TABLE OF CONTENTS:

INTRODUCTION:

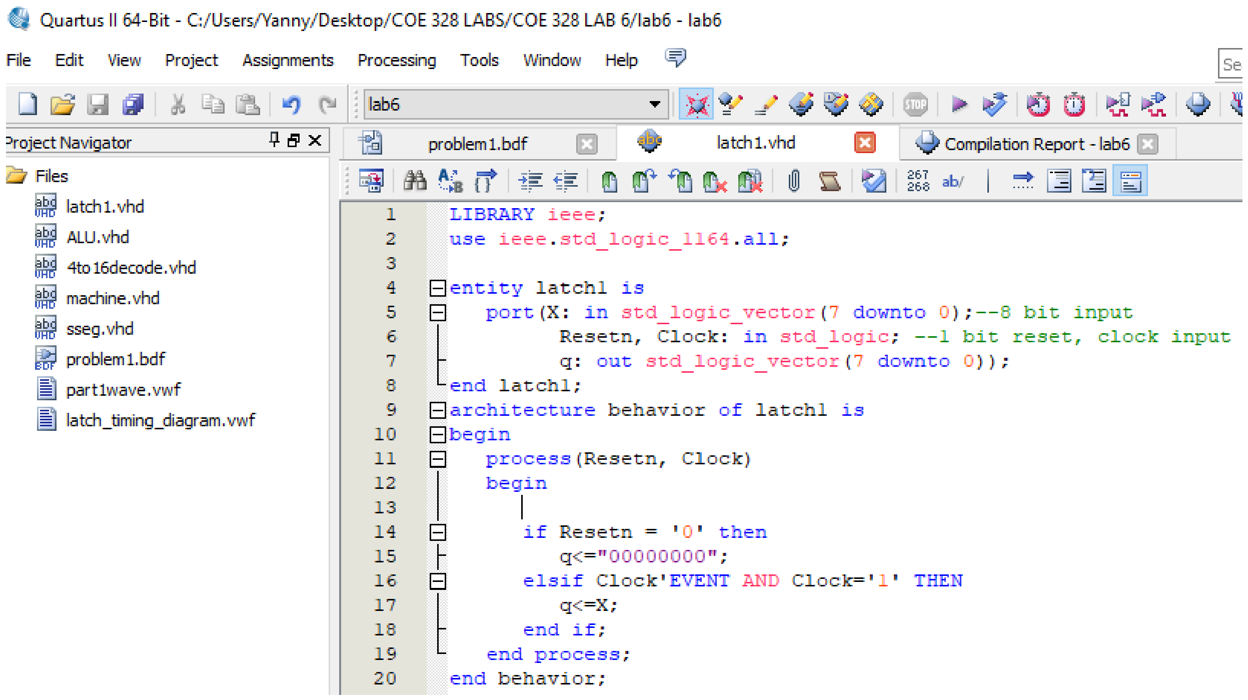
The main objective of this lab is to design and construct three general processing units (GPUs) using components from previous labs in unison with new knowledge of sequential circuits and arithmetic logic units. To construct the GPU, two latches, an ALU, a seven segment display converter and a control unit which is comprised of a 4:16 decoder and a finite state machine (FSM) will be used. Overall the processor should take two 8-bit inputs and perform an operation on them based on the microcode produced by the control unit. The results from the components will then be ran through a seven segment display converter and displayed.

COMPONENTS:

1. LATCHES

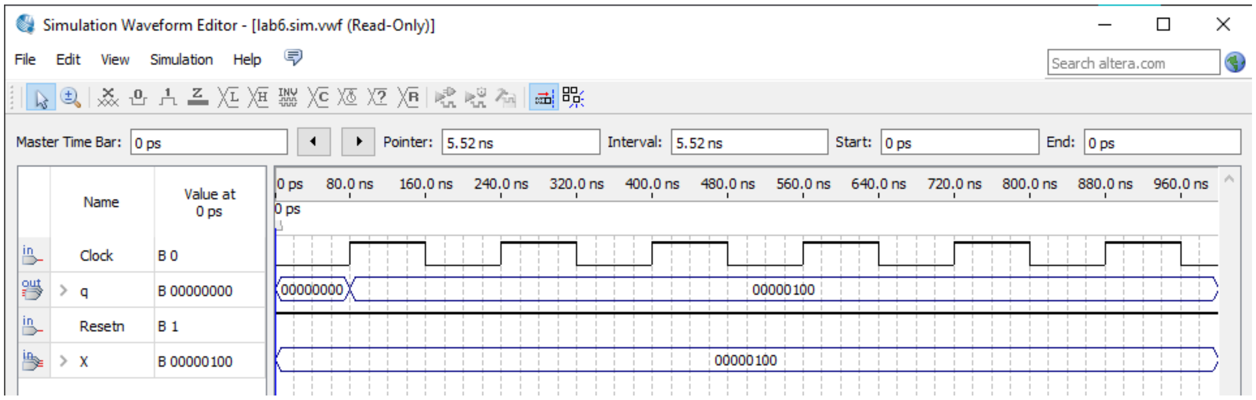
Two latches were used for the Arithmetic Logic Unit and were storage for the two inputs, A and B. Essentially, a Latch is a storage unit that holds required data using a signal. The signal in this case would be the clock input which is used to determine the value of Q alongside the input of X. The latches used in this case are both D flip-flops as they are controlled by the clock and the changes in state are at each rising edge of the clock. As shown in the truth table in *Table 1*, if the clock has a value of 0 and the, value of the output (q) would be 0 and when the clock has a value of 1, the output (q) would match the value of X.

Figure 1: VHDL Code for the Latch


Quartus II 64-Bit - C:/Users/Yanny/Desktop/COE 328 LABS/COE 328 LAB 6/lab6 - lab6
File Edit View Project Assignments Processing Tools Window Help
lab6
Project Navigator
problem1.bdf
latch1.vhd
Compilation Report - lab6
Files
latch1.vhd
ALU.vhd
4to16decode.vhd
machine.vhd
sseg.vhd
problem1.bdf
part1wave.vwf
latch_timing_diagram.vwf

```
LIBRARY ieee;
use ieee.std_logic_1164.all;

entity latch1 is
   port(X: in std_logic_vector(7 downto 0);--8 bit input
         Resetn, Clock: in std_logic; --1 bit reset, clock input
         q: out std_logic_vector(7 downto 0));
end latch1;
architecture behavior of latch1 is
begin
   process(Resetn, Clock)
   begin

      if Resetn = '0' then
         q<="00000000";
      elsif Clock'EVENT AND Clock='1' THEN
         q<=X;
      end if;
   end process;
end behavior;
```

Figure 2: The compiled timing diagram for the Latch

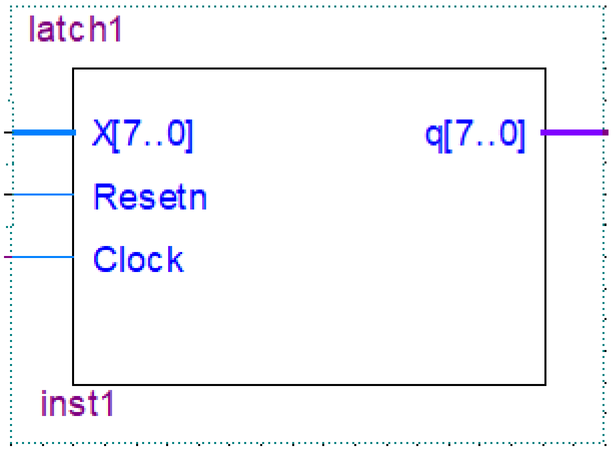

Simulation Waveform Editor - [lab6.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer: 5.52 ns
Interval: 5.52 ns
Start: 0 ps
End: 0 ps
0 ps 80.0 ns 160.0 ns 240.0 ns 320.0 ns 400.0 ns 480.0 ns 560.0 ns 640.0 ns 720.0 ns 800.0 ns 880.0 ns 960.0 ns
00000000
00000100
00000100

Name	Value at 0 ps
Clock	B 0
q	B 00000000
Resetn	B 1
X	B 00000100

Table 1: The truth table for the Latch

Clock	X	q(t+1)
0	X [DONT CARE]	q(t)
1	0	0
1	1	1

Component 1: The circuit diagram for the latch

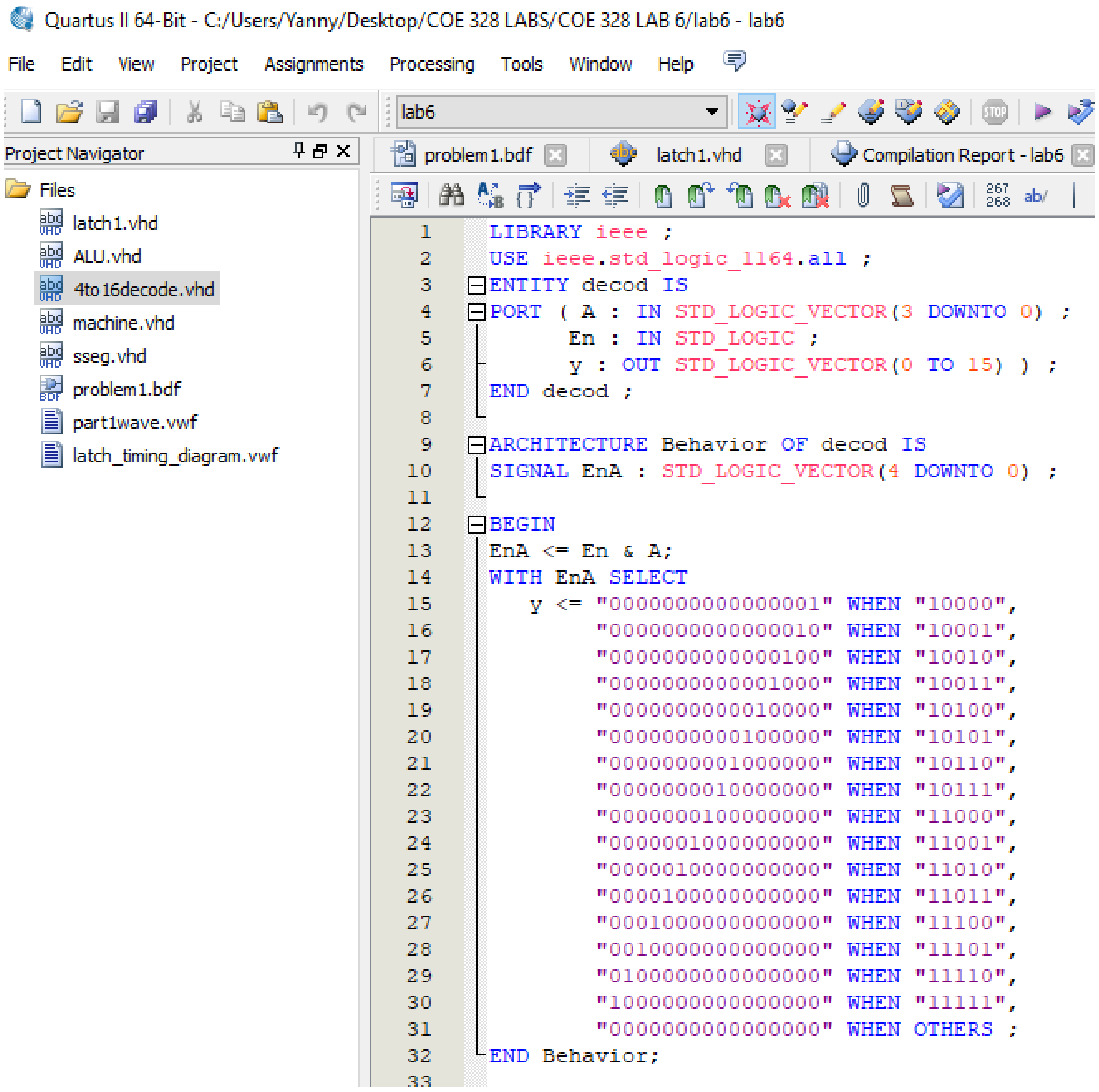


By comparing the truth table with the compiled waveform it can be seen that the latch was successfully constructed as when the clock is 0, the output is also 0. When the Clock is 1, the output matches the value of the input X which corresponds to the truth table which means that the latch is operational and the two needed latches can be implemented.

2. 4:16 DECODER

The 4 to 16 decoder takes the 4-bit input from the finite state machine and outputs a unique 16-bit output. This output is then used as input for the main ALU.

