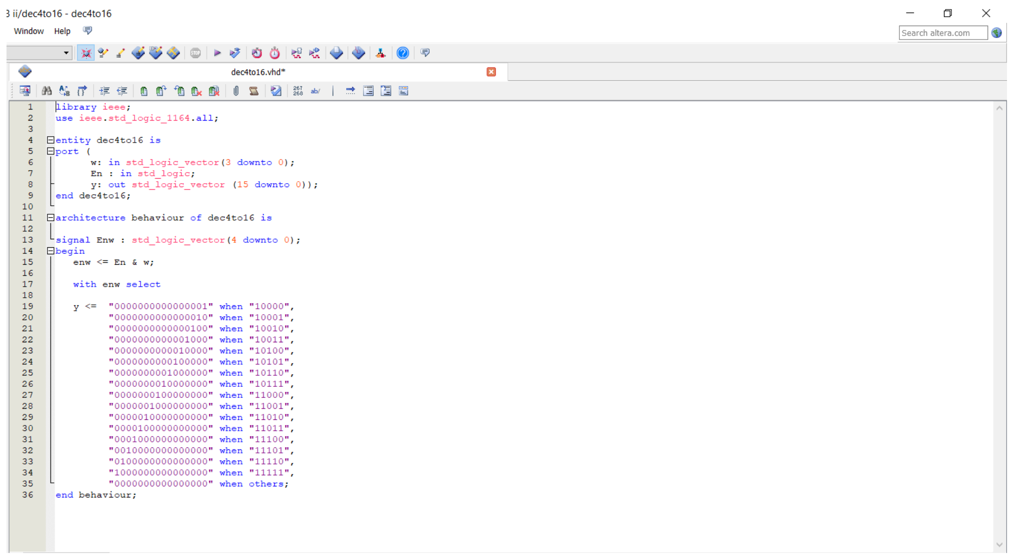
Components VHDL Codes:

______Latch VHDL Code:



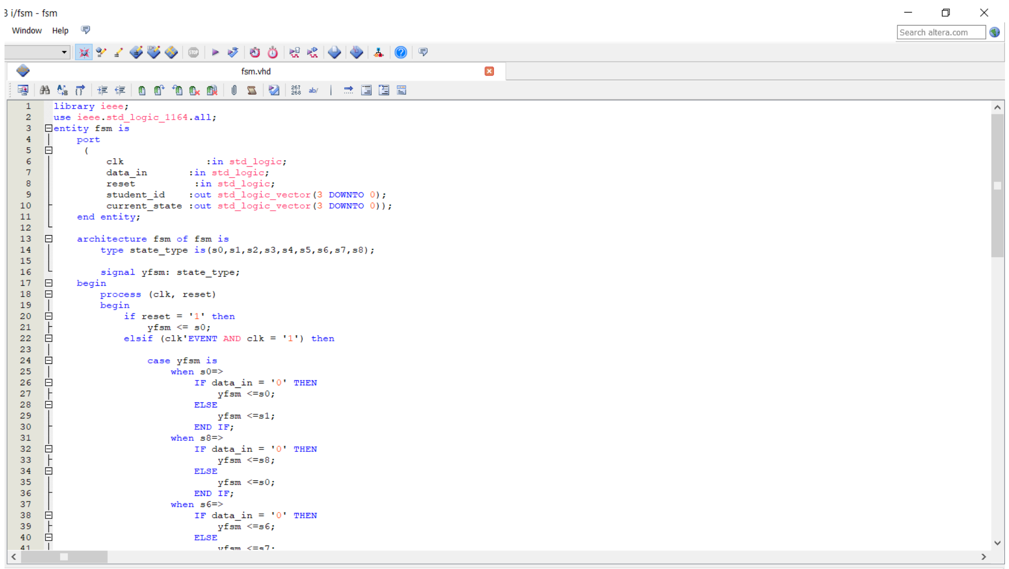
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY latch1 IS
   PORT (A : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         Resetn, Clock : IN STD_LOGIC;
         Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END latch1;

ARCHITECTURE Behaviour OF latch1 IS
BEGIN
   PROCESS (Resetn,Clock)
   BEGIN
      IF Resetn= '0' THEN
         Q <= "00000000";
      ELSIF Clock'EVENT AND Clock= '1' THEN
         Q <= A;
      END IF;
   END PROCESS;
END Behaviour;
```

______4:16 Decoder VHDL Code:

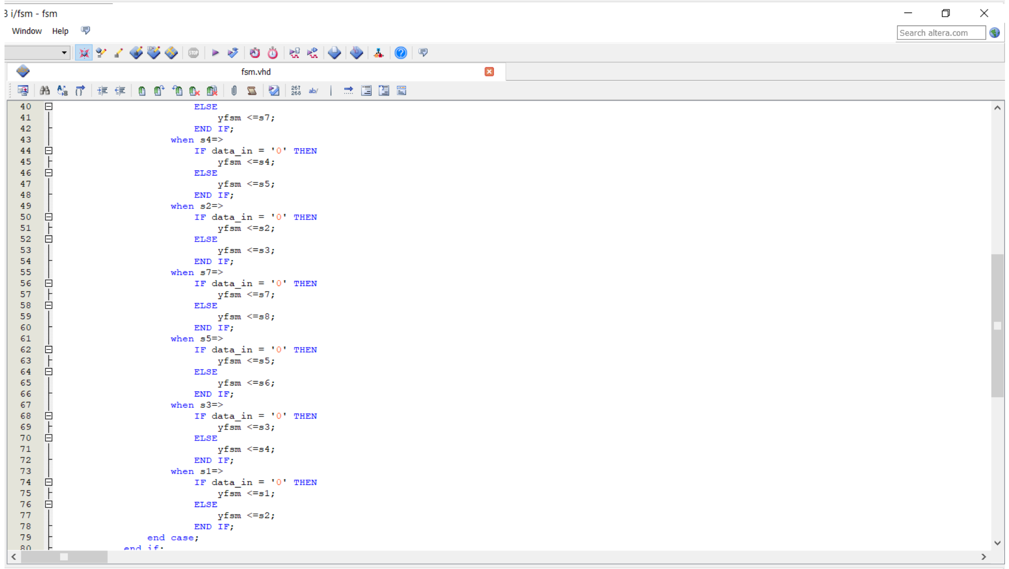


```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity dec4to16 is
port (
      w: in std_logic_vector(3 downto 0);
      En : in std_logic;
      y: out std_logic_vector (15 downto 0));
end dec4to16;

architecture behaviour of dec4to16 is

signal Enw : std_logic_vector(4 downto 0);
begin
   enw <= En & w;

   with enw select

   y <=  "0000000000000001" when "10000",
         "0000000000000010" when "10001",
         "0000000000000100" when "10010",
         "0000000000001000" when "10011",
         "0000000000010000" when "10100",
         "0000000000100000" when "10101",
         "0000000001000000" when "10110",
         "0000000010000000" when "10111",
         "0000000100000000" when "11000",
         "0000001000000000" when "11001",
         "0000010000000000" when "11010",
         "0000100000000000" when "11011",
         "0001000000000000" when "11100",
         "0010000000000000" when "11101",
         "0100000000000000" when "11110",
         "1000000000000000" when "11111",
         "0000000000000000" when others;
end behaviour;
```

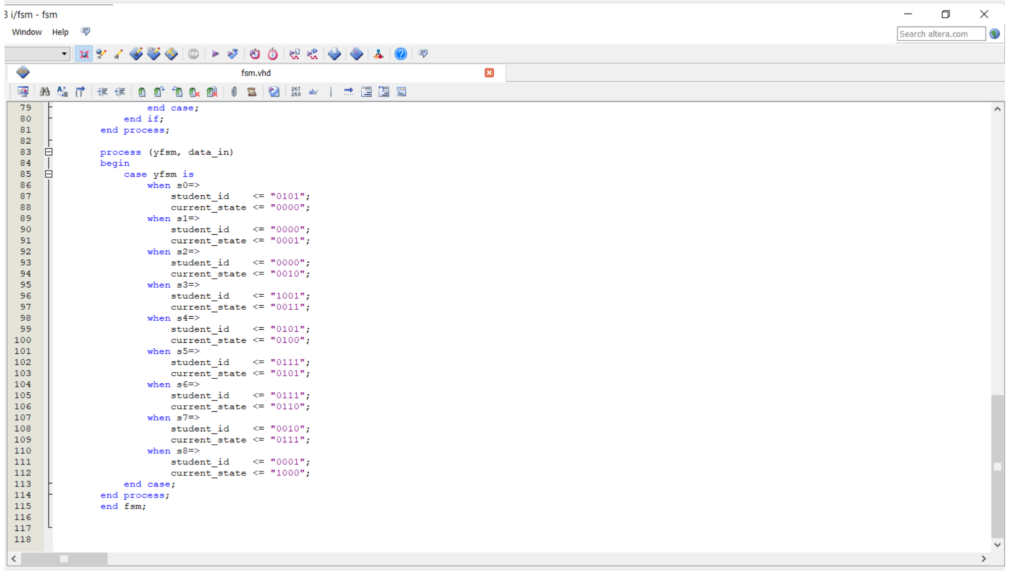
Moore FSM VHDL Code:



