Introduction

Although the lab was divided into three sections, the overall goal was to build a basic general-purpose processor. The first section of the lab needed an Arithmetic Logic Unit (ALU), which performed several duties in various states. Using prior lab components such as a finite state machine (FSM), a 4x16 decoder, and a Seven Segment Display (SSEG). These were linked together using a block diagram to display a student number as well as the solutions to each provided function. Part 2 requires changing the ALU into the same block diagram, but now displaying new, more difficult functions using A and B as input, with "B" being the final two digits of the student number and "A" being the two before. The final portion of this experiment had us compare the student number to the values of "B" to see if any digit of "B" was less than each value of the student number. This was accomplished by altering both the ALU to compare and the SSEG code to show a "Y" or "N" value. A student number SSEG block and a modified SSEG block were added to the block diagram. A waveform was constructed for each portion of the lab in order to input values for "A" and "B," reset the clock and FSM, and show the solutions to the functions and student number.

Results

Components: These are the necessary components that need to be implemented before starting the lab.

Seven Segment Display

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

ENTITY ssegNEW IS
   PORT (
      bcd : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      Sign : IN STD_LOGIC;
      leds : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
   ledss : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)); -- -VE sign
END ssegNEW;

ARCHITECTURE Behavior OF ssegNEW IS
BEGIN
   PROCESS (bcd)
   BEGIN
      CASE bcd IS -- abcdefg ON LED board
         WHEN "0000" => leds <= "1111110"; -- 0
         WHEN "0001" => leds <= "0110000"; -- 1
         WHEN "0010" => leds <= "1101101"; -- 2
         WHEN "0011" => leds <= "1111001"; -- 3
         WHEN "0100" => leds <= "0110011"; -- 4
         WHEN "0101" => leds <= "1011011"; -- 5
         WHEN "0110" => leds <= "1011111"; -- 6
         WHEN "0111" => leds <= "1110000"; -- 7
         WHEN "1000" => leds <= "1111111"; -- 8
         WHEN "1001" => leds <= "1111011"; -- 9
         WHEN "1010" => leds <= "1110111"; -- A
         WHEN "1011" => leds <= "0011111"; -- B
         WHEN "1100" => leds <= "1001110"; -- C
         WHEN "1101" => leds <= "0111101"; -- D
         WHEN "1110" => leds <= "1001111"; -- E
         WHEN "1111" => leds <= "1000111"; -- F
         WHEN OTHERS => leds <= "-------"; -- _
```

```
lected device family is not supported by the report_metastability command.
 is not fully constrained for setup requirements
```

```vhdl
      END CASE;
   END PROCESS;
   PROCESS (Sign)
      BEGIN
         CASE Sign IS
            WHEN '1' => ledss <= "0000001";
            WHEN '0' => ledss <= "0000000";
         END CASE;
      END PROCESS;
END Behavior;
```