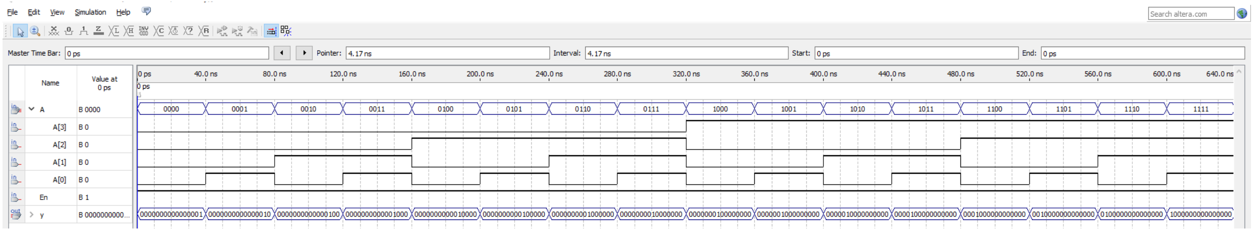
Figure 3: VHDL Code for the 4:16 Decoder

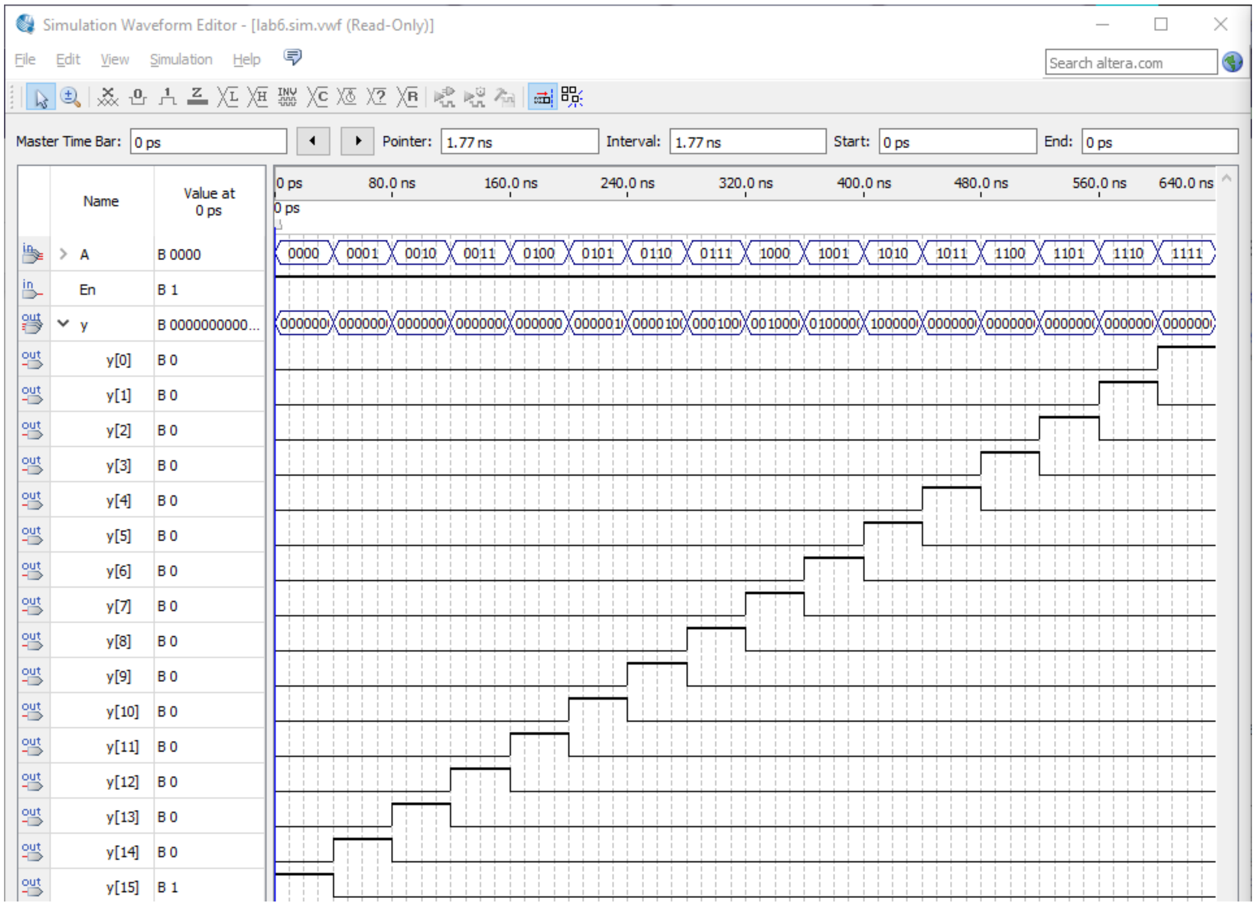


Quartus II 64-Bit - C:/Users/Yanny/Desktop/COE 328 LABS/COE 328 LAB 6/lab6 - lab6
File Edit View Project Assignments Processing Tools Window Help
lab6
Project Navigator
problem1.bdf
latch1.vhd
Compilation Report - lab6
Files
latch1.vhd
ALU.vhd
4to16decode.vhd
machine.vhd
sseg.vhd
problem1.bdf
part1wave.vwf
latch_timing_diagram.vwf

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY decod IS
PORT ( A : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
       En : IN STD_LOGIC ;
       y : OUT STD_LOGIC_VECTOR(0 TO 15) ) ;
END decod ;

ARCHITECTURE Behavior OF decod IS
SIGNAL EnA : STD_LOGIC_VECTOR(4 DOWNTO 0) ;

BEGIN
EnA <= En & A;
WITH EnA 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 Behavior;
```

Figure 4: Compiled Waveform for the 4:16 Decoder





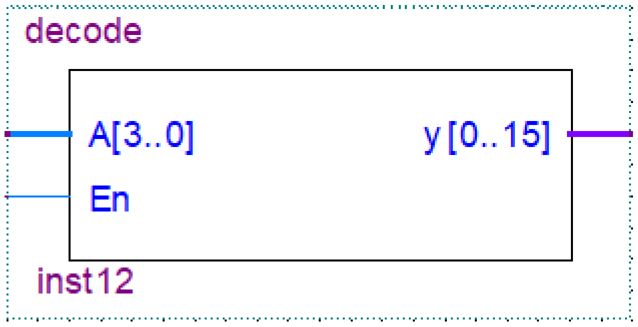
Note: Two Screenshots were provided for a better view of the waveforms

Table 2: The Truth Table for the 4:16 Decoder

A	**y**
0000	0000000000000001
0001	0000000000000010
0010	0000000000000100
0011	0000000000001000

0100	0000000000010000
0101	0000000000100000
0110	0000000001000000
0111	0000000010000000
1000	0000000100000000
1001	0000001000000000
1010	0000010000000000
1011	0000100000000000
1100	0001000000000000
1101	0010000000000000
1110	0100000000000000
1111	1000000000000000

Component 2: The circuit diagram for the decoder

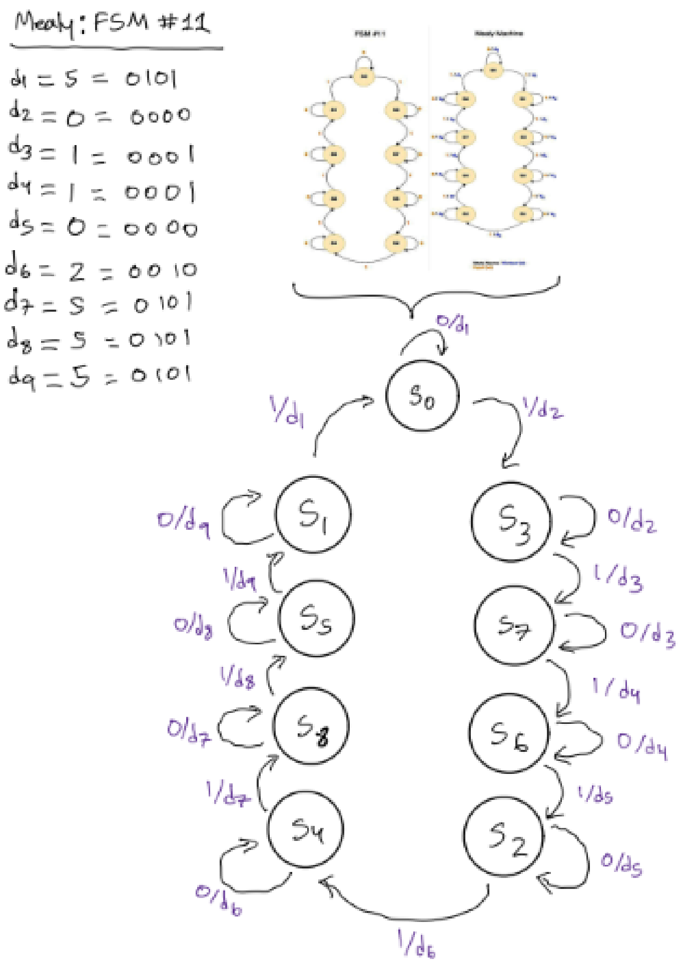


For the waveform, the enable is assigned a value of 1 for the entirety of the diagram while the values for each input, *A*, were given a 4-bit binary number starting at 0 and going up to 15. The desired outcome was achieved as the timing diagram matches up with the results in the truth table which means that the 4:16 decoder was successfully made using VHDL.

3. STUDENT-UNIQUE FSM (FINITE STATE MACHINE)

The Finite State Machine which was used to construct the ALU was taken from the previous lab where a Mealy machine was constructed. Essentially, the machine cycles through states from 0 to 8, and the next state is determined based on the input data and the design of the machine. As seen in *Figure 5*, if the machine were to start off at State 0 and the input data was 1, the value assigned to d2 would be outputted and the next state would be State 3. This will happen until the final state is reached where it will transition back to the zeroth state and start over. The values in the machine correspond to a single digit of a student number. As the machine cycles through each state, the corresponding value will be outputted which would just be a single digit of the student number in binary and the current state will be sent to the 4:16 decoder which will essentially be the microcode for the ALU. Overall, the function of the finite state machine will be to cycle through the states and values creating a unique pattern for the controller sequence.

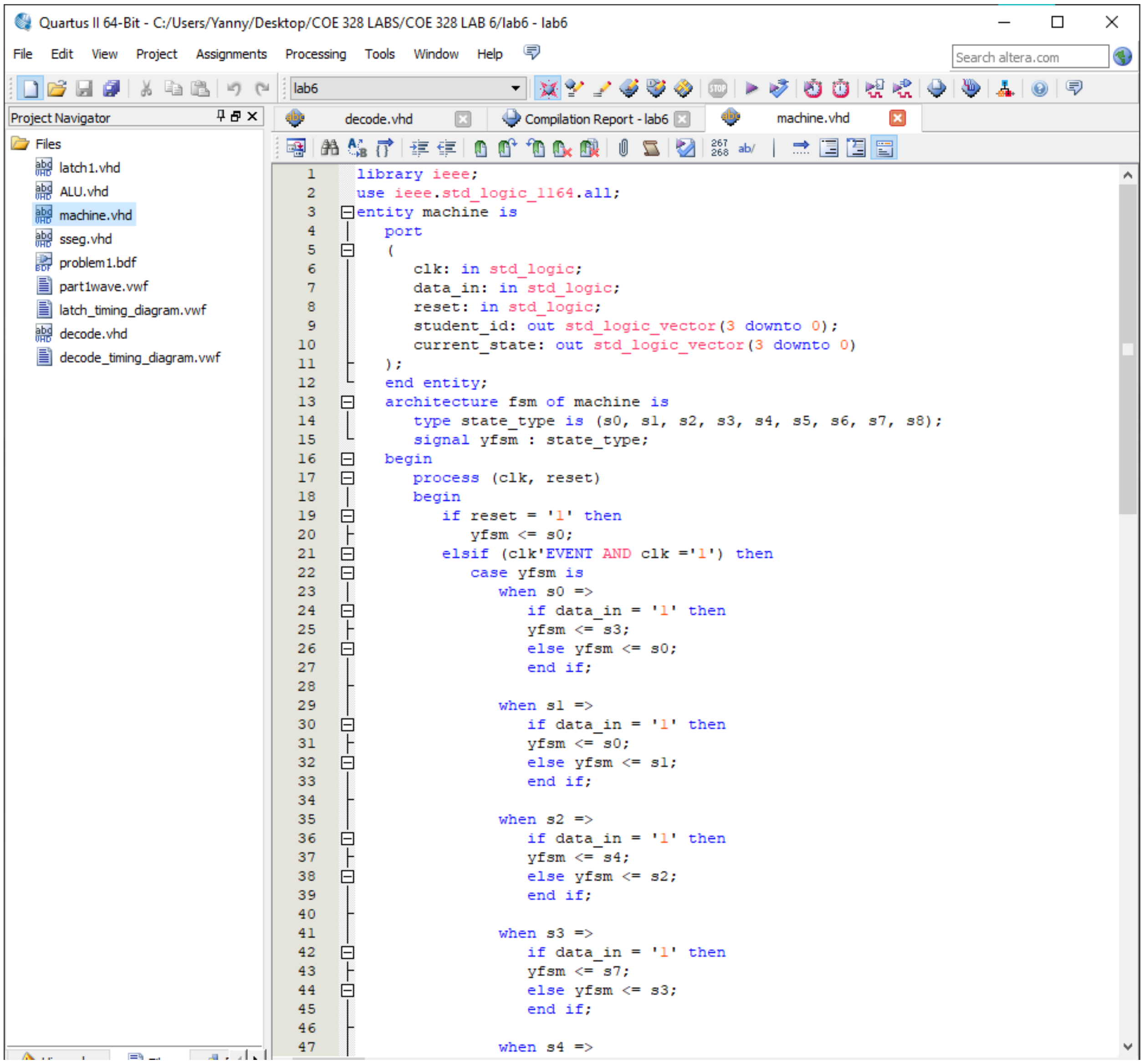
Figure 5: Mealy Machine Design demonstrating state changes based on input data



Mealy: FSM #11

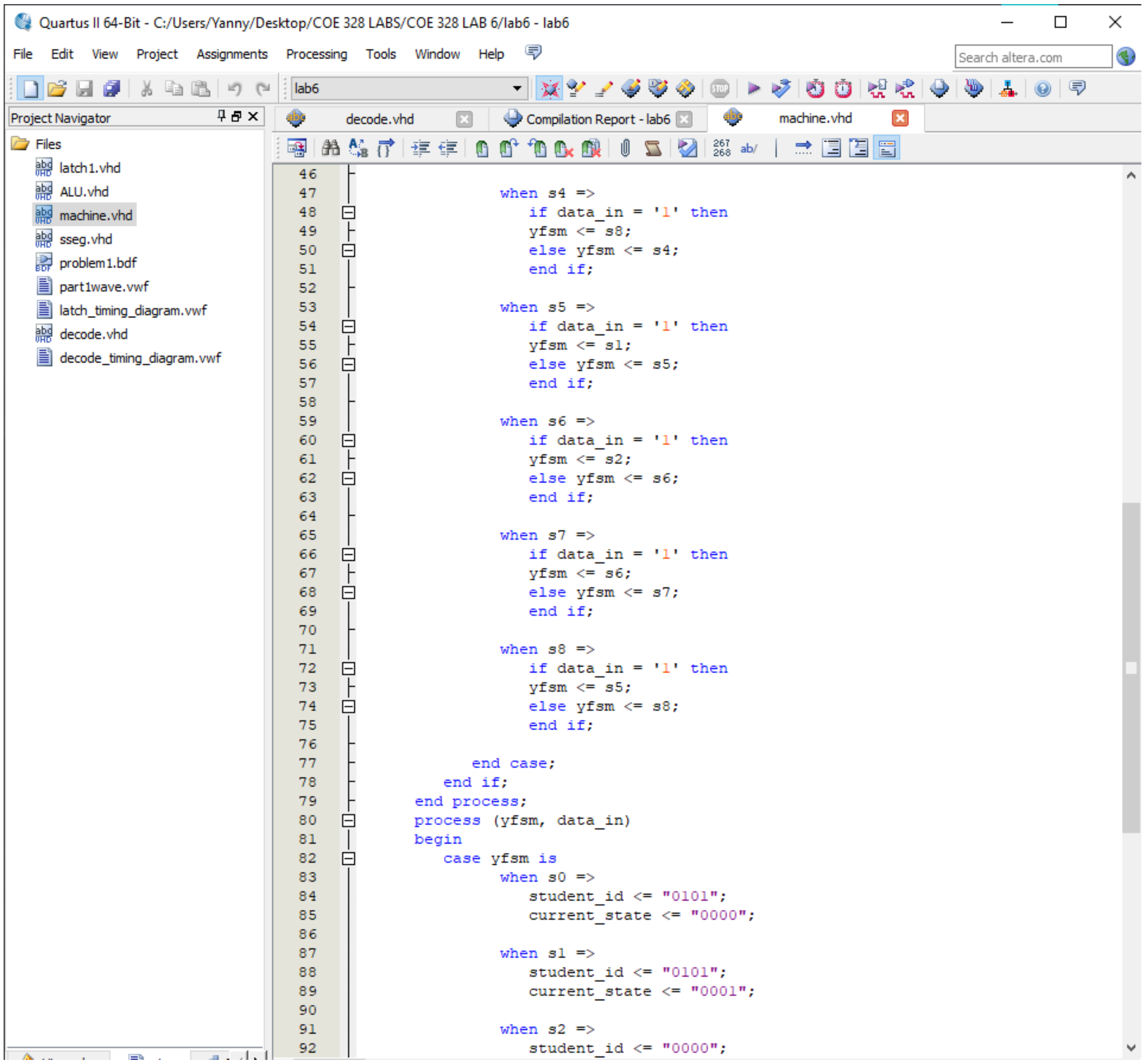
$d_1 = 5 = 0101$
$d_2 = 0 = 0000$
$d_3 = 1 = 0001$
$d_4 = 1 = 0001$
$d_5 = 0 = 0000$
$d_6 = 2 = 0010$
$d_7 = 5 = 0101$
$d_8 = 5 = 0101$
$d_9 = 5 = 0101$

Figure 6: The VHDL Code for the Mealy Machine

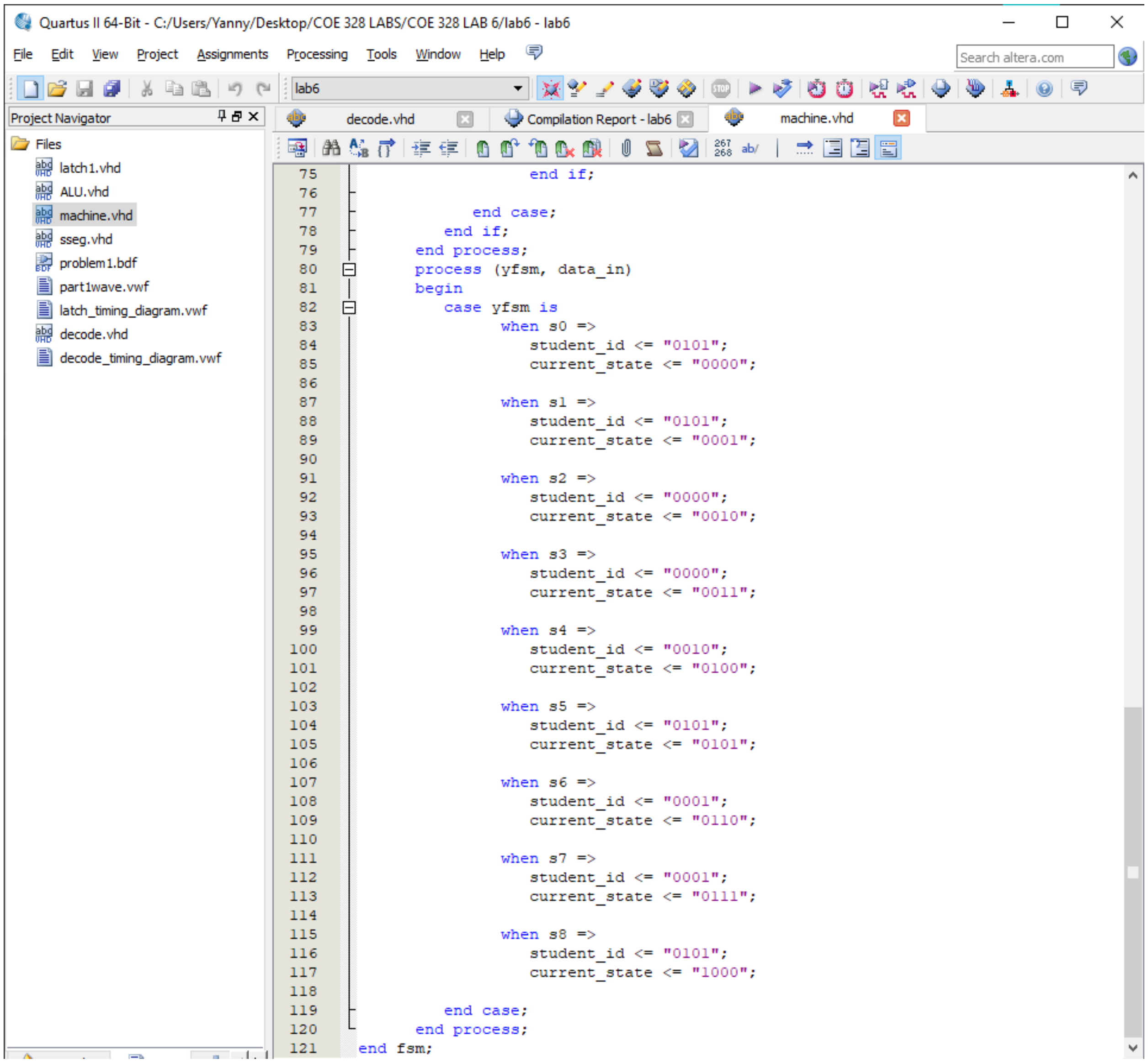


Quartus II 64-Bit - C:/Users/Yanny/Desktop/COE 328 LABS/COE 328 LAB 6/lab6 - lab6