```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity fsm 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 fsm 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 s8=>
                        IF data_in = '0' THEN
                            yfsm <=s8;
                        ELSE
                            yfsm <=s0;
                        END IF;
                    when s6=>
                        IF data_in = '0' THEN
                            yfsm <=s6;
                        ELSE
```

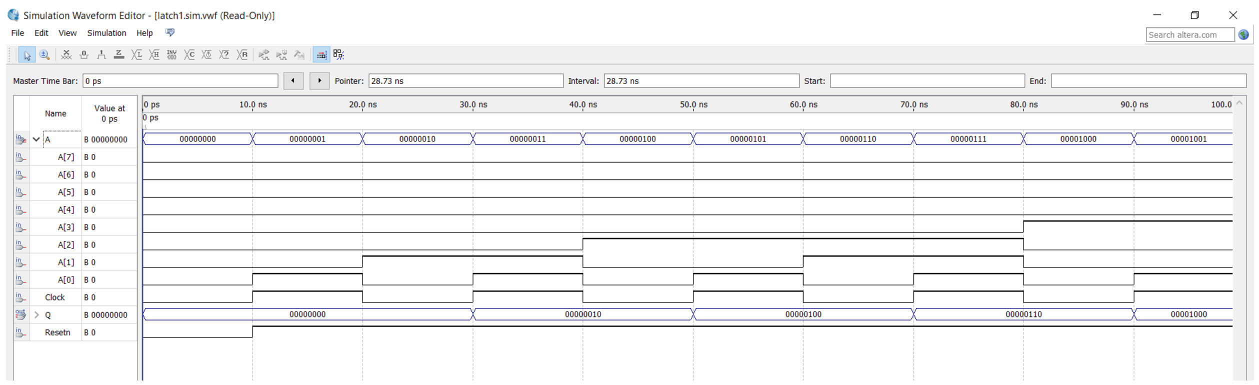


```vhdl
                        ELSE
                            yfsm <=s7;
                        END IF;
                    when s4=>
                        IF data_in = '0' THEN
                            yfsm <=s4;
                        ELSE
                            yfsm <=s5;
                        END IF;
                    when s2=>
                        IF data_in = '0' THEN
                            yfsm <=s2;
                        ELSE
                            yfsm <=s3;
                        END IF;
                    when s7=>
                        IF data_in = '0' THEN
                            yfsm <=s7;
                        ELSE
                            yfsm <=s8;
                        END IF;
                    when s5=>
                        IF data_in = '0' THEN
                            yfsm <=s5;
                        ELSE
                            yfsm <=s6;
                        END IF;
                    when s3=>
                        IF data_in = '0' THEN
                            yfsm <=s3;
                        ELSE
                            yfsm <=s4;
                        END IF;
                    when s1=>
                        IF data_in = '0' THEN
                            yfsm <=s1;
                        ELSE
                            yfsm <=s2;
                        END IF;
                end case;
```



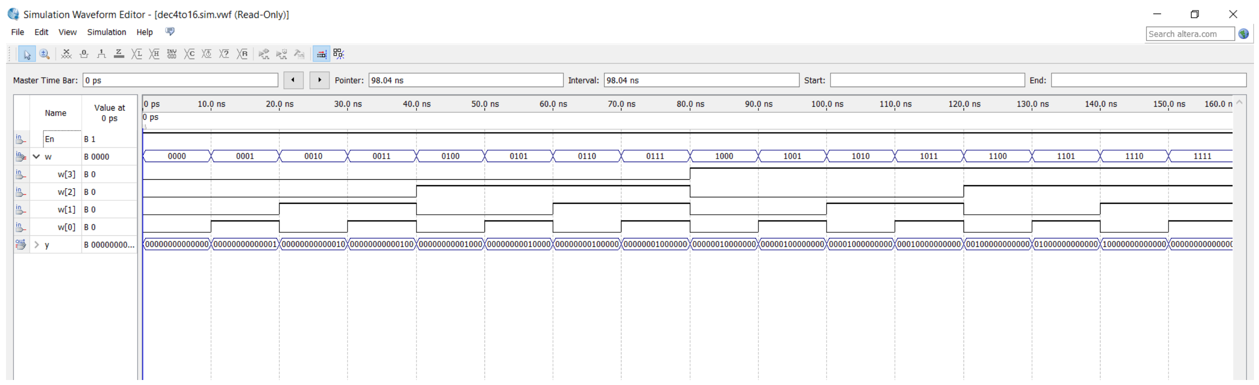
```vhdl
                end case;
            end if;
        end process;