```
ected device family is not supported by the report_metastability command.
is not fully constrained for setup requirements
```

This is the Seven Segment Display (SSEG) VHDL program, and the program determines the output of the components. The 1's represent a "on light," while the 0's represent a "off light," and by combining the 1's and 0's, we can get single digit numbers (0-9) or letters in HEX (A-F or 10-15) that represent the corresponding value in decimal, which are utilized to calculate the output from our inputs. When we have a negative decimal value, the variable ledss will show "0000001" or a "-" on SSEG to indicate that it is negative. However, if the number is larger than 0, the ledss value will be "0000000," indicating that there is no "-" and the value is positive.

Seven Segment Modified

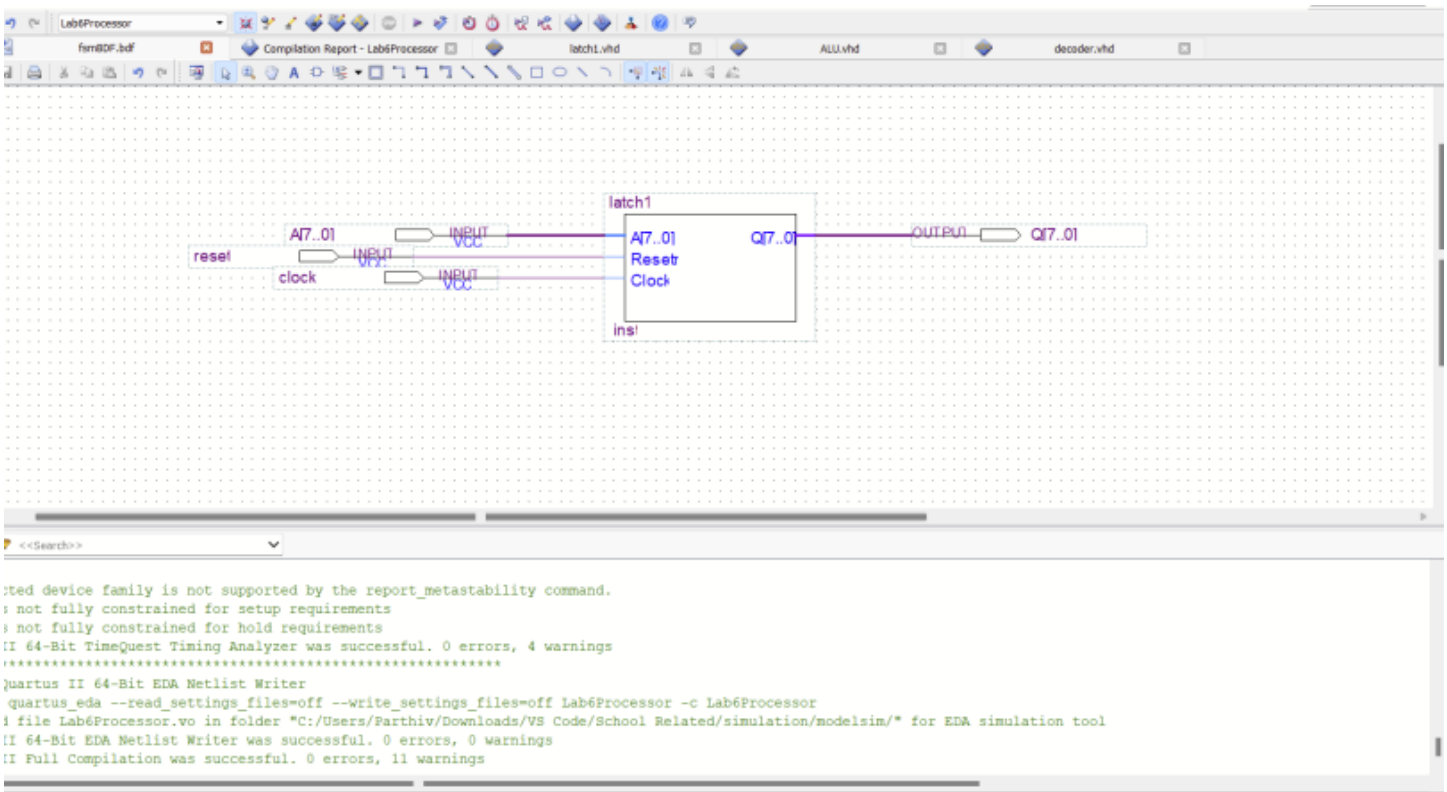
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;


Entity ssegNEWMOD IS
PORT(
    bcd: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
    Cout: IN  STD_LOGIC;
    leds: OUT  STD_LOGIC_VECTOR(0 TO 6);
    negleds: OUT STD_LOGIC_VECTOR(0 TO 6));
END ssegNEWMOD;

ARCHITECTURE Behaviour OF ssegNEWMOD IS
BEGIN
   PROCESS(bcd)
   BEGIN
      CASE bcd IS -- abcdefg
         WHEN "0000" => leds <= "1110110";
         WHEN "0001" => leds <= "0111011";
         WHEN OTHERS => leds <= "-------";
      END CASE;
   END PROCESS;
   PROCESS(Cout)
   BEGIN
      CASE Cout IS
         WHEN  '0' => negleds <= "0000000";
         WHEN  '1' => negleds <= "0000001";
      END CASE;
   END PROCESS;
END Behaviour;
```

```
II Full Compilation was successful. 0 errors, 27 warnings
```