```
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 = '1' then
                        yfsm <= s3;
                        else yfsm <= s0;
                        end if;

                    when s1 =>
                        if data_in = '1' then
                        yfsm <= s0;
                        else yfsm <= s1;
                        end if;

                    when s2 =>
                        if data_in = '1' then
                        yfsm <= s4;
                        else yfsm <= s2;
                        end if;

                    when s3 =>
                        if data_in = '1' then
                        yfsm <= s7;
                        else yfsm <= s3;
                        end if;

                    when s4 =>
```


Quartus II 64-Bit - C:/Users/Yanny/Desktop/COE 328 LABS/COE 328 LAB 6/lab6 - lab6
File Edit View Project Assignments Processing Tools Window Help
Search altera.com
lab6
Project Navigator
decode.vhd
Compilation Report - lab6
machine.vhd
Files
latch1.vhd
ALU.vhd
machine.vhd
sseg.vhd
problem1.bdf
part1wave.vwf
latch_timing_diagram.vwf
decode.vhd
decode_timing_diagram.vwf

```
                when s4 =>
                   if data_in = '1' then
                   yfsm <= s8;
                   else yfsm <= s4;
                   end if;

                when s5 =>
                   if data_in = '1' then
                   yfsm <= s1;
                   else yfsm <= s5;
                   end if;

                when s6 =>
                   if data_in = '1' then
                   yfsm <= s2;
                   else yfsm <= s6;
                   end if;

                when s7 =>
                   if data_in = '1' then
                   yfsm <= s6;
                   else yfsm <= s7;
                   end if;

                when s8 =>
                   if data_in = '1' then
                   yfsm <= s5;
                   else yfsm <= s8;
                   end if;

             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 <= "0101";
                   current_state <= "0001";

                when s2 =>
                   student_id <= "0000";
```


Quartus II 64-Bit - C:/Users/Yanny/Desktop/COE 328 LABS/COE 328 LAB 6/lab6 - lab6
File Edit View Project Assignments Processing Tools Window Help
lab6
Project Navigator
Files
latch1.vhd
ALU.vhd
machine.vhd
sseg.vhd
problem1.bdf
part1wave.vwf
latch_timing_diagram.vwf
decode.vhd
decode_timing_diagram.vwf
decode.vhd
Compilation Report - lab6
machine.vhd

```
                  end if;

            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 <= "0101";
                  current_state <= "0001";

               when s2 =>
                  student_id <= "0000";
                  current_state <= "0010";

               when s3 =>
                  student_id <= "0000";
                  current_state <= "0011";

               when s4 =>
                  student_id <= "0010";
                  current_state <= "0100";

               when s5 =>
                  student_id <= "0101";
                  current_state <= "0101";

               when s6 =>
                  student_id <= "0001";
                  current_state <= "0110";

               when s7 =>
                  student_id <= "0001";
                  current_state <= "0111";

               when s8 =>
                  student_id <= "0101";
                  current_state <= "1000";