        process (yfsm, data_in)
        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    <= "0111";
                    current_state <= "0101";
                when s6=>
                    student_id    <= "0111";
                    current_state <= "0110";
                when s7=>
                    student_id    <= "0010";
                    current_state <= "0111";
                when s8=>
                    student_id    <= "0001";
                    current_state <= "1000";
            end case;
        end process;
        end fsm;
```

Latch Waveform:


Simulation Waveform Editor - [latch1.sim.vwf (Read-Only)]

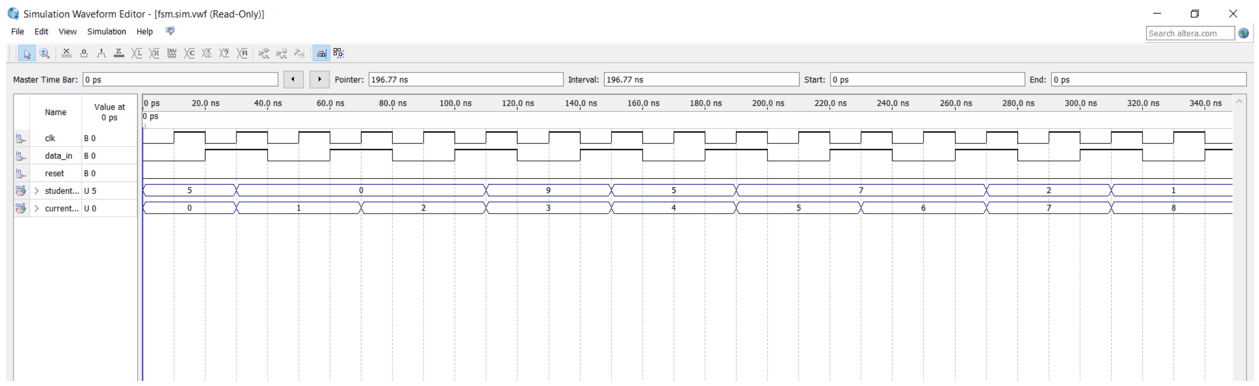
Master Time Bar: 0 ps Pointer: 28.73 ns Interval: 28.73 ns Start: End:

4:16 Decoder Waveform:


Simulation Waveform Editor - [dec4to16.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: 98.04 ns Interval: 98.04 ns Start: End:

Moore FSM Waveform:


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

Master Time Bar: 0 ps Pointer: 196.77 ns Interval: 196.77 ns Start: 0 ps End: 0 ps

File Edit View Simulation Help
Name
Value at 0 ps
clk B 0
data_in B 0
reset B 0
student... U 5
current... U 0
5 0 9 5 7 2 1
0 1 2 3 4 5 6 7 8
0 ps 20.0 ns 40.0 ns 60.0 ns 80.0 ns 100.0 ns 120.0 ns 140.0 ns 160.0 ns 180.0 ns 200.0 ns 220.0 ns 240.0 ns 260.0 ns 280.0 ns 300.0 ns 320.0 ns 340.0 ns

ALU for Problem set 1

Description of the ALU and its functions:

Function i: Addition of A and B

Function ii: Subtraction of A and B

Function iii: “A” inverse

Function iv: Boolean NAND of A and B

Function v: Boolean NOR of A and B

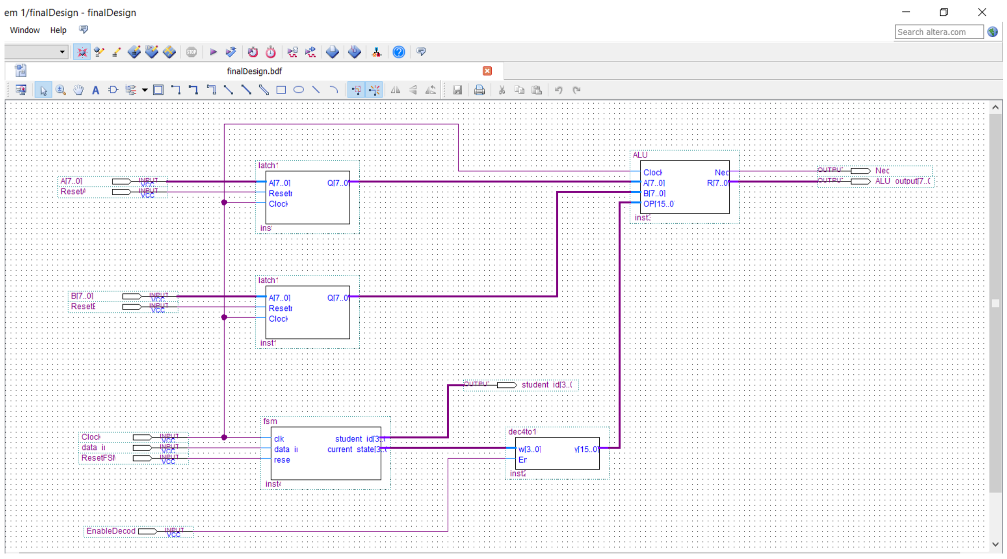
Function vi: Boolean AND of A and B

Function vii: Boolean OR of A and B

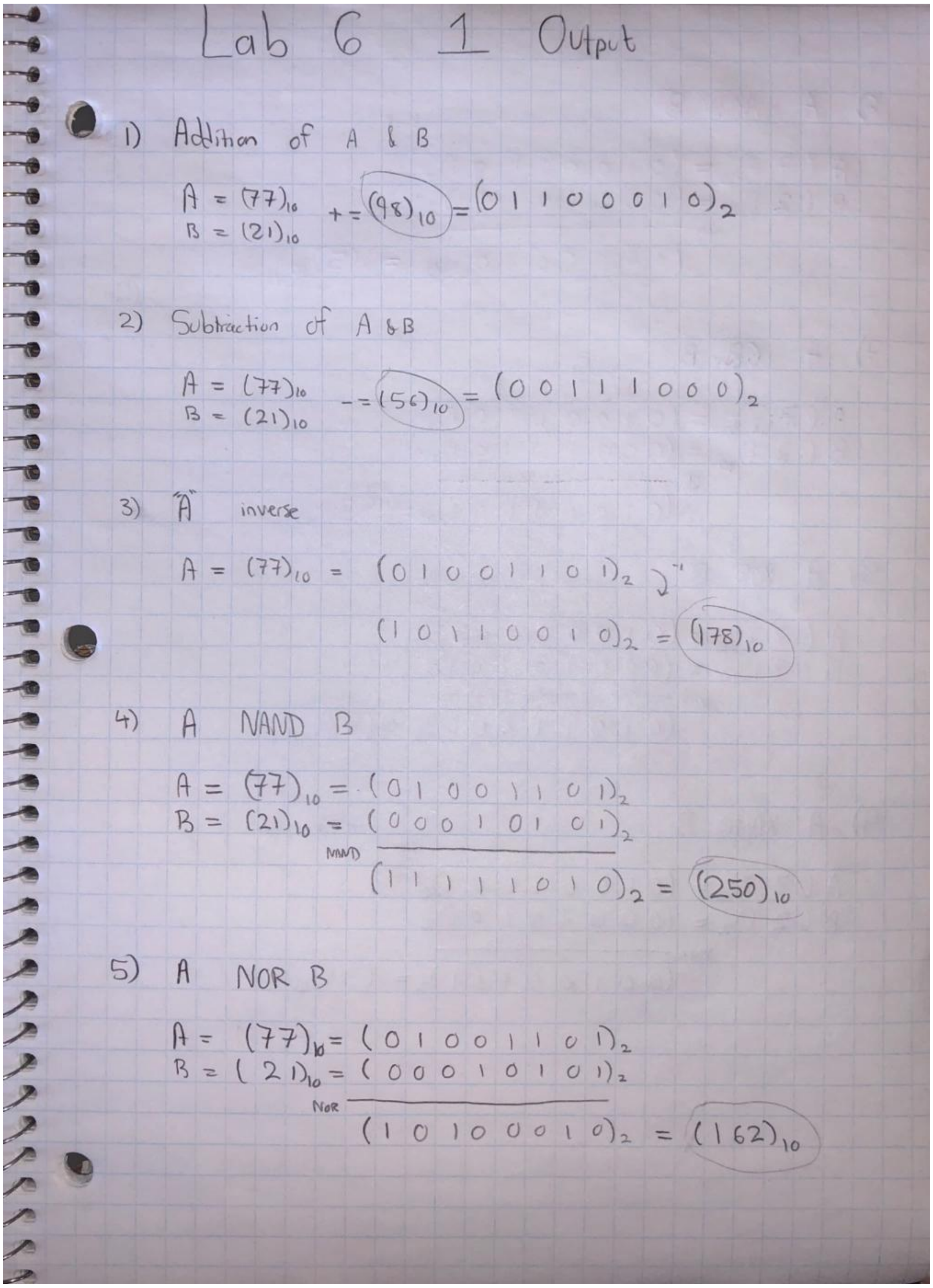
Function viii: Boolean XOR of A and B

Function ix: Boolean XNOR of A and B

Block Diagram for ALU:



Calculations for ALU Functions:



Lab 6 1 Output

1) Addition of A & B

$A = (77)_{10}$ $+ = \boxed{(98)_{10}} = (01100010)_2$
$B = (21)_{10}$

2) Subtraction of A & B

$A = (77)_{10}$ $- = \boxed{(56)_{10}} = (00111000)_2$
$B = (21)_{10}$

3) "$\bar{A}$" inverse

$A = (77)_{10} = (01001101)_2 \downarrow^{-1}$

$(10110010)_2 = \boxed{(178)_{10}}$

4) A NAND B

$A = (77)_{10} = (01001101)_2$
$B = (21)_{10} = (00010101)_2$
NAND
$(11111010)_2 = \boxed{(250)_{10}}$

5) A NOR B

$A = (77)_{10} = (01001101)_2$
$B = (21)_{10} = (00010101)_2$
NOR
$(10100010)_2 = \boxed{(162)_{10}}$

6) A AND B

$$
\begin{array}{lcl}
A\ (77)_{10} & = & (0\,1\,0\,0\,1\,1\,0\,1)_2 \\
B\ (21)_{10} & = & (0\,0\,0\,1\,0\,1\,0\,1)_2 \\
\hline
 & \text{AND} & (0\,0\,0\,0\,0\,1\,0\,1)_2 = \boxed{(5)_{10}}
\end{array}
$$

7) A OR B

$$
\begin{array}{lcl}
A\ (77)_{10} & = & (0\,1\,0\,0\,1\,1\,0\,1)_2 \\
B\ (21)_{10} & = & (0\,0\,0\,1\,0\,1\,0\,1)_2 \\
\hline
 & \text{OR} & (0\,1\,0\,1\,1\,1\,0\,1)_2 = \boxed{(93)_{10}}
\end{array}
$$

8) A XOR B

$$
\begin{array}{lcl}
A\ (77)_{10} & = & (0\,1\,0\,0\,1\,1\,0\,1)_2 \\
B\ (21)_{10} & = & (0\,0\,0\,1\,0\,1\,0\,1)_2 \\
\hline
 & \text{XOR} & (0\,1\,0\,1\,1\,0\,0\,0)_2 = \boxed{(88)_{10}}
\end{array}
$$

9) A XNOR B

$$
\begin{array}{lcl}
A\ (77)_2 & = & (0\,1\,0\,0\,1\,1\,0\,1)_2 \\
B\ (21)_2 & = & (0\,0\,0\,1\,0\,1\,0\,1)_2 \\
\hline
 & \text{XNOR} & (1\,0\,1\,0\,0\,1\,1\,1)_2 = \boxed{(167)_{10}}
\end{array}
$$

Purpose of inputs & outputs of ALU:

Inputs:

A, B : 8-bit inputs from Latches A and B

OP : 16-bit inputs from Moore fsm

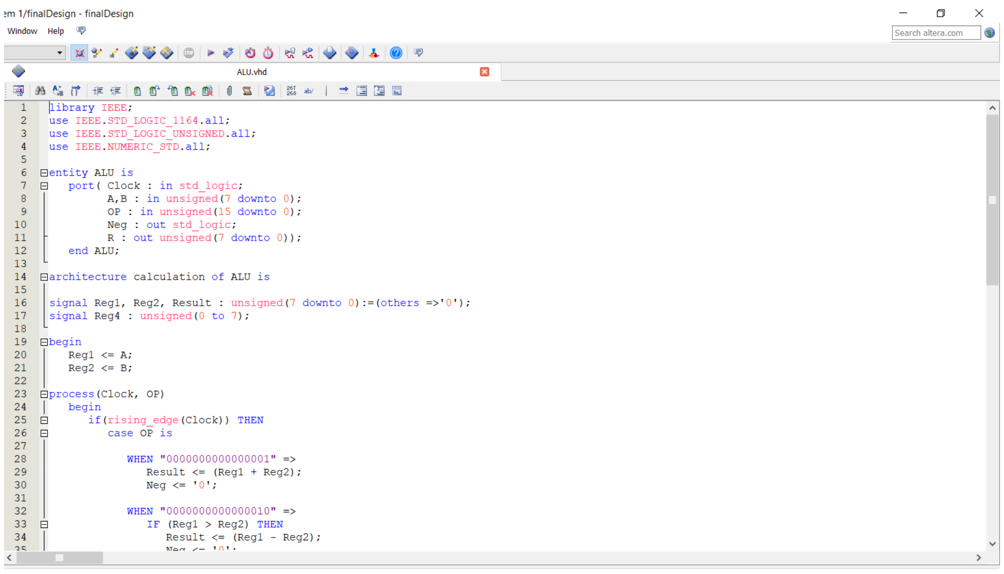
Clock : Signals the clock to synchronize everything

Outputs:

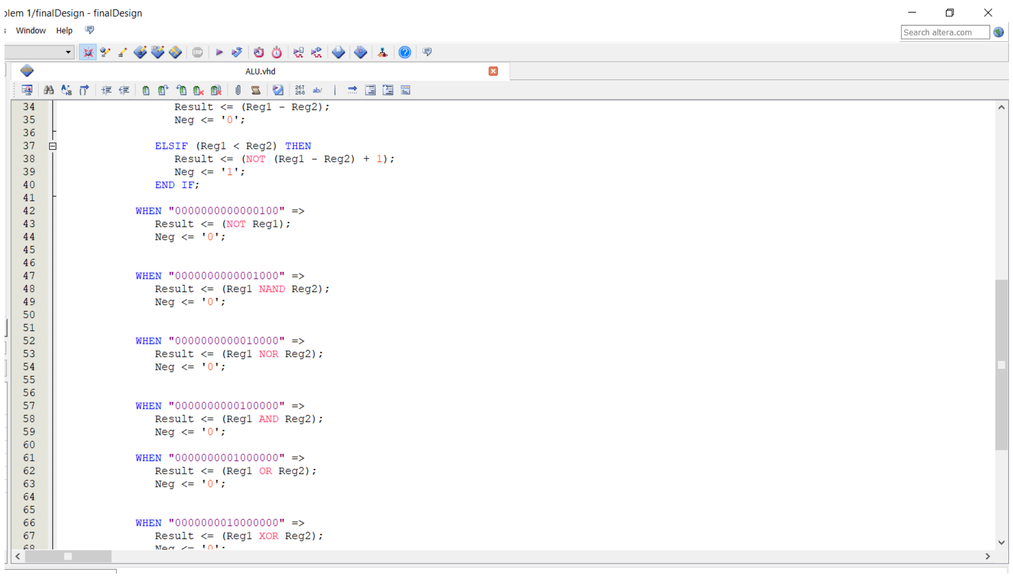
ALU_output : (self-explanatory) but outputs the ALU functions in waveform format

Neg : Outputs a wave whose value is 1 if it negative number or remains 0 if positive

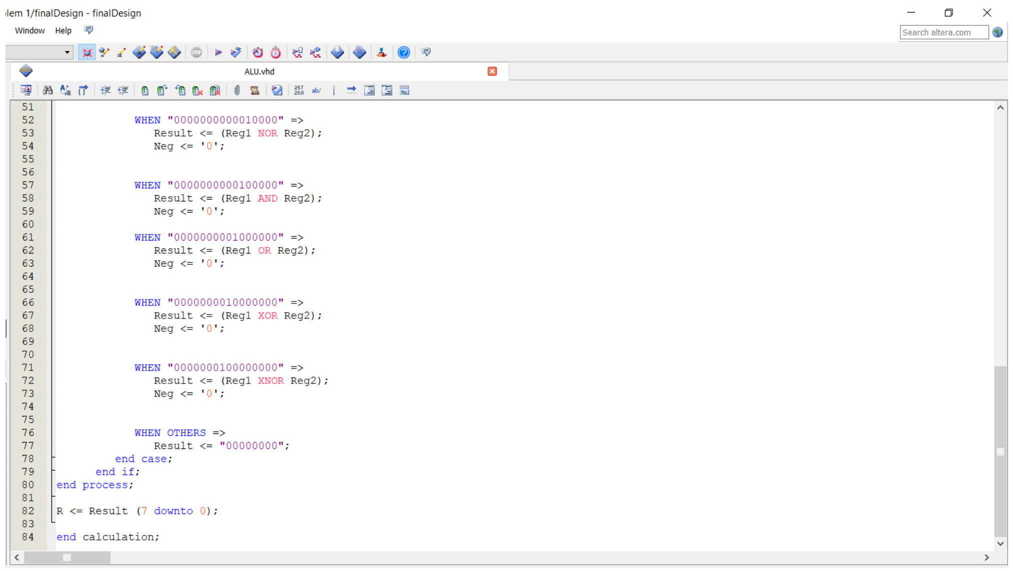
VHDL Codes of ALU:



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

entity ALU is
   port( Clock : in std_logic;
         A,B : in unsigned(7 downto 0);
         OP : in unsigned(15 downto 0);
         Neg : out std_logic;
         R : out unsigned(7 downto 0));
   end ALU;

architecture calculation of ALU 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 <= (Reg1 + Reg2);
               Neg <= '0';

            WHEN "0000000000000010" =>
               IF (Reg1 > Reg2) THEN
                  Result <= (Reg1 - Reg2);
                  Neg <= '0';
```