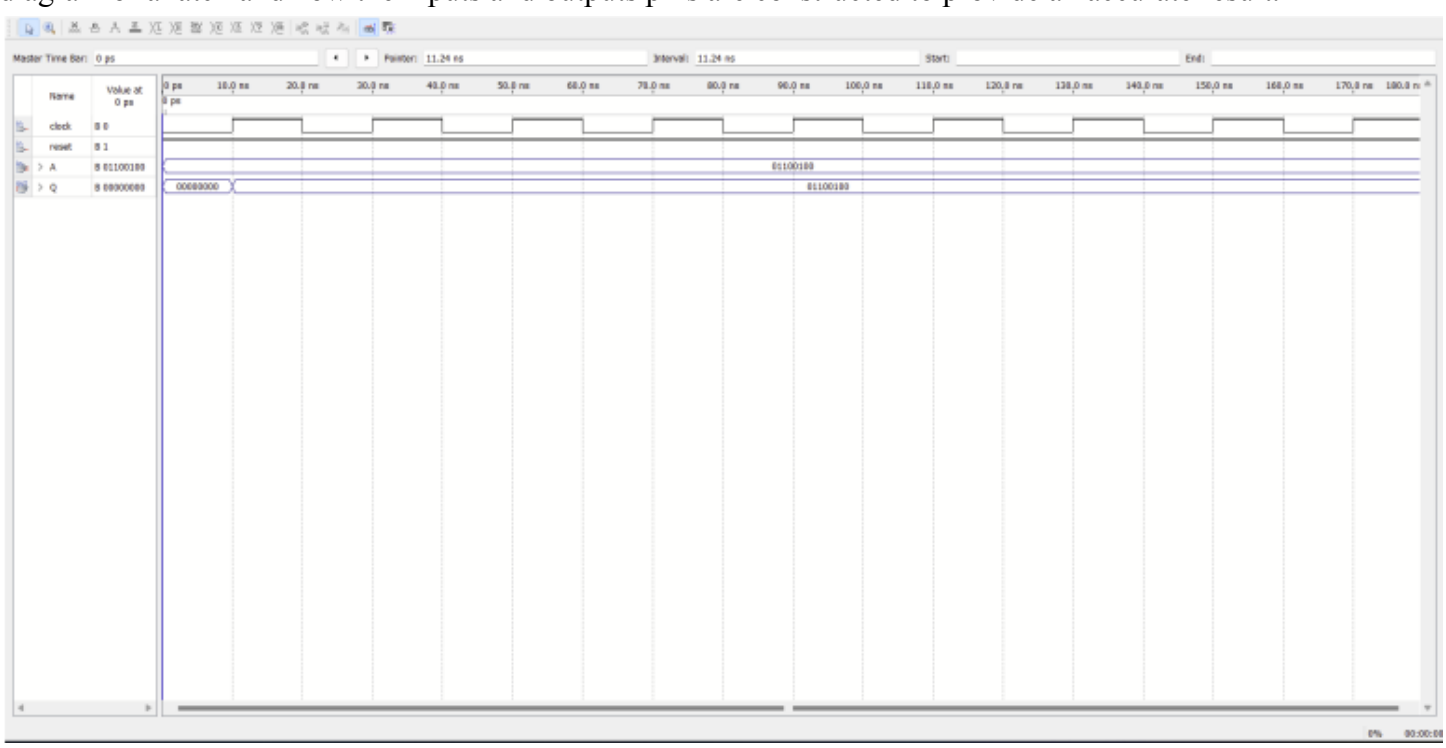
This is the VHDL program of the seven-segment display modified, the purpose of this is to print “y” or “n”, which is used in part 3 of the lab. When the conditions are true the output should be “Y”, which is “y” on the SSEG and when the conditions are false the output should be “N”, which is “n” on the SSEG. Therefore, with the help of his code, we will be able to successfully complete part 3 of the lab.

Latch

```
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 Behavior 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 Behavior;
```



This figure represents the VHDL program to simulate a latch. A latch is a data storage device that uses the feedback lane to store data. Until the device is set to 1, the latch saves 1 -bit. When the enable input is set to 1, the latch alters the stored data and continually tries the inputs. The bottom figure represents the block diagram of a latch and how the inputs and outputs pins are constructed to provide an accurate result.



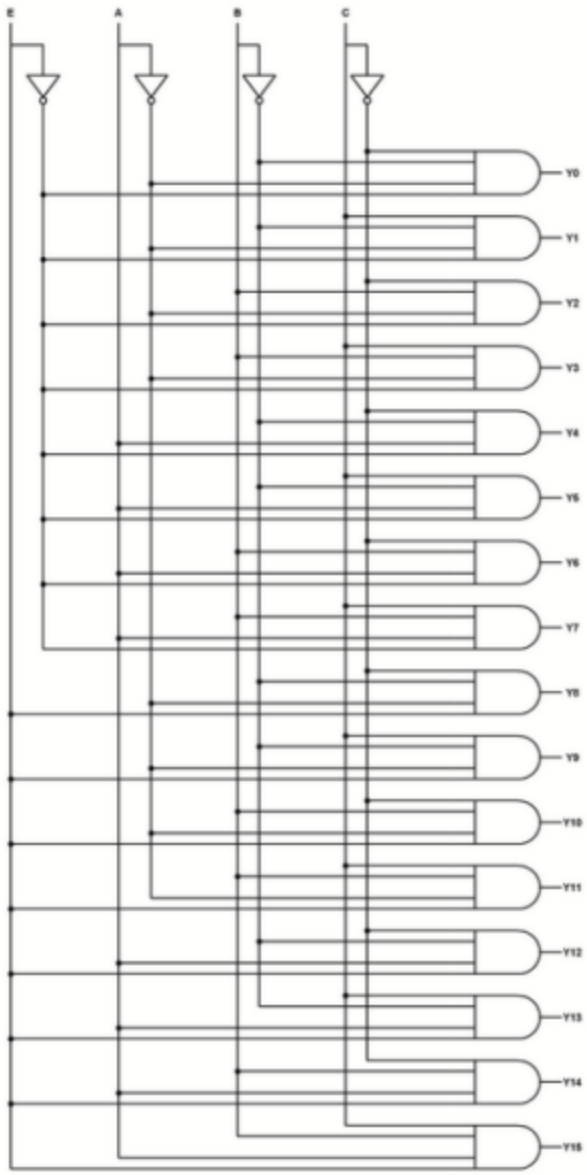
This figure represents the latch whose function is to momentarily store the input A and then deliver it as the output. Since there is “0000000” meaning no output for the first input, input A is temporarily held here. Following that, the input is transmitted as the output.

Decoder