         end case;
      end process;
end fsm;
```

Figure 7: Compiled Waveform for the Mealy Machine

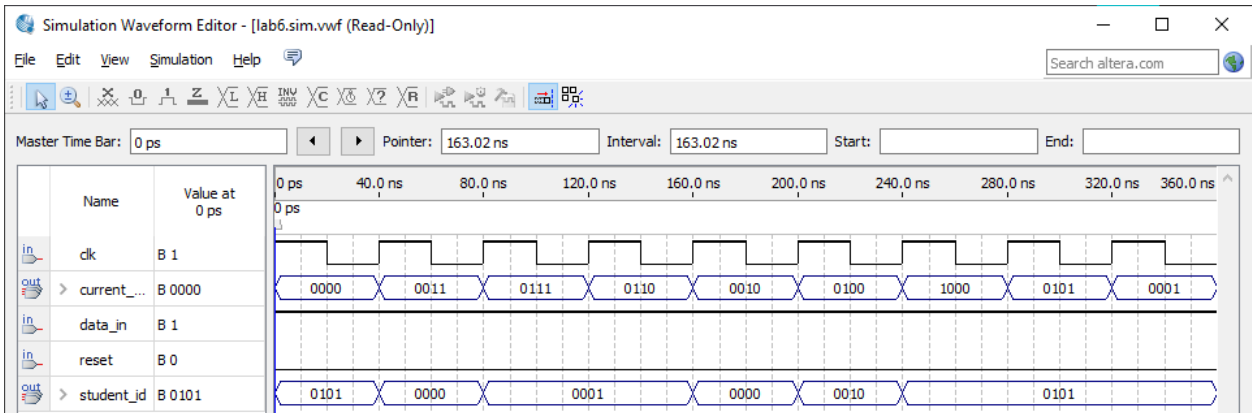
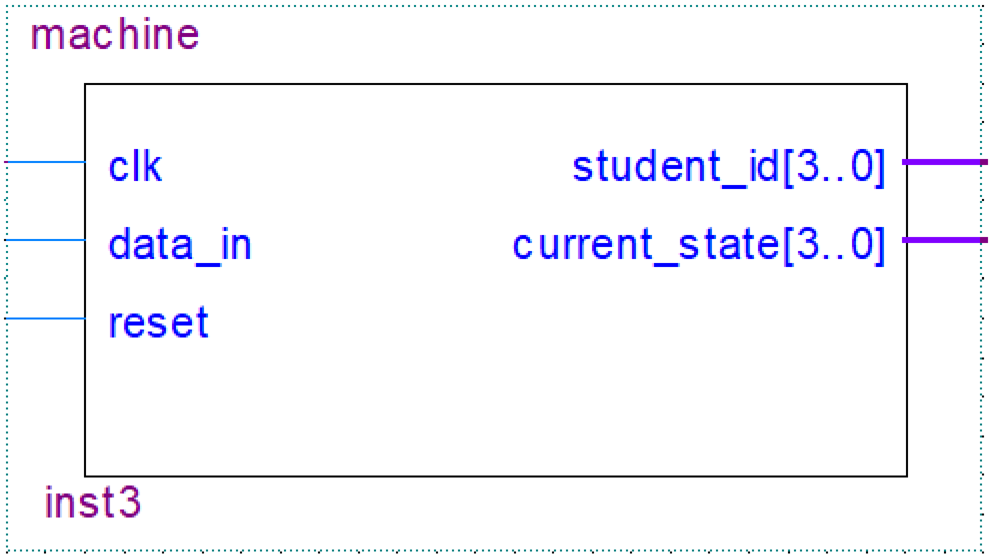


Table 3: The Truth Table for the FSM

	INPUT		OUTPUT	
Current State	w = 0	w = 1	w = 0	w = 1
0000	0000	0011	0101	0000
0001	0001	0000	0101	0101
0010	0010	0100	0000	0010
0011	0011	0111	0000	0001
0100	0100	1000	0010	0101
0101	0101	0001	0101	0101
0110	0110	0010	0001	0000
0111	0111	0110	0001	0001
1000	1000	0101	0101	0101

Component 3: The circuit diagram for the FSM

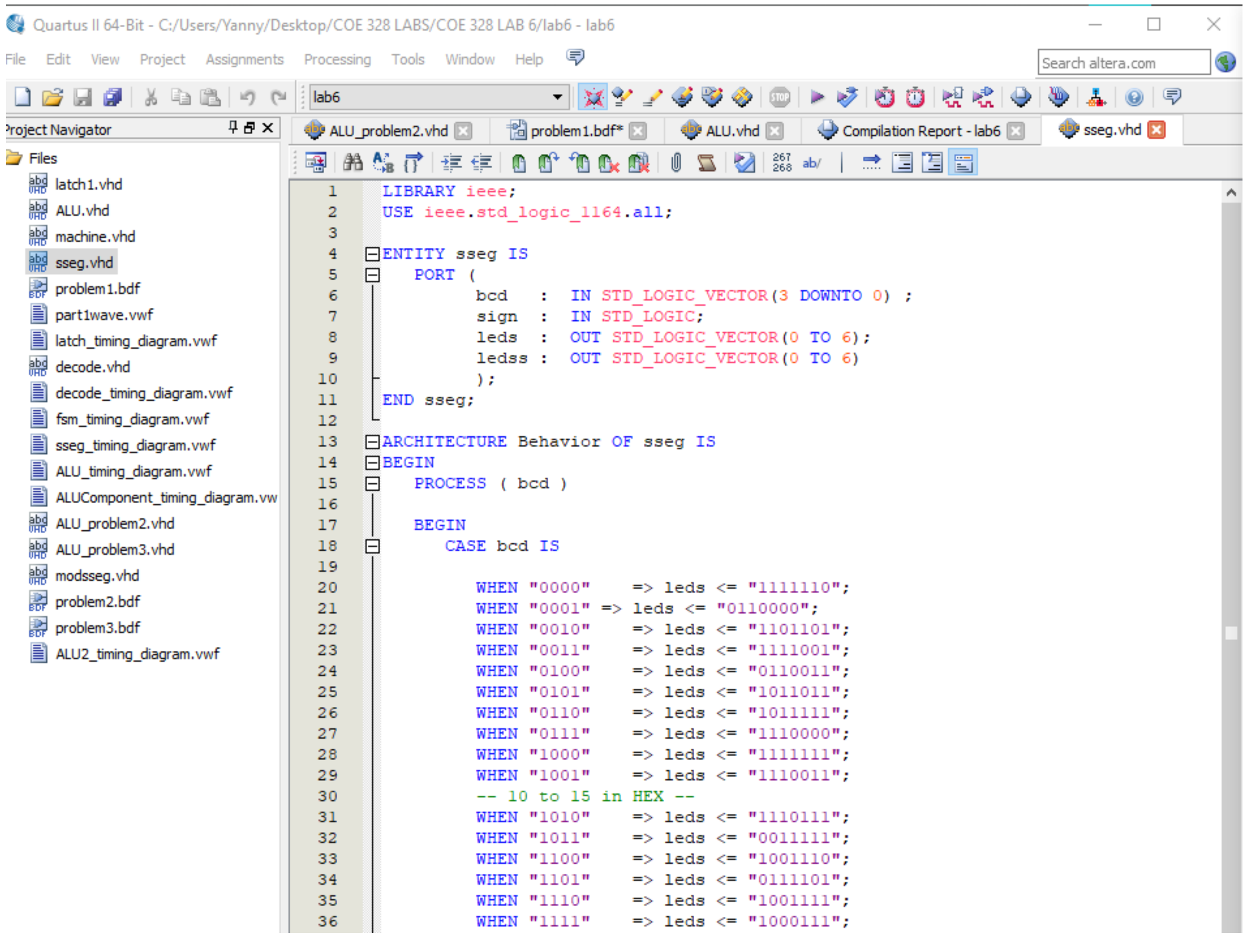


Upon comparison it can be seen that the truth table and the compiled waveform match and the desired outcomes are produced. Since the input data was given a value of 1 for the entirety of the waveform, the inputs, and outputs where w = 1 will be used. Starting with the states, it can be seen that with a data-in of 1, the states cycle from 1 to 3 to 7 and so on which corresponds to the states shown in the truth table. As for the values themselves, the correct output was produced since the desired student number was outputted (501102555) in binary. This indicates that the FSM was successful and was constructed correctly.

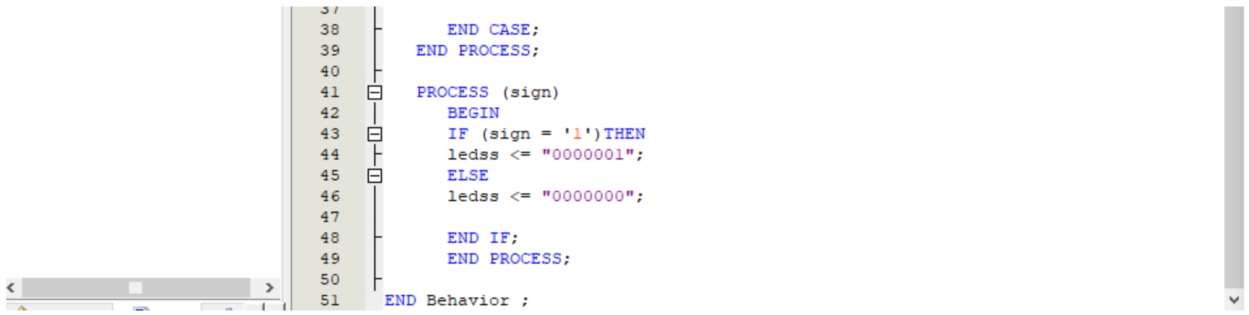
4. 7-SEGMENT DISPLAY

The seven-segment display was re-used from previous labs. Essentially the code takes a 4-bit input and outputs a 7-bit output which can be used to display the integer value of the input on a seven-segment display. The VHDL code uses "when" statements to check the input and output the set value in the code as seen in *Figure 8*. In the lab, multiple seven-segment displays were used to display the student id, FSM values, and even the output from the ALU itself.

Figure 8: The VHDL Code for the seven-segment display

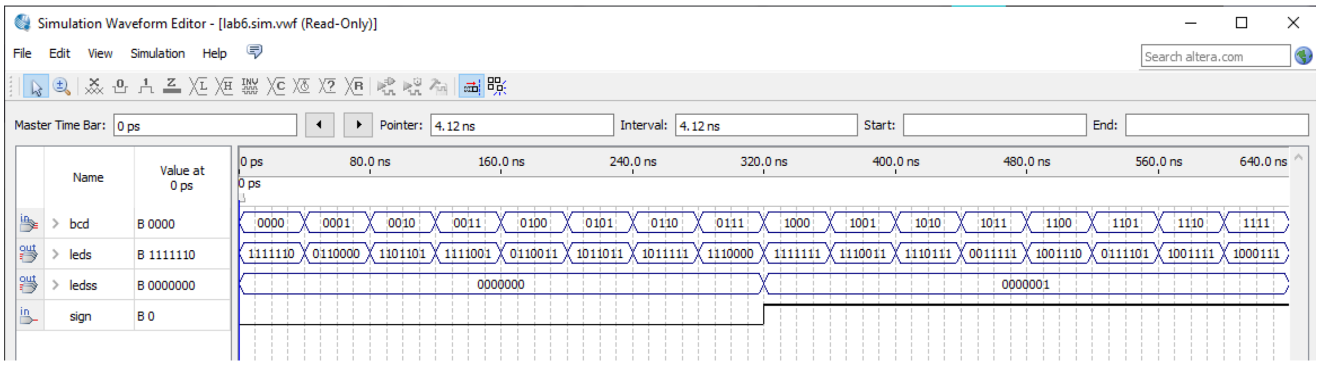


```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

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

ARCHITECTURE Behavior OF sseg IS
BEGIN
    PROCESS ( bcd )

    BEGIN
        CASE bcd IS

            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 <= "1110011";
            -- 10 to 15 in HEX --
            WHEN "1010"    => leds <= "1110111";
            WHEN "1011"    => leds <= "0011111";
            WHEN "1100"    => leds <= "1001110";
            WHEN "1101"    => leds <= "0111101";
            WHEN "1110"    => leds <= "1001111";
            WHEN "1111"    => leds <= "1000111";
```



```
      END CASE;
   END PROCESS;

   PROCESS (sign)
      BEGIN
      IF (sign = '1')THEN
      ledss <= "0000001";
      ELSE
      ledss <= "0000000";

      END IF;
      END PROCESS;

END Behavior ;
```

Figure 9: Compiled Waveform for the seven-segment display



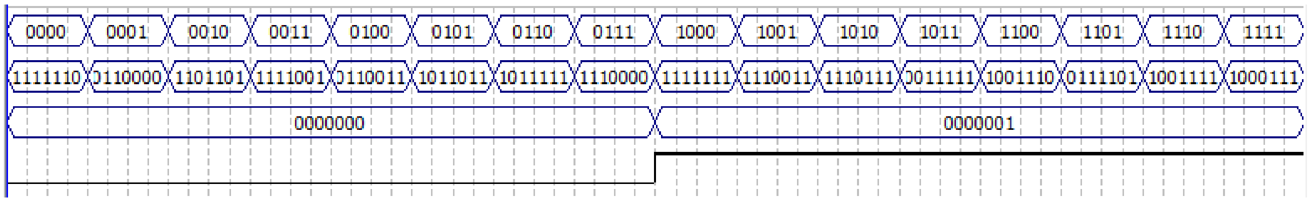


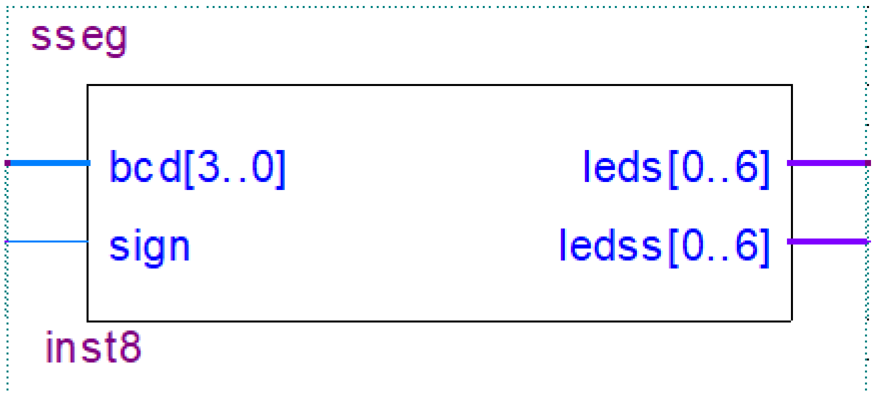
Table 4: The Truth Table for the seven-segment display

INPUT (bcd)	OUTPUT (leds)
0000	1111110
0001	0110000
0010	1101101
0011	1111001
0100	0110011
0101	1011011
0110	1011111
0111	1110000

1000	1111111
1001	1110011
1010	1110111 (A)
1011	0011111 (B)
1100	1001110 (C)
1101	0111101 (D)
1110	1001111 (E)
1111	1000111 (F)

INPUT (sign)	OUTPUT (ledss)
0	0000000
1	0000001

Component 4: The circuit diagram for the sseg



Once again, comparing the truth table with the waveform reveals that the sseg was correctly constructed and can be used for the final ALU design. As seen by the waveform, each input ranging from 0 to 15 was correctly converted to its corresponding value suitable for a seven-segment display. When the “sign” input is changed to a value of 1, it can be seen that the value of ledss also changes to a value of 0000001 which would be a negative sign if shown on a seven-segment display. Overall it can be concluded that the seven-segment display works as expected and can be used towards the final design of the ALU.

5. ALU (ARITHMETIC LOGICAL UNIT) - PROBLEM 1

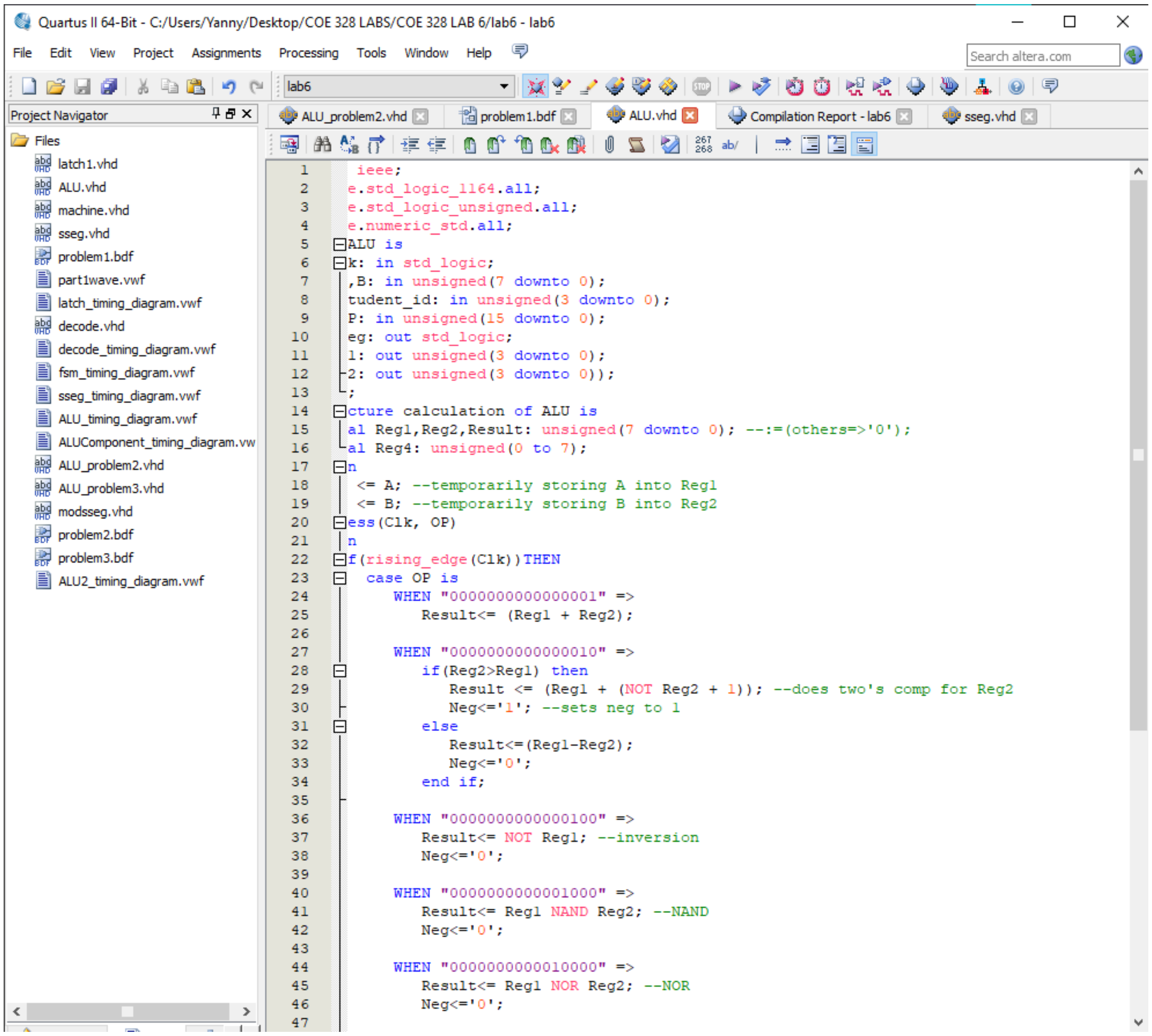
The ALU is the logic component of the processor as it's the unit that does the majority of operations in the processor. Based on the state of the FSM, the ALU does an operation. The output from FSM is sent to the 4:16 decoder which takes the 4-bit input which represents the state number in binary and decodes it to a 16-bit number in binary which is then used as the microcode for the processor. The microcode essentially tells the ALU what operation to execute. In this processor, there were a total of 9 microcodes which means the ALU has a total of 9 operations it can conduct based on the microcode that is inputted. Overall there are 5 inputs that were fed into the ALU, the clock, variables A and B, the student id, and the microcode (OP). In this part of the lab, the student id is not used in any logical calculations so the input on the ALU for student id is set to ground. The output "Neg", is used for the difference operation in the ALU as if B is greater than A, "Neg" is set to 1 to indicate that the difference between A and B is negative which will be shown as a negative sign after processed by the sseg block. The clock is a shared input between the ALU, the latches, and the FSM. The clock input was set to alternate between 1 and 0 which allows the ALU to periodically check the microcode input and execute the required operation. To explain the general operation of this GPU, variables A and B are inputted into the two latches as 8-bit binary numbers along with the clock. These variables are then sent to the ALU where they will be used as the values for each operation. The control unit which consists of the FSM and the decoder will be fed a reset, clock, and data_in input which will cause the FSM to produce outputs for the student id and the current state. The student id is then sent to a seven-segment display block which will take the 4-bit student id and convert it so that it's suitable for a seven-segment display. The current state produced by the FSM will then be sent to the decoder which will output a 16-bit value that is then fed into the ALU as the microcode. With the inputs from the latches and the control unit, the ALU checks the microcode and performs the required operations with variables A and B, outputting 2 4-bit values which are then fed into two seven-segment displays. As seen in *Table 5*, the microcodes along with their respective operations are listed and the block diagram for the GPU can be found in *Figure 10*.

Table 5: ALU operations with corresponding microcode for problem 1

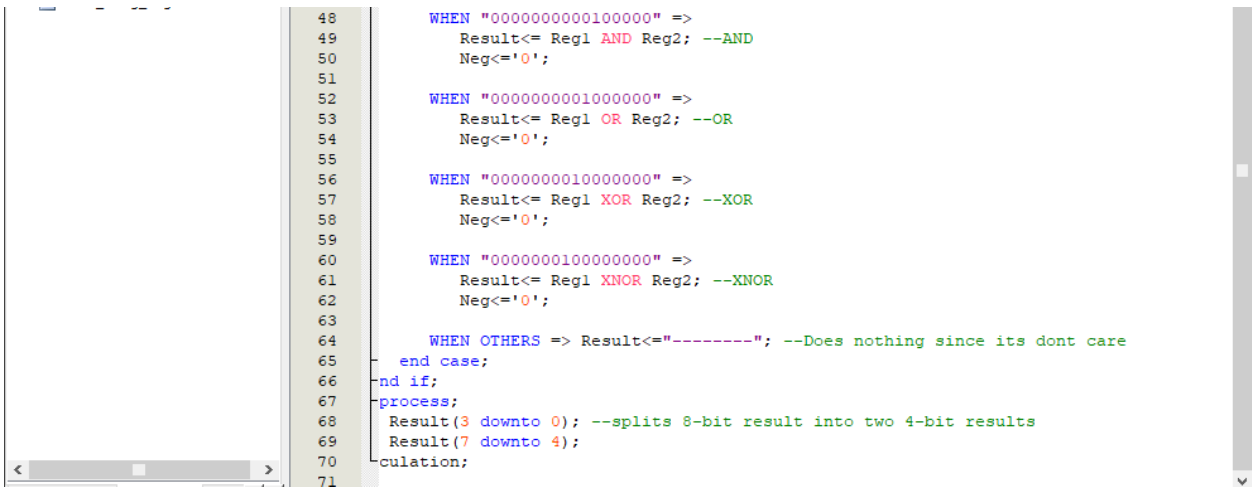
MICROCODE	BOOLEAN OPERATION
0000000000000001	sum(A, B)
0000000000000010	diff(A, B)
0000000000000100	$\bar{A}$
0000000000001000	$\overline{A \bullet B}$
0000000000010000	$\overline{A + B}$

0000000000100000	$A \bullet B$
0000000001000000	$A \oplus B$
0000000010000000	$A + B$
0000000100000000	$\overline{A \oplus B}$

Figure 10: VHDL Code for ALU (For Problem 1)