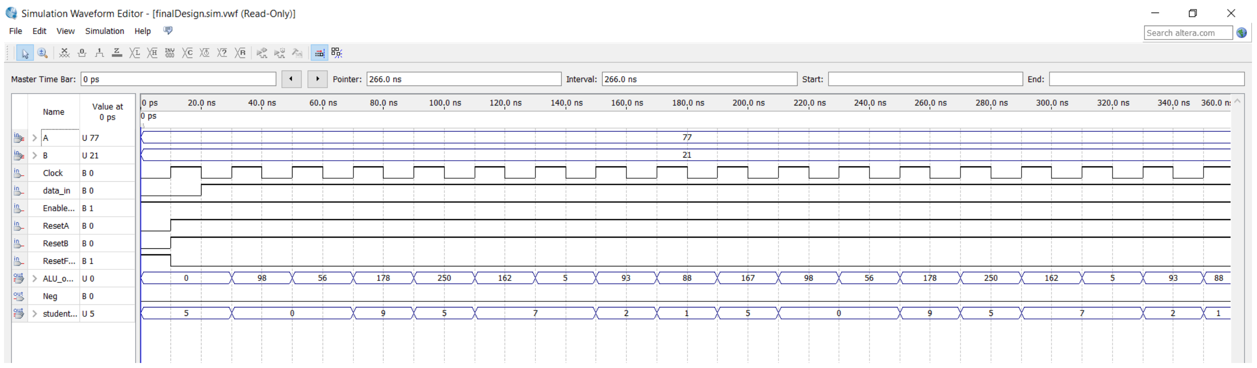


```
                Result <= (Reg1 - Reg2);
                Neg <= '0';

            ELSIF (Reg1 < Reg2) THEN
                Result <= (NOT (Reg1 - Reg2) + 1);
                Neg <= '1';
            END IF;

        WHEN "0000000000000100" =>
            Result <= (NOT Reg1);
            Neg <= '0';


        WHEN "0000000000001000" =>
            Result <= (Reg1 NAND Reg2);
            Neg <= '0';


        WHEN "0000000000010000" =>
            Result <= (Reg1 NOR Reg2);
            Neg <= '0';


        WHEN "0000000000100000" =>
            Result <= (Reg1 AND Reg2);
            Neg <= '0';

        WHEN "0000000001000000" =>
            Result <= (Reg1 OR Reg2);
            Neg <= '0';


        WHEN "0000000010000000" =>
            Result <= (Reg1 XOR Reg2);
            Neg <= '0';
```



```

        WHEN "0000000000010000" =>
            Result <= (Reg1 NOR Reg2);
            Neg <= '0';


        WHEN "0000000000100000" =>
            Result <= (Reg1 AND Reg2);
            Neg <= '0';

        WHEN "0000000001000000" =>
            Result <= (Reg1 OR Reg2);
            Neg <= '0';


        WHEN "0000000010000000" =>
            Result <= (Reg1 XOR Reg2);
            Neg <= '0';


        WHEN "0000000100000000" =>
            Result <= (Reg1 XNOR Reg2);
            Neg <= '0';


        WHEN OTHERS =>
            Result <= "00000000";
      end case;
   end if;
end process;

R <= Result (7 downto 0);

end calculation;
```

Waveform for ALU Problem 1:



ALU for Problem set 2F

Description of the ALU 2F and its functions:

Function i: Decrement B by 9

Function ii: Swap the lower and upper 4 bits of B

Function iii: Shift A to left by 2 bits; input bit = 0 (SHL)

Function iv: Boolean NAND of A and B

Function v: Find the greater value between A and B and output the value

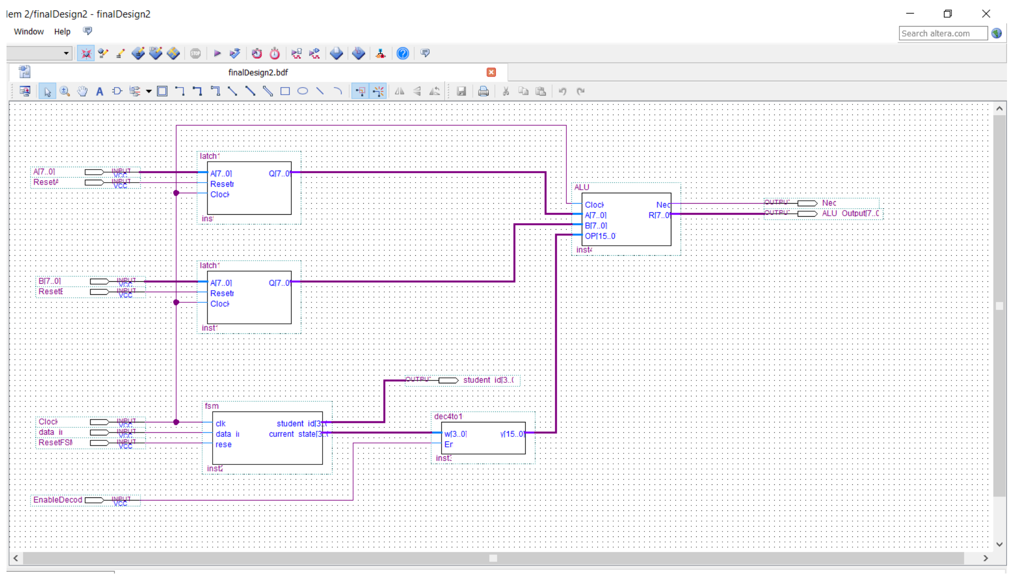
Function vi: Invert the even bits of B

Function vii: Produce null on the output

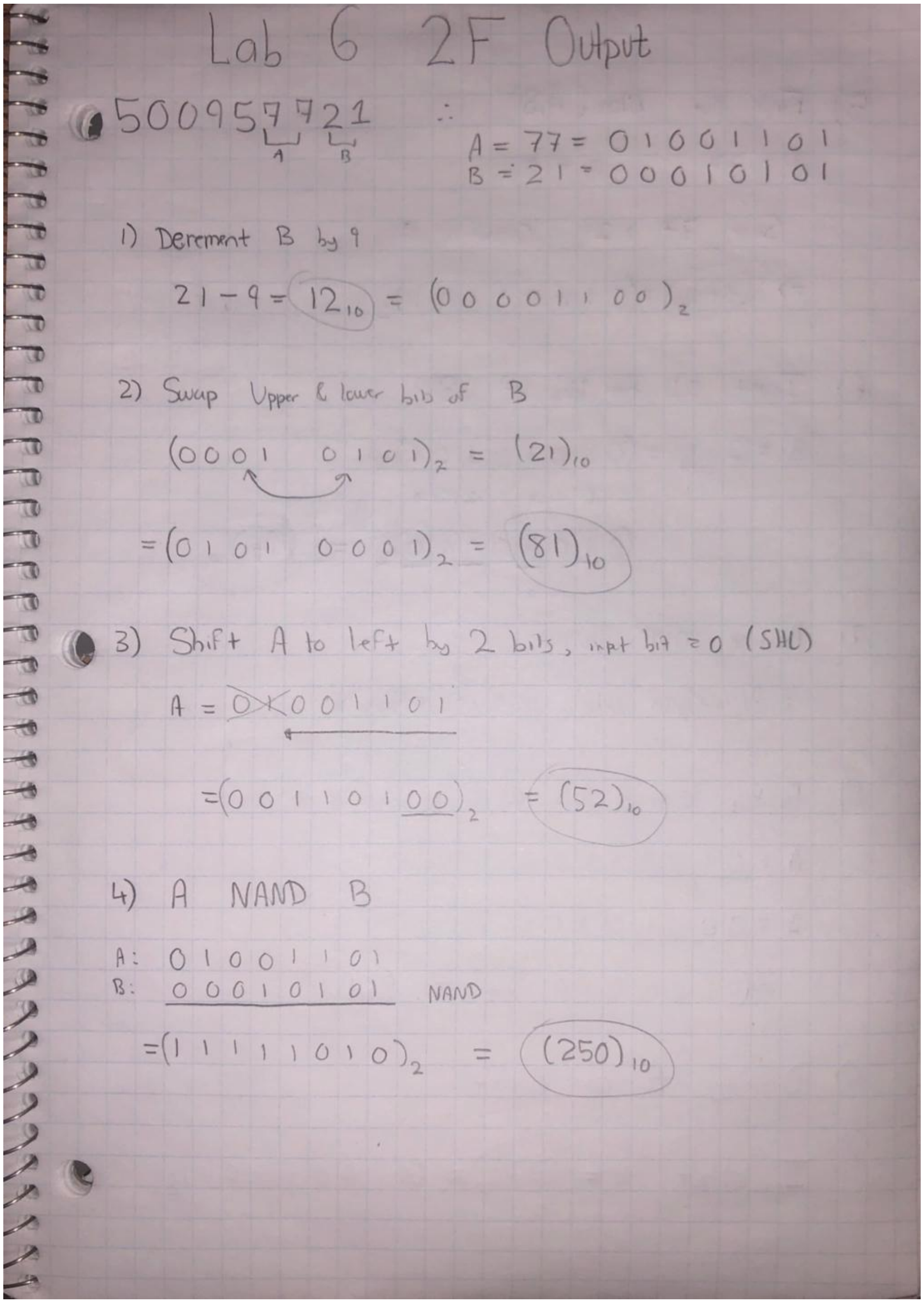
Function viii: Replace the upper four bits of B by upper four bits of A

Function ix: Display A on the output

Block Diagram for ALU 2F:



Calculations for ALU 2F Functions:



Lab 6 2F Output

$50095\underbrace{77}_{A}\underbrace{21}_{B}$ ∴

$A = 77 = 01001101$

$B = 21 = 00010101$

1) Derement B by 9

$21 - 9 = \boxed{12_{10}} = (00001100)_2$

2) Swap Upper & lower bits of B

$(0001\ \ 0101)_2 = (21)_{10}$

$= (0101\ \ 0001)_2 = \boxed{(81)_{10}}$

3) Shift A to left by 2 bits, input bit = 0 (SHL)