```
 ieee;
e.std_logic_1164.all;
e.std_logic_unsigned.all;
e.numeric_std.all;
ALU is
k: in std_logic;
,B: in unsigned(7 downto 0);
tudent_id: in unsigned(3 downto 0);
P: in unsigned(15 downto 0);
eg: out std_logic;
1: out unsigned(3 downto 0);
2: out unsigned(3 downto 0));
;
cture calculation of ALU is
al Reg1,Reg2,Result: unsigned(7 downto 0); --:=(others=>'0');
al Reg4: unsigned(0 to 7);
n
 <= A; --temporarily storing A into Reg1
 <= B; --temporarily storing B into Reg2
ess(Clk, OP)
n
f(rising_edge(Clk))THEN
  case OP is
     WHEN "0000000000000001" =>
        Result<= (Reg1 + Reg2);

     WHEN "0000000000000010" =>
        if(Reg2>Reg1) then
           Result <= (Reg1 + (NOT Reg2 + 1)); --does two's comp for Reg2
           Neg<='1'; --sets neg to 1
        else
           Result<=(Reg1-Reg2);
           Neg<='0';
        end if;

     WHEN "0000000000000100" =>
        Result<= NOT Reg1; --inversion
        Neg<='0';

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

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

```



```
        WHEN "0000000000100000" =>
           Result<= Reg1 AND Reg2; --AND
           Neg<='0';

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

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

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

        WHEN OTHERS => Result<="--------"; --Does nothing since its dont care
     end case;
nd if;
process;
 Result(3 downto 0); --splits 8-bit result into two 4-bit results
 Result(7 downto 4);
culation;
```

Figure 11: Block Diagram for the final GPU for problem 1

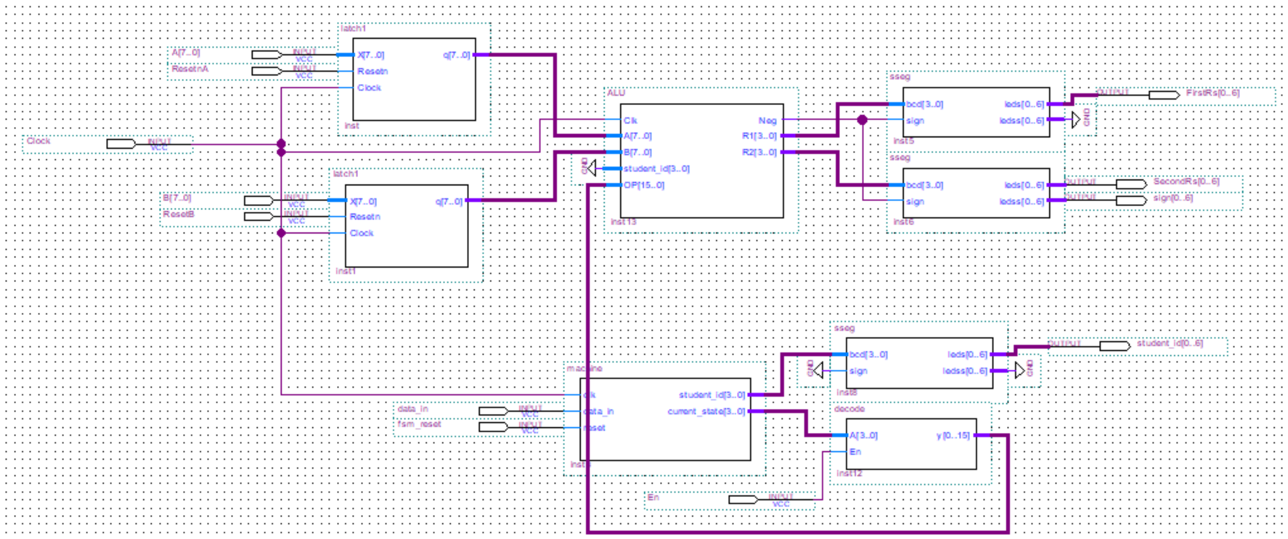
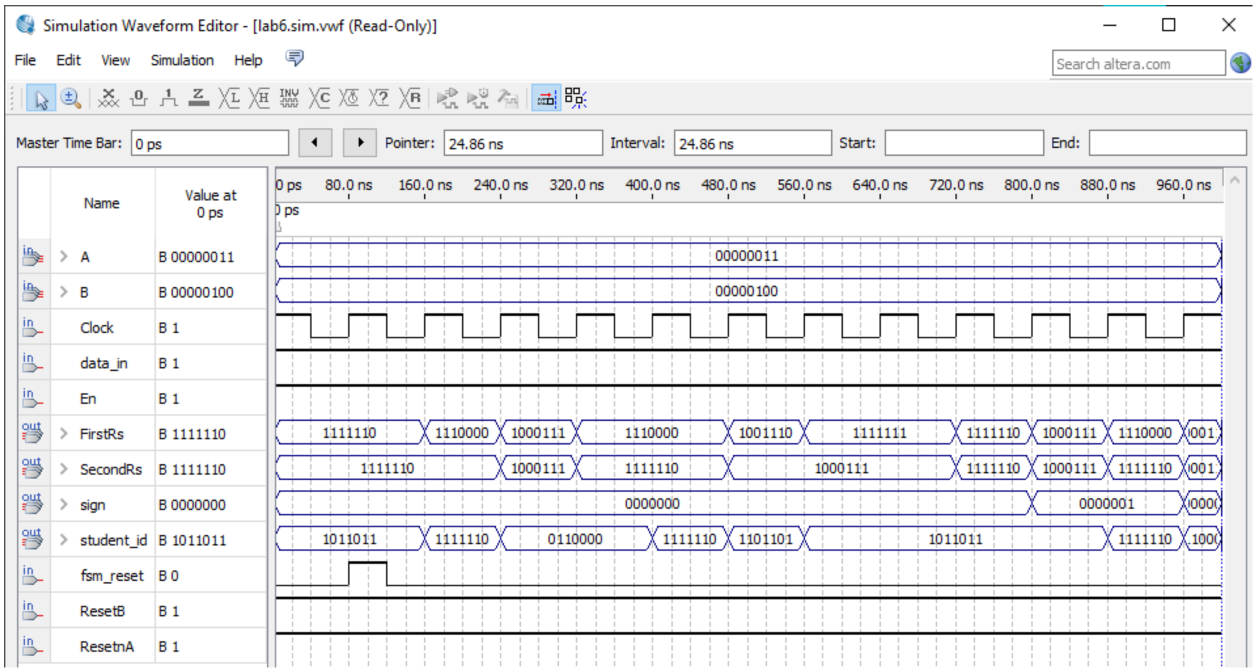


Figure 12: Compiled Waveform for the finished GPU



Name	Value at 0 ps
A	B 00000011
B	B 00000100
Clock	B 1
data_in	B 1
En	B 1
FirstRs	B 1111110
SecondRs	B 1111110
sign	B 0000000
student_id	B 1011011
fsm_reset	B 0
ResetB	B 1
ResetnA	B 1

6. ALU (ARITHMETIC LOGICAL UNIT) - PROBLEM 2

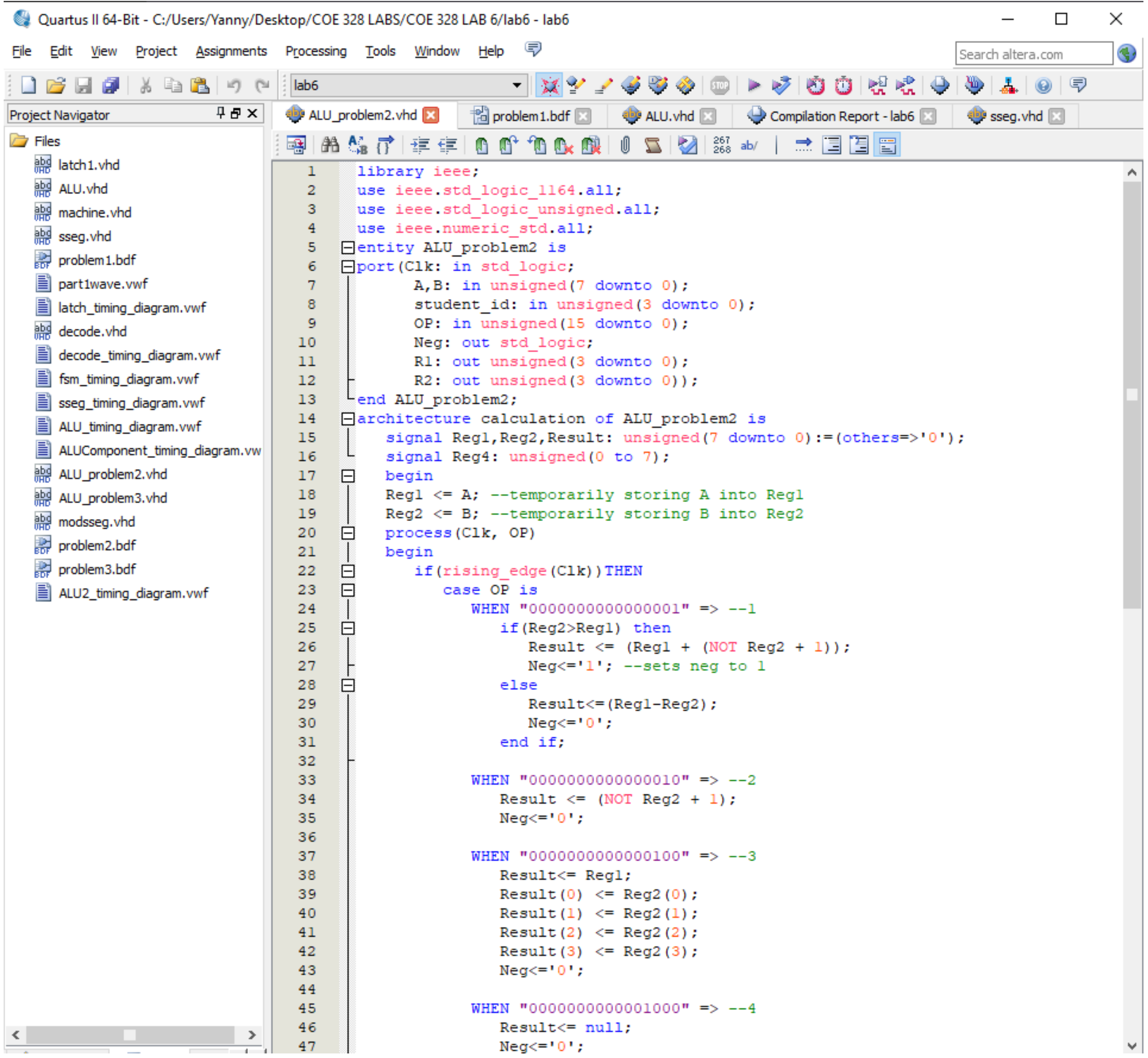
For the second part of the lab, the main function of the general processing unit was the same except the operations performed by the ALU were different. For the set of operations to be performed problem set c) was used as seen in *Table 6*. In terms of the input and output, the variables and their function were the exact same as the first part of the lab, although the values that were outputted were different as the ALU performed a different set of operations. Just as the previous part, the ALU does not use the student id in any logical calculations rather, it is simply fed into a seven segment display directly from the FSM and outputted. Below, the VHDL code, the finished block diagram and its respective waveform are shown for further reference.

Table 6: ALU operations with corresponding microcode for problem 2

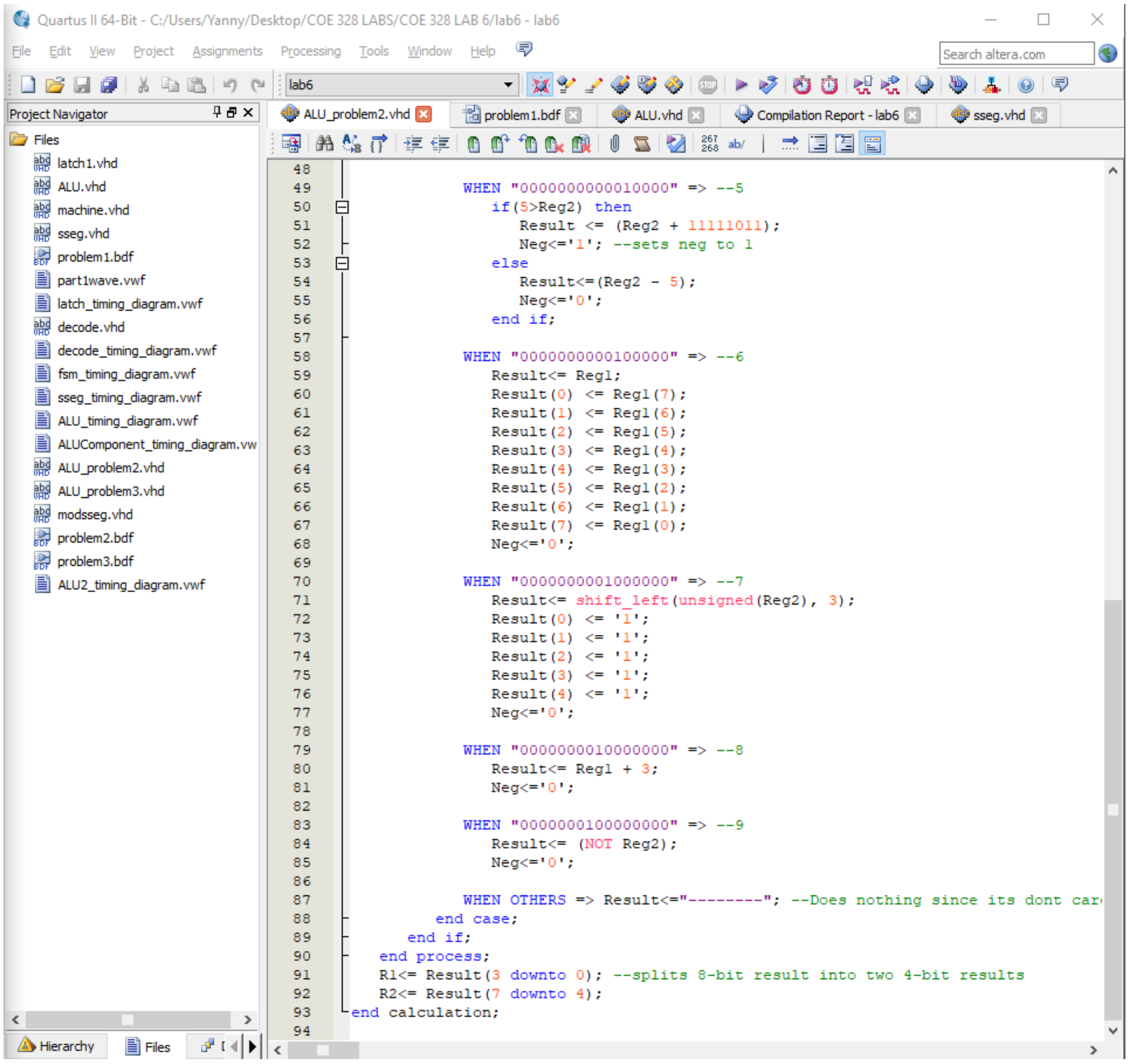
MICROCODE	BOOLEAN OPERATION
0000000000000001	Produce the difference between A and B
0000000000000010	Produce the 2's complement of B
0000000000000100	Swap the lower 4 bits of A with lower 4 bits of B
0000000000001000	Produce null on the output
0000000000010000	Decrement B by 5

0000000000100000	Invert the bit-significance order of A
0000000001000000	Shift B to left by three bits, input bit = 1 (SHL)
0000000010000000	Increment A by 3
0000000100000000	Invert all bits of B

Figure 13: VHDL Code for ALU (For Problem 2)



```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
entity ALU_problem2 is
port(Clk: 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 ALU_problem2;
architecture calculation of ALU_problem2 is
    signal Reg1,Reg2,Result: unsigned(7 downto 0):=(others=>'0');
    signal Reg4: unsigned(0 to 7);
    begin
    Reg1 <= A; --temporarily storing A into Reg1
    Reg2 <= B; --temporarily storing B into Reg2
    process(Clk, OP)
    begin
        if(rising_edge(Clk))THEN
            case OP is
                WHEN "0000000000000001" => --1
                    if(Reg2>Reg1) then
                        Result <= (Reg1 + (NOT Reg2 + 1));
                        Neg<='1'; --sets neg to 1
                    else
                        Result<=(Reg1-Reg2);
                        Neg<='0';
                    end if;

                WHEN "0000000000000010" => --2
                    Result <= (NOT Reg2 + 1);
                    Neg<='0';

                WHEN "0000000000000100" => --3
                    Result<= Reg1;
                    Result(0) <= Reg2(0);
                    Result(1) <= Reg2(1);
                    Result(2) <= Reg2(2);
                    Result(3) <= Reg2(3);
                    Neg<='0';

                WHEN "0000000000001000" => --4
                    Result<= null;
                    Neg<='0';
```



```vhdl

            WHEN "0000000000010000" => --5
               if(5>Reg2) then
                  Result <= (Reg2 + 11111011);
                  Neg<='1'; --sets neg to 1
               else
                  Result<=(Reg2 - 5);
                  Neg<='0';
               end if;

            WHEN "0000000000100000" => --6
               Result<= Reg1;
               Result(0) <= Reg1(7);
               Result(1) <= Reg1(6);
               Result(2) <= Reg1(5);
               Result(3) <= Reg1(4);
               Result(4) <= Reg1(3);
               Result(5) <= Reg1(2);
               Result(6) <= Reg1(1);
               Result(7) <= Reg1(0);
               Neg<='0';

            WHEN "0000000001000000" => --7
               Result<= shift_left(unsigned(Reg2), 3);
               Result(0) <= '1';
               Result(1) <= '1';
               Result(2) <= '1';
               Result(3) <= '1';
               Result(4) <= '1';
               Neg<='0';

            WHEN "0000000010000000" => --8
               Result<= Reg1 + 3;
               Neg<='0';

            WHEN "0000000100000000" => --9
               Result<= (NOT Reg2);
               Neg<='0';