$A = \cancel{01}001101$

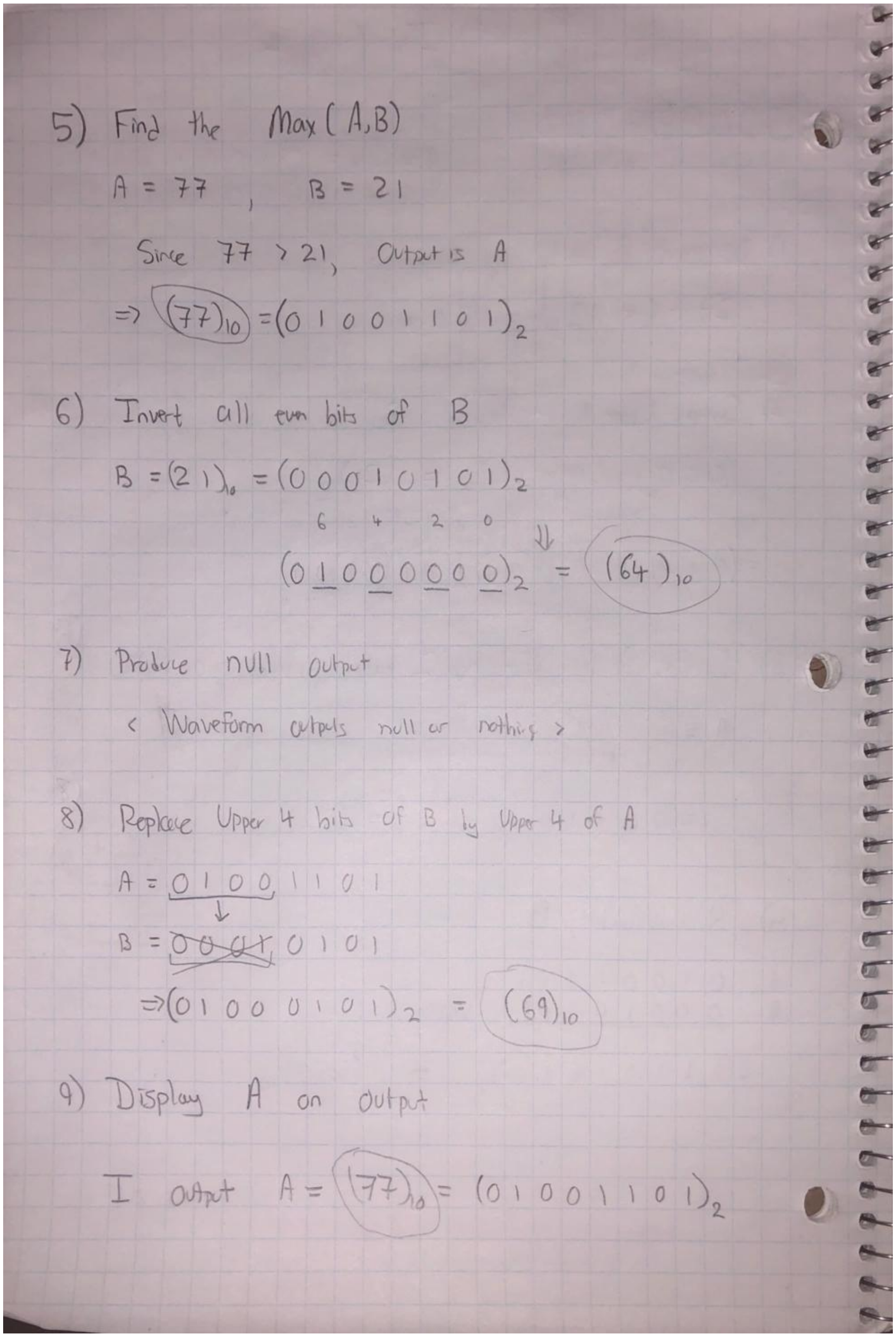
$= (001101\underline{00})_2 = \boxed{(52)_{10}}$

4) A NAND B

A: 01001101

B: 00010101 NAND

$= (11111010)_2 = \boxed{(250)_{10}}$



5) Find the Max(A,B)

$A = 77$, $B = 21$

Since $77 > 21$, Output is A

$\Rightarrow \boxed{(77)_{10}} = (0\,1\,0\,0\,1\,1\,0\,1)_2$

6) Invert all even bits of B

$B = (21)_{10} = (0\,0\,0\,1\,0\,1\,0\,1)_2$

6 4 2 0

$\Downarrow$

$(0\,\underline{1}\,0\,\underline{0}\,0\,\underline{0}\,0\,\underline{0})_2 = \boxed{(64)_{10}}$

7) Produce null output

< Waveform outputs null or nothing >

8) Replace Upper 4 bits of B by Upper 4 of A

$A = \underline{0\,1\,0\,0},\,1\,1\,0\,1$

$\downarrow$

$B = \cancel{0\,0\,0\,1},\,0\,1\,0\,1$

$\Rightarrow (0\,1\,0\,0\,0\,1\,0\,1)_2 = \boxed{(69)_{10}}$

9) Display A on output

I output $A = \boxed{(77)_{10}} = (0\,1\,0\,0\,1\,1\,0\,1)_2$

Purpose of inputs & outputs of ALU 2F:

Inputs:

A, B : 8-bit inputs from Latches A and B

OP : 16-bit inputs from Moore fsm

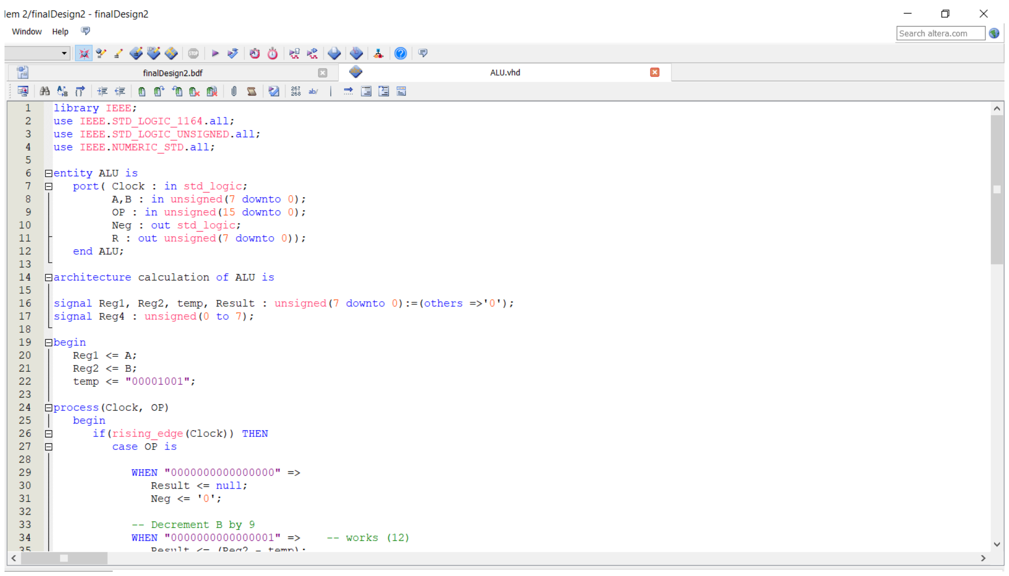
Clock : Signals the clock to synchronize everything

Outputs:

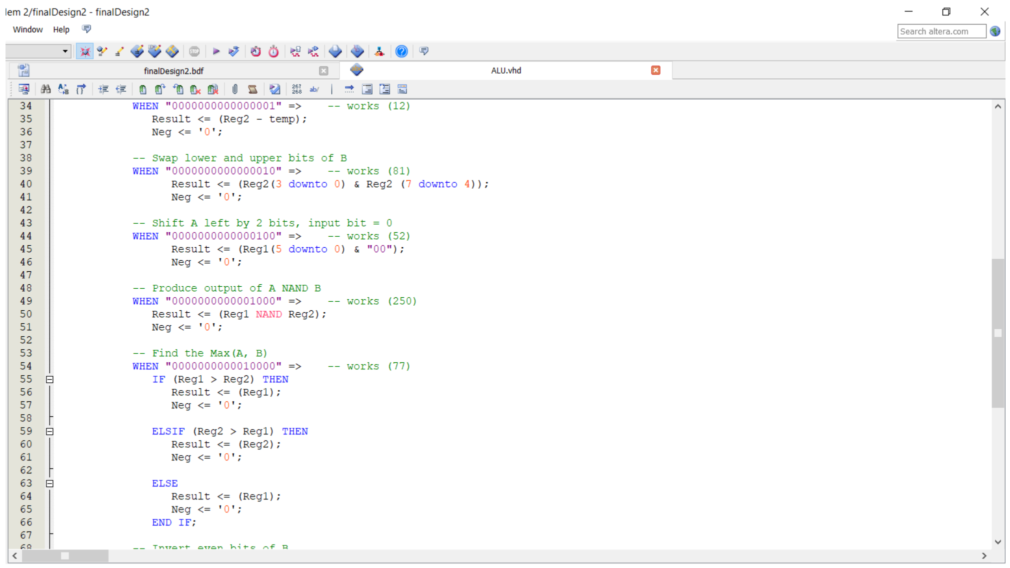
ALU_output : (self-explanatory) but outputs the ALU functions in waveform format

Neg : Outputs a wave whose value is 1 if it negative number or remains 0 if positive

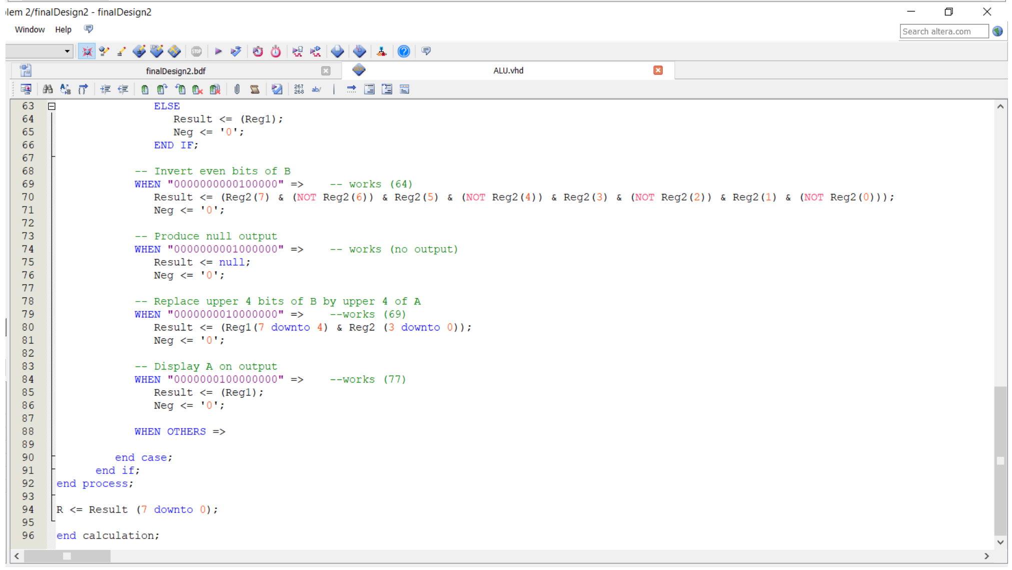
VHDL Codes of ALU 2F:



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

entity ALU is
   port( Clock : in std_logic;
         A,B : in unsigned(7 downto 0);
         OP : in unsigned(15 downto 0);
         Neg : out std_logic;
         R : out unsigned(7 downto 0));
   end ALU;

architecture calculation of ALU is

signal Reg1, Reg2, temp, Result : unsigned(7 downto 0):=(others =>'0');
signal Reg4 : unsigned(0 to 7);

begin
   Reg1 <= A;
   Reg2 <= B;
   temp <= "00001001";

process(Clock, OP)
   begin
      if(rising_edge(Clock)) THEN
         case OP is

            WHEN "0000000000000000" =>
               Result <= null;
               Neg <= '0';

            -- Decrement B by 9
            WHEN "0000000000000001" =>     -- works (12)
```

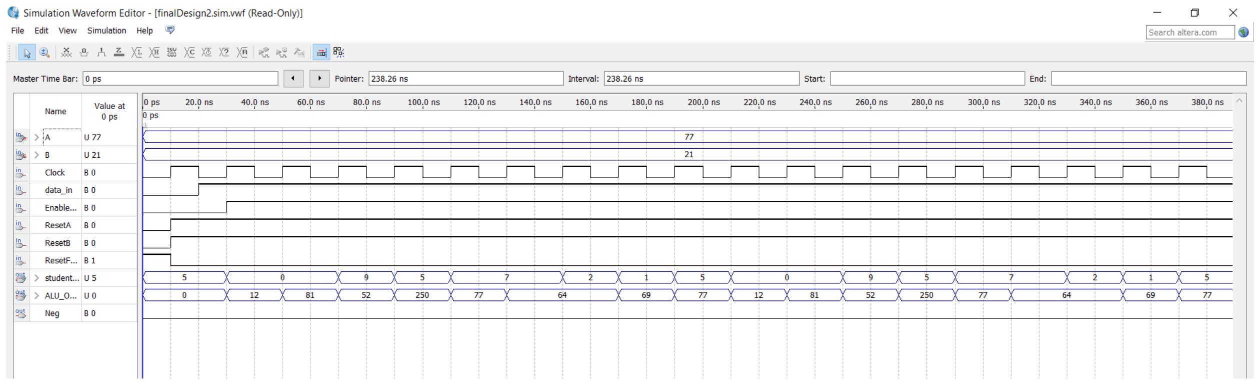


```vhdl
                WHEN "0000000000000001" =>     -- works (12)
                   Result <= (Reg2 - temp);
                   Neg <= '0';

                -- Swap lower and upper bits of B
                WHEN "0000000000000010" =>     -- works (81)
                      Result <= (Reg2(3 downto 0) & Reg2 (7 downto 4));
                      Neg <= '0';

                -- Shift A left by 2 bits, input bit = 0
                WHEN "0000000000000100" =>     -- works (52)
                      Result <= (Reg1(5 downto 0) & "00");
                      Neg <= '0';

                -- Produce output of A NAND B
                WHEN "0000000000001000" =>     -- works (250)
                   Result <= (Reg1 NAND Reg2);
                   Neg <= '0';

                -- Find the Max(A, B)
                WHEN "0000000000010000" =>     -- works (77)
                   IF (Reg1 > Reg2) THEN
                      Result <= (Reg1);
                      Neg <= '0';

                   ELSIF (Reg2 > Reg1) THEN
                      Result <= (Reg2);
                      Neg <= '0';

                   ELSE
                      Result <= (Reg1);
                      Neg <= '0';
                   END IF;

                -- Invert even bits of B
```



```vhdl
                   ELSE
                      Result <= (Reg1);
                      Neg <= '0';
                   END IF;

                -- Invert even bits of B
                WHEN "0000000000100000" =>     -- works (64)
                   Result <= (Reg2(7) & (NOT Reg2(6)) & Reg2(5) & (NOT Reg2(4)) & Reg2(3) & (NOT Reg2(2)) & Reg2(1) & (NOT Reg2(0)));
                   Neg <= '0';

                -- Produce null output
                WHEN "0000000001000000" =>     -- works (no output)
                   Result <= null;
                   Neg <= '0';

                -- Replace upper 4 bits of B by upper 4 of A
                WHEN "0000000010000000" =>     --works (69)
                   Result <= (Reg1(7 downto 4) & Reg2 (3 downto 0));
                   Neg <= '0';

                -- Display A on output
                WHEN "0000000100000000" =>     --works (77)
                   Result <= (Reg1);
                   Neg <= '0';

                WHEN OTHERS =>

            end case;
        end if;
end process;

R <= Result (7 downto 0);