            WHEN OTHERS => Result<="--------"; --Does nothing since its dont car
         end case;
      end if;
   end process;
   R1<= Result(3 downto 0); --splits 8-bit result into two 4-bit results
   R2<= Result(7 downto 4);
end calculation;

```

Figure 14: Block Diagram for the final GPU for problem 2

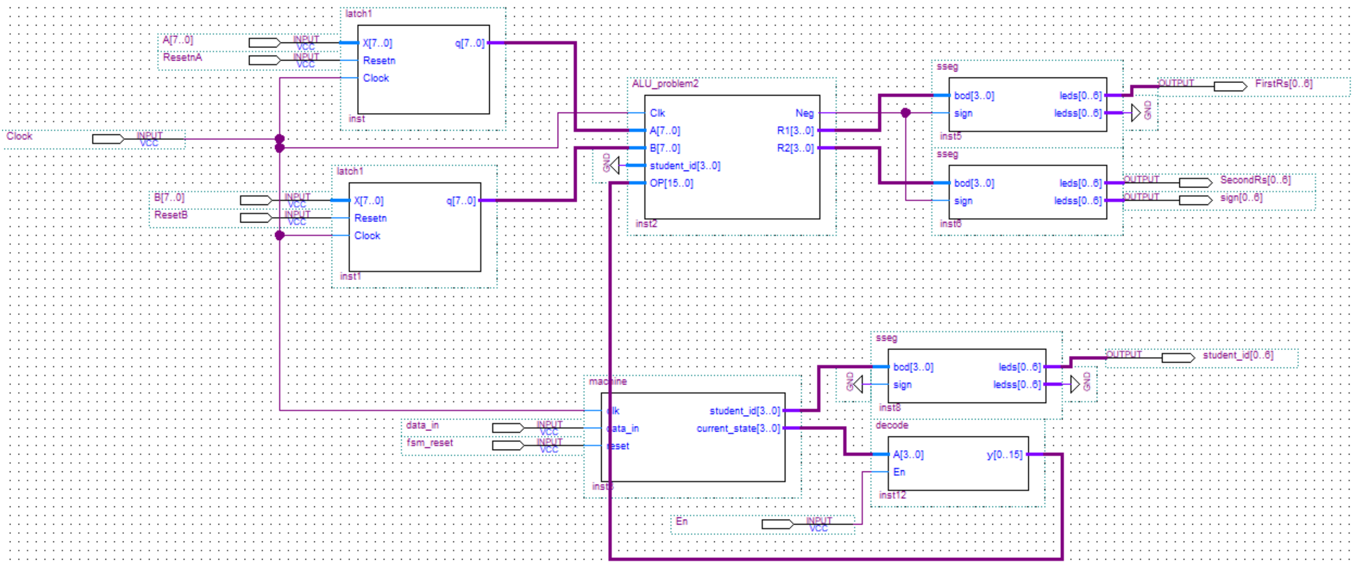
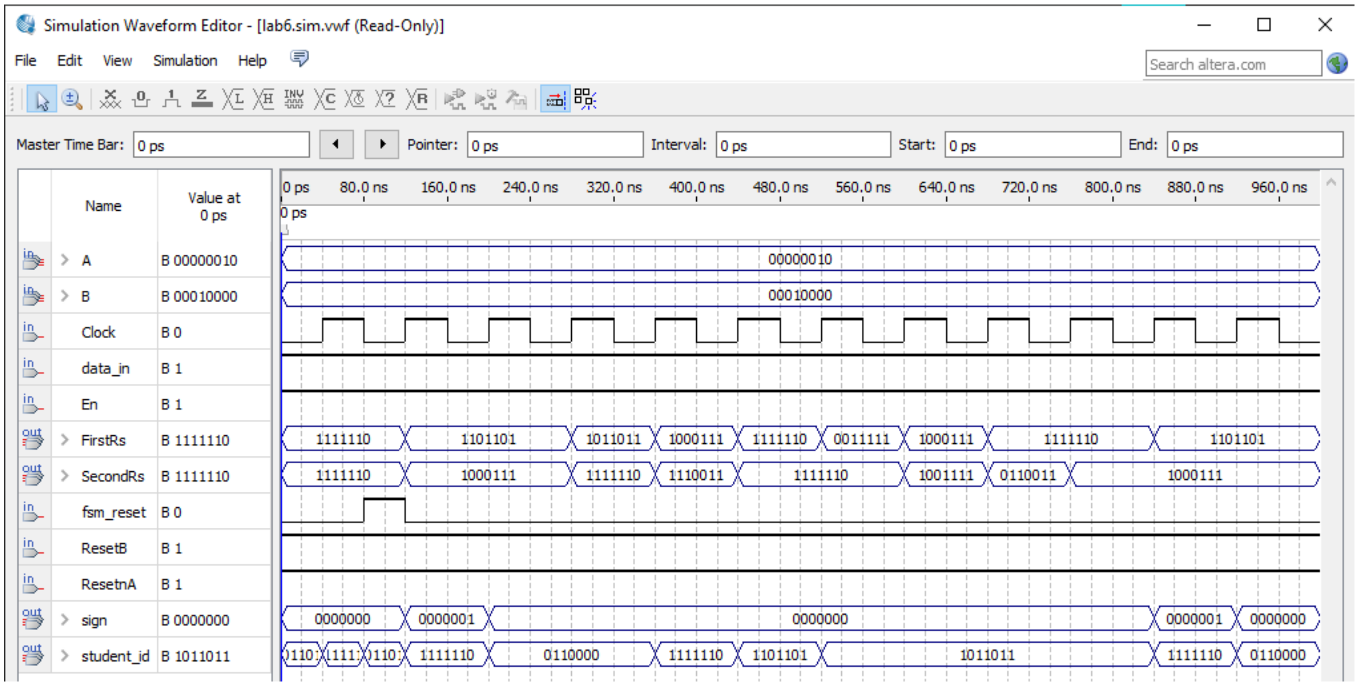


Figure 15: Compiled Waveform for the finished GPU



Simulation Waveform Editor - [lab6.sim.vwf (Read-Only)]
File Edit View Simulation Help
Search altera.com

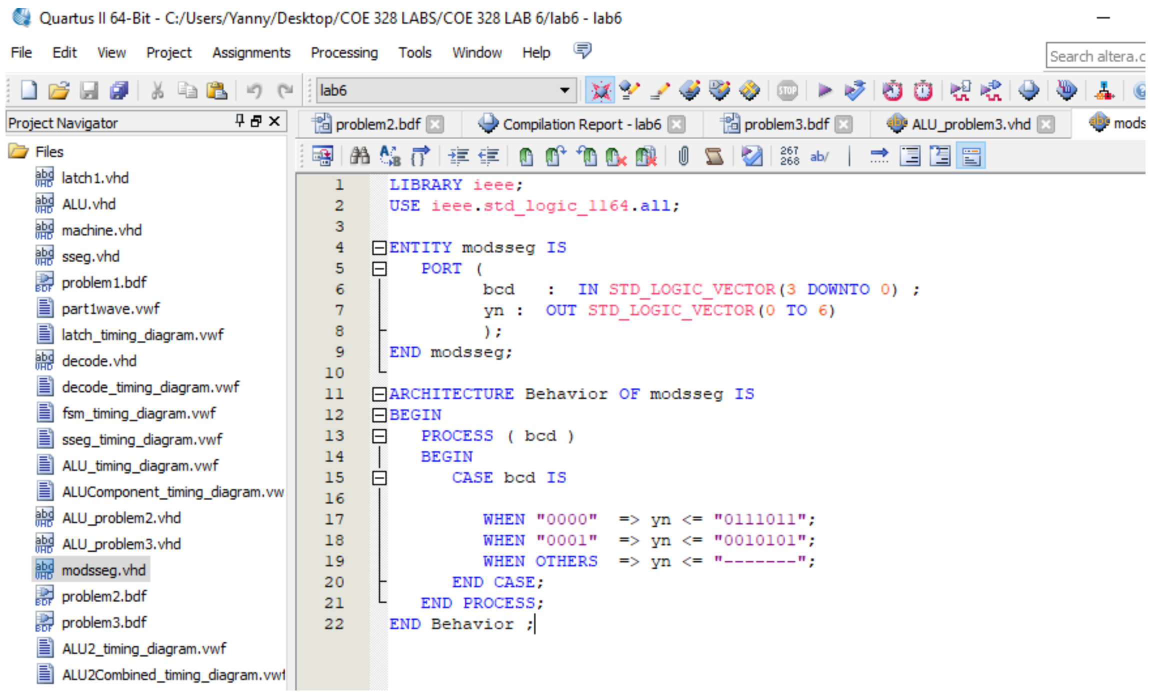
Master Time Bar: 0 ps Pointer: 0 ps Interval: 0 ps Start: 0 ps End: 0 ps

Name	Value at 0 ps
A	B 00000010
B	B 00010000
Clock	B 0
data_in	B 1
En	B 1
FirstRs	B 1111110
SecondRs	B 1111110
fsm_reset	B 0
ResetB	B 1
ResetnA	B 1
sign	B 0000000
student_id	B 1011011

7. MODIFIED 7-SEGMENT DISPLAY FOR PROBLEM 3 ALU

For the third part of the lab, the seven segment display code was modified to display a ‘y’ or ‘n’ based on the input. Essentially the objective of the third segment of the lab is to determine if the student number outputted at each microcode has an even parity or not. The main function of the ALU will further be discussed in the next section as the main focus of this section is the display code itself. If the value “0000” was inputted, the output would be “0111011” which creates a ‘y’ in the seven segment display. Likewise, the value “0001” produces the output, "0010101" which creates a 'n' in the display. Below the VHDL code, the circuit diagram and the produced waveform are shown. To verify if the component is operational, the inputs and outputs will be tested to see if the correct symbol is displayed.

Figure 16: VHDL Code for the modified seven segment display



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

ENTITY modsseg IS
   PORT (
         bcd    :  IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
         yn :  OUT STD_LOGIC_VECTOR(0 TO 6)
         );
END modsseg;

ARCHITECTURE Behavior OF modsseg IS
BEGIN
   PROCESS ( bcd )
   BEGIN
      CASE bcd IS

         WHEN "0000"  => yn <= "0111011";
         WHEN "0001"  => yn <= "0010101";
         WHEN OTHERS  => yn <= "-------";
      END CASE;
   END PROCESS;
END Behavior ;
```

Component 5: The circuit diagram for the modified sseg

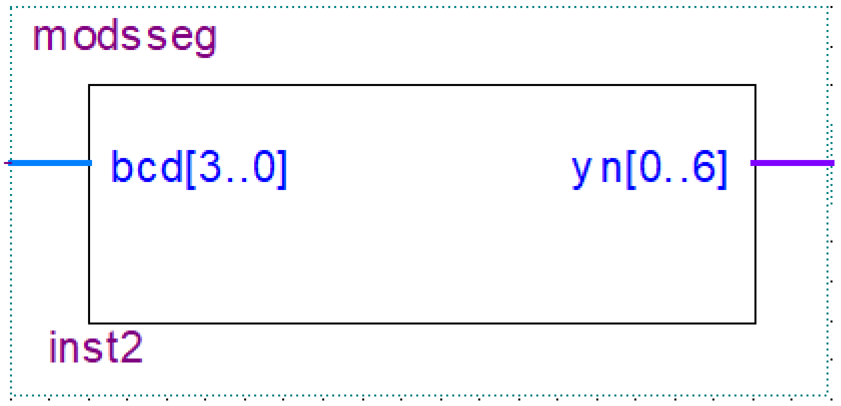
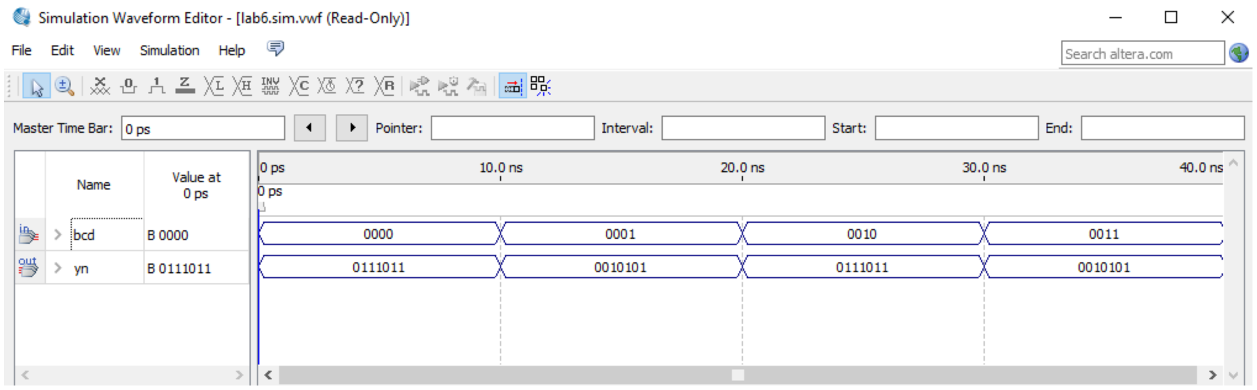


Figure 17: Compiled Waveform for the modified sseg



```
Name   Value at 0 ps
bcd    B 0000      0000      0001      0010      0011
yn     B 0111011   0111011   0010101   0111011   0010101
```

As seen above the desired outputs are achieved which means that the modified seven segment display code is operational and can be used for the third segment of the lab.

8. ALU (ARITHMETIC LOGICAL UNIT) - PROBLEM 3

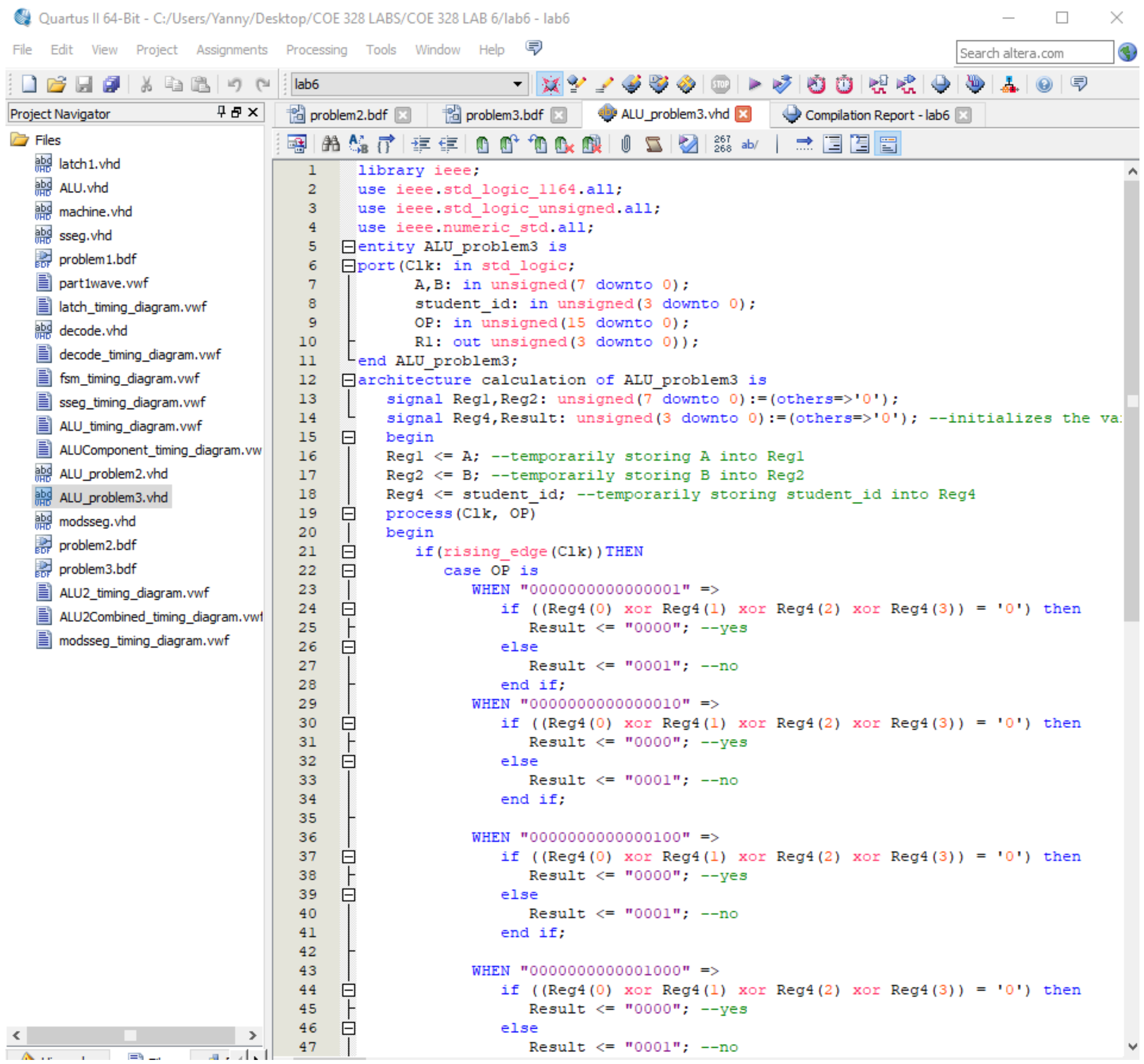
The main purpose of the third segment of the lab was to modify the logical unit to perform a single operation multiple times with different inputs which are fed in with each period of the clock. As mentioned earlier, the operation in this case was to check the inputted student id from the FSM and to check the parity. When the amount of 1's in a binary number are even, it is said that the number has even parity. To check this condition, the inputted 4-bit student id was split into separate bits and XOR'd. This meant that if there was an even parity, the result would have a value of "0000" which would output 'y' when sent to the modified seven segment display code. As seen below in *Figure 18*, the ALU code was changed quite a bit but the overall structure of the code remained unchanged. Firstly, "Result" was changed from a 8 bit to a 4-bit signal and the signal, and "Reg4" was assigned the input, "student_id". Next the main logic for the parity check was written and pasted into each microcode condition so the same logic is executed regardless of the microcode. Overall the ALU now the same 5 inputs but only one output, "yn". As opposed to the previous segments of the lab, the "student_id" input is now used in the actual logic of the ALU.

Table 7: ALU operations with corresponding microcode for problem 3

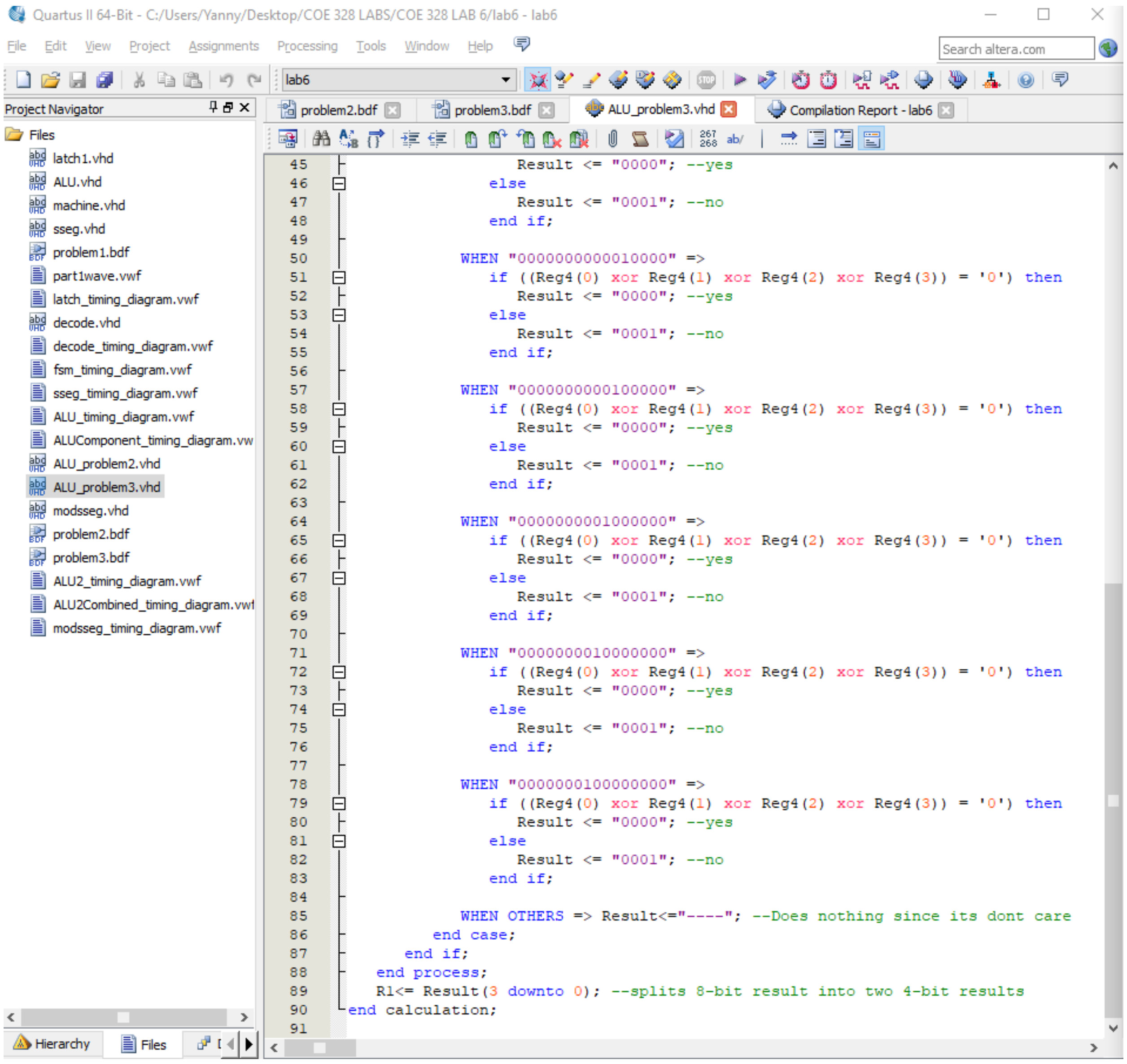
MICROCODE	BOOLEAN OPERATION	Student Number	Expected Result
0000000000000001	Check even parity	5 (0101)	Y
0000000000000010	Check even parity	0 (0000)	N
0000000000000100	Check even parity	1 (0001)	N
0000000000001000	Check even parity	1 (0001)	N

0000000000010000	Check even parity	0 (0000)	N
0000000000100000	Check even parity	2 (0010)	N
0000000001000000	Check even parity	5 (0101)	Y
0000000010000000	Check even parity	5 (0101)	Y
0000000100000000	Check even parity	5 (0101)	Y

Figure 18: VHDL Code for ALU (For Problem 3)



```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
entity ALU_problem3 is
port(Clk: in std_logic;
      A,B: in unsigned(7 downto 0);
      student_id: in unsigned(3 downto 0);
      OP: in unsigned(15 downto 0);
      R1: out unsigned(3 downto 0));
end ALU_problem3;
architecture calculation of ALU_problem3 is
   signal Reg1,Reg2: unsigned(7 downto 0):=(others=>'0');
   signal Reg4,Result: unsigned(3 downto 0):=(others=>'0'); --initializes the va
   begin
   Reg1 <= A; --temporarily storing A into Reg1
   Reg2 <= B; --temporarily storing B into Reg2
   Reg4 <= student_id; --temporarily storing student_id into Reg4
   process(Clk, OP)
   begin
      if(rising_edge(Clk))THEN
         case OP is
            WHEN "0000000000000001" =>
               if ((Reg4(0) xor Reg4(1) xor Reg4(2) xor Reg4(3)) = '0') then
                  Result <= "0000"; --yes
               else
                  Result <= "0001"; --no
               end if;
            WHEN "0000000000000010" =>
               if ((Reg4(0) xor Reg4(1) xor Reg4(2) xor Reg4(3)) = '0') then
                  Result <= "0000"; --yes
               else
                  Result <= "0001"; --no
               end if;

            WHEN "0000000000000100" =>
               if ((Reg4(0) xor Reg4(1) xor Reg4(2) xor Reg4(3)) = '0') then
                  Result <= "0000"; --yes
               else
                  Result <= "0001"; --no
               end if;

            WHEN "0000000000001000" =>
               if ((Reg4(0) xor Reg4(1) xor Reg4(2) xor Reg4(3)) = '0') then
                  Result <= "0000"; --yes
               else
                  Result <= "0001"; --no
```



```
                    Result <= "0000"; --yes
                else
                    Result <= "0001"; --no
                end if;

            WHEN "0000000000010000" =>
                if ((Reg4(0) xor Reg4(1) xor Reg4(2) xor Reg4(3)) = '0') then
                    Result <= "0000"; --yes
                else
                    Result <= "0001"; --no
                end if;

            WHEN "0000000000100000" =>
                if ((Reg4(0) xor Reg4(1) xor Reg4(2) xor Reg4(3)) = '0') then
                    Result <= "0000"; --yes
                else
                    Result <= "0001"; --no
                end if;

            WHEN "0000000001000000" =>
                if ((Reg4(0) xor Reg4(1) xor Reg4(2) xor Reg4(3)) = '0') then
                    Result <= "0000"; --yes
                else
                    Result <= "0001"; --no
                end if;

            WHEN "0000000010000000" =>
                if ((Reg4(0) xor Reg4(1) xor Reg4(2) xor Reg4(3)) = '0') then
                    Result <= "0000"; --yes
                else
                    Result <= "0001"; --no
                end if;

            WHEN "0000000100000000" =>
                if ((Reg4(0) xor Reg4(1) xor Reg4(2) xor Reg4(3)) = '0') then
                    Result <= "0000"; --yes
                else
                    Result <= "0001"; --no
                end if;

            WHEN OTHERS => Result<="----"; --Does nothing since its dont care
        end case;
      end if;
    end process;
    R1<= Result(3 downto 0); --splits 8-bit result into two 4-bit results
end calculation;
```

Figure 19: Block Diagram for the final GPU for problem 3

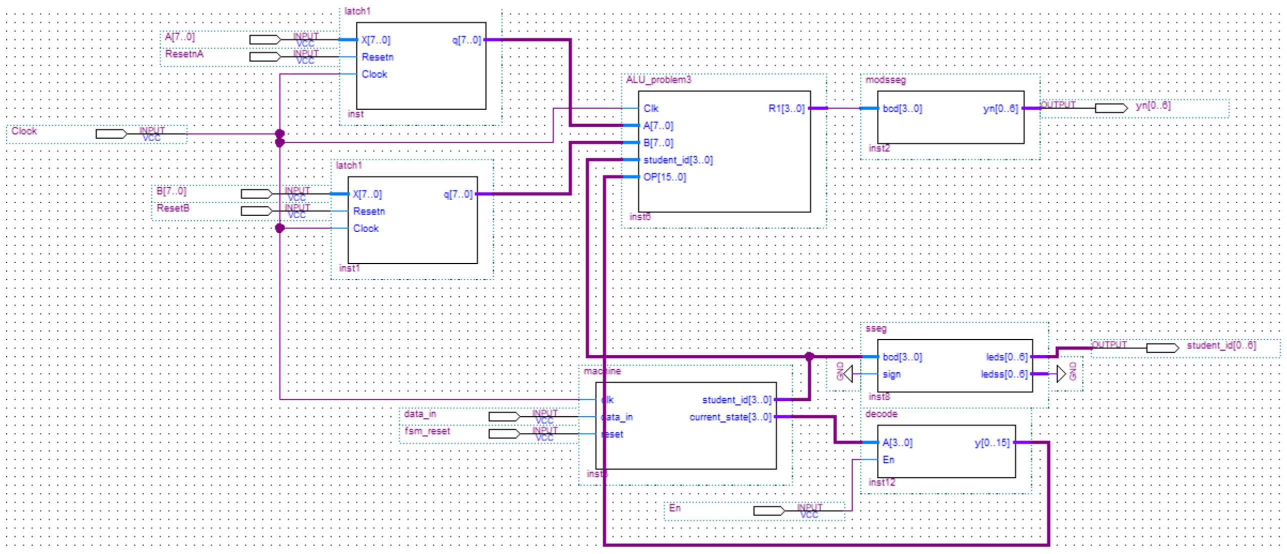
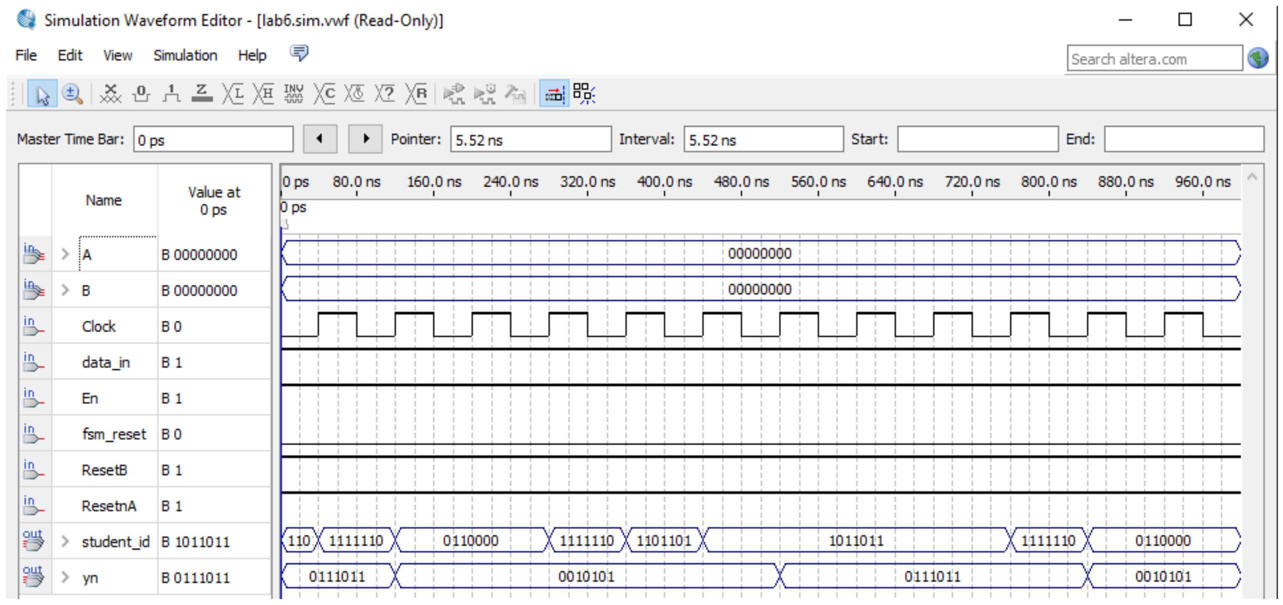


Figure 20: Compiled Waveform for the finished GPU



Name	Value at 0 ps	
A	B 00000000	00000000
B	B 00000000	00000000
Clock	B 0	
data_in	B 1	
En	B 1	
fsm_reset	B 0	
ResetB	B 1	
ResetnA	B 1	
student_id	B 1011011	110, 1111110, 0110000, 1111110, 1101101, 1011011, 1111110, 0110000
yn	B 0111011	0111011, 0010101, 0111011, 0010101

CONCLUSION:

Overall the target of this lab was met as the main purpose was to combine the previous lab experiments with the new addition of sequential circuits and the arithmetic logical unit. Using VHDL code, block diagrams and components from previous labs, each of the units in the ALU's were assembled. In summary, the latches serve as the storage unit for the 8-bit inputs, A and B which are then fed into the ALU. For the control unit, the clock which is used for all the components is fed into the FSM, which produces the desired student id and the current state number. The current state number serves as a way to control the operations done by the ALU as each state is fed into the decoder and turned into a microcode that is then sent into the ALU.

Next the ALU performs the required operations to A and B and splits the final 8-bit result into two 4-bit results which are then fed into two seven segment converters that create a 7-bit value suitable for seve segment displays. It can be seen that each component serves a vital purpose in the function of the processor ranging from the storage units to the ALU itself. Overall all the components and parts of the lab were successfully constructed and each yielded the desired output. Almost all the VHDL code was operational on the first go and after some changes each component compiled successfully and the expected waveforms were produced. In conclusion the main objective of the lab was met and the components of the lab were operation which meant that the construction of the three GPUs was a success.