end calculation;
```

Waveform for ALU Problem 2F:



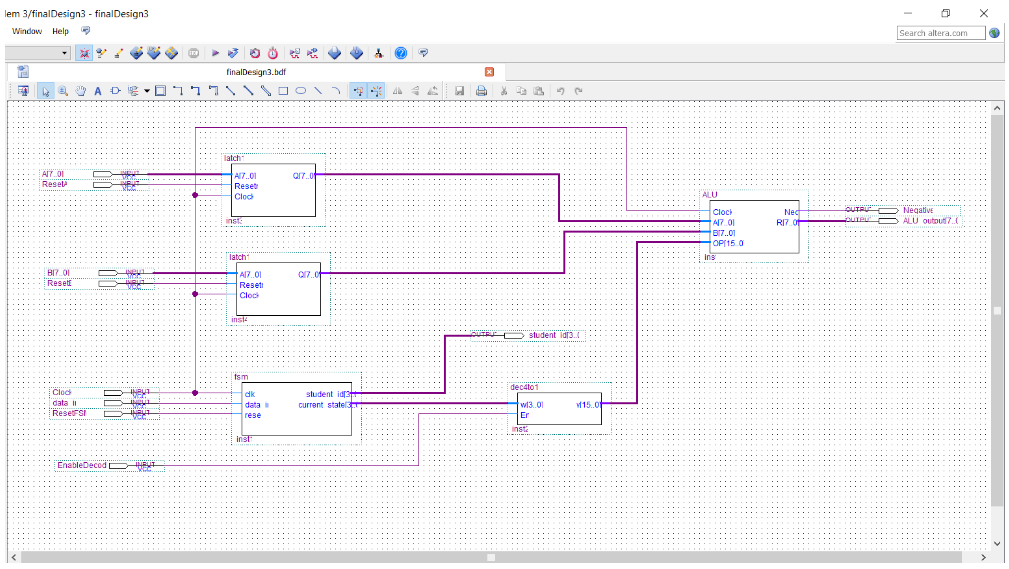
It is to note that my waveform for “ALU_Output” has a 0 for 20 ns. I tried fixing this issue but I couldn't. The output is still correct from then onwards, I just do not know how to fix the 20ns clock lag.

ALU for Problem set 3B

Description of the ALU 3B and its functions:

For each microcode instruction, display ‘y’ if the FSM output (student_id) is even and ‘n’ otherwise. (In our case, we used “00000001” for ‘y’ and “00000000” for ‘n’).

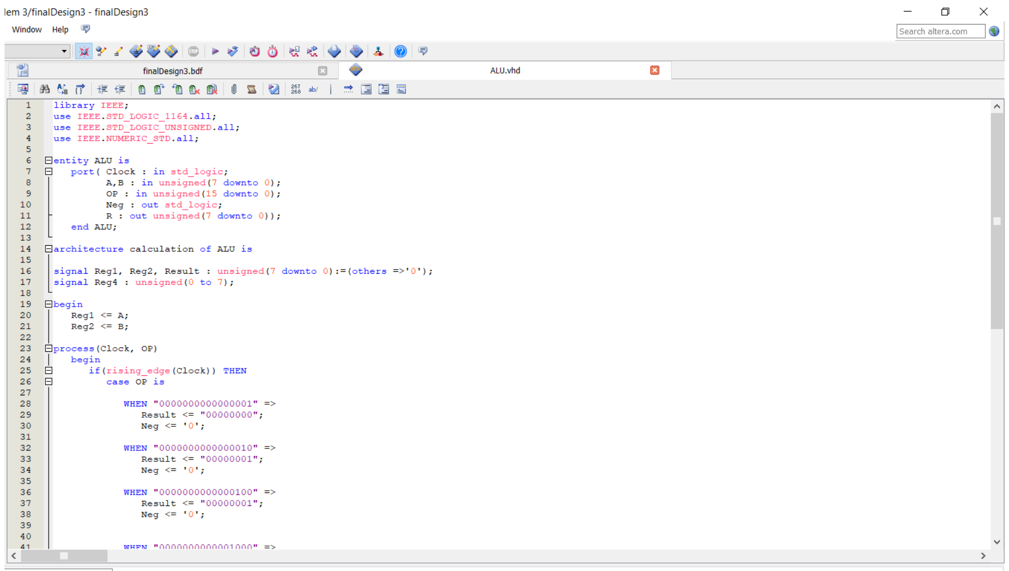
Block Diagram for ALU 3B:



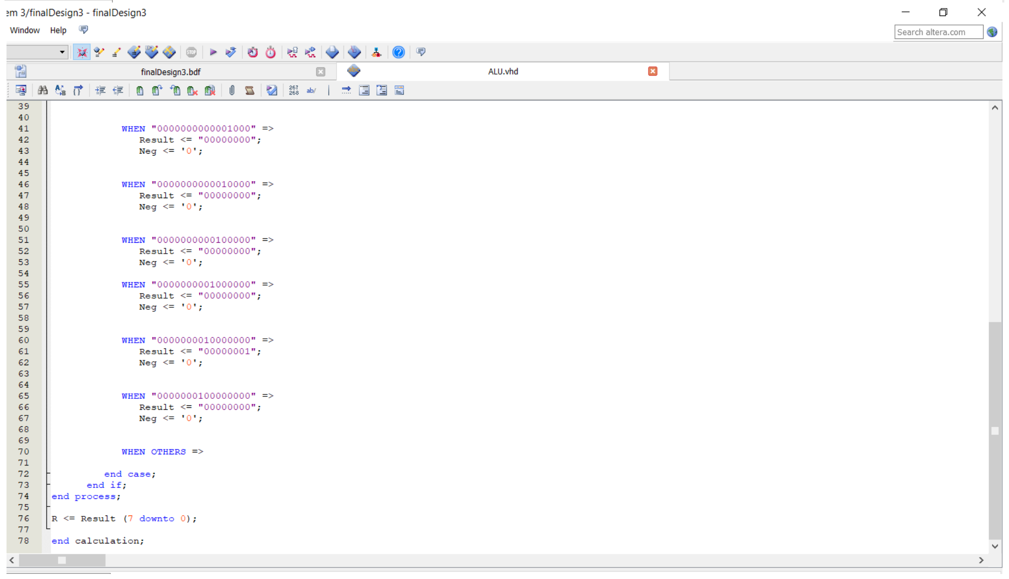
Calculations for ALU 3B Functions:

______**My student number :**	500957721
Even/odd digits (y/n):	nyynnnnyn
Desired output:	011000010

VHDL Codes of ALU 3B:

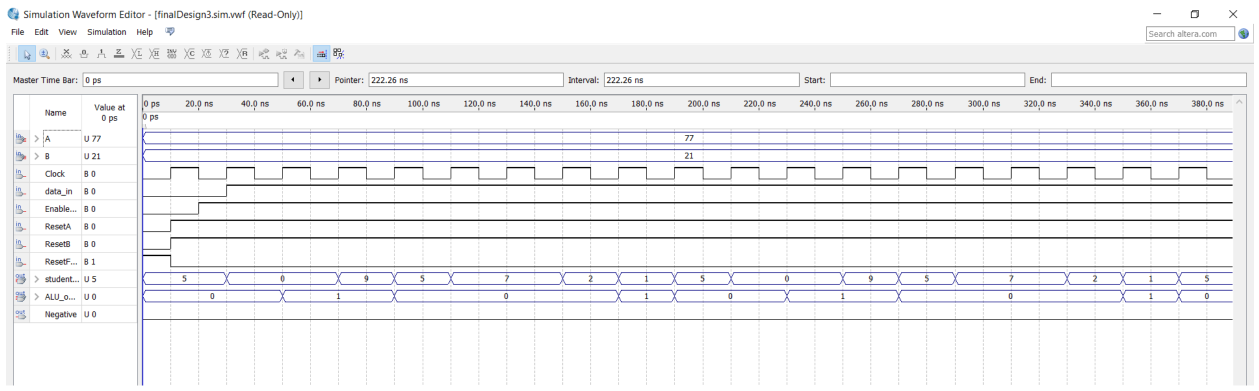


```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use IEEE.NUMERIC_STD.all;

entity ALU is
   port( Clock : in std_logic;
         A,B : in unsigned(7 downto 0);
         OP : in unsigned(15 downto 0);
         Neg : out std_logic;
         R : out unsigned(7 downto 0));
   end ALU;

architecture calculation of ALU 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 <= "00000000";
               Neg <= '0';

            WHEN "0000000000000010" =>
               Result <= "00000001";
               Neg <= '0';

            WHEN "0000000000000100" =>
               Result <= "00000001";
               Neg <= '0';


            WHEN "0000000000001000" =>
```



```vhdl


            WHEN "0000000000001000" =>
               Result <= "00000000";
               Neg <= '0';


            WHEN "0000000000010000" =>
               Result <= "00000000";
               Neg <= '0';


            WHEN "0000000000100000" =>
               Result <= "00000000";
               Neg <= '0';

            WHEN "0000000001000000" =>
               Result <= "00000000";
               Neg <= '0';


            WHEN "0000000010000000" =>
               Result <= "00000001";
               Neg <= '0';


            WHEN "0000000100000000" =>
               Result <= "00000000";
               Neg <= '0';


            WHEN OTHERS =>

         end case;
      end if;
end process;

R <= Result (7 downto 0);

end calculation;
```

Waveform for ALU Problem 3B:



Master Time Bar: 0 ps
Pointer: 222.26 ns
Interval: 222.26 ns
Start:
End:

```
A          U 77   77
B          U 21   21
Clock      B 0
data_in    B 0
Enable...  B 0
ResetA     B 0
ResetB     B 0
ResetF...  B 1
student... U 5    5 | 0 | 9 | 5 | 7 | 2 | 1 | 5 | 0 | 9 | 5 | 7 | 2 | 1 | 5
ALU_o...   U 0    0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0
Negative   U 0
```

Similar to my 2F waveform, for some reason my ALU_Output lags or clocks 20ns after my student number of when its supposed to and that's why the 0 in the beginning looks longer and throws the output off a bit, but I assure you the waveform is correct if the lag is disregarded.

Purpose of inputs & outputs of ALU 3B:

Inputs:

A, B : 8-bit inputs from Latches A and B

OP : 16-bit inputs from Moore fsm

Clock : Signals the clock to synchronize everything

Outputs:

ALU_output : (self-explanatory) but outputs the ALU functions in waveform format

Neg : Outputs a wave whose value is 1 if it negative number or remains 0 if positive

Conclusion

Therefore, this final lab of this course gives us students a solid foundation of the process of an ALU unit and how to design its function. The General Purpose Processor or GPU that we made consisted of 2 latches, a 4:16 decoder, a Moore FSM as well as an ALU core. After several days of coding VHDL, building block diagrams, and compiling waveforms, we students were able to apply all our knowledge and understanding into this final lab project for this course. Everything that we learned till now was incorporated. Whether it was learning NAND operand on the 3rd week of class, or it was learning about sequential circuits near the end of the course, whatever it may be, if the content was taught in this course, it was used/applied in this lab. The main takeaway from a lab like this was not to complain about how much work this lab was but it was to feel accomplished, as if I've achieved something. And truth be told, I actually had fun during this lab, not only was I having a lot of hands-on experience (not literally because of this pandemic) and I feel like this really tested my knowledge and understanding that I had up until now. In the end, during the entire duration of this project, the main thing each student learnt was the basic functionalities of each GPU and how it performs each given task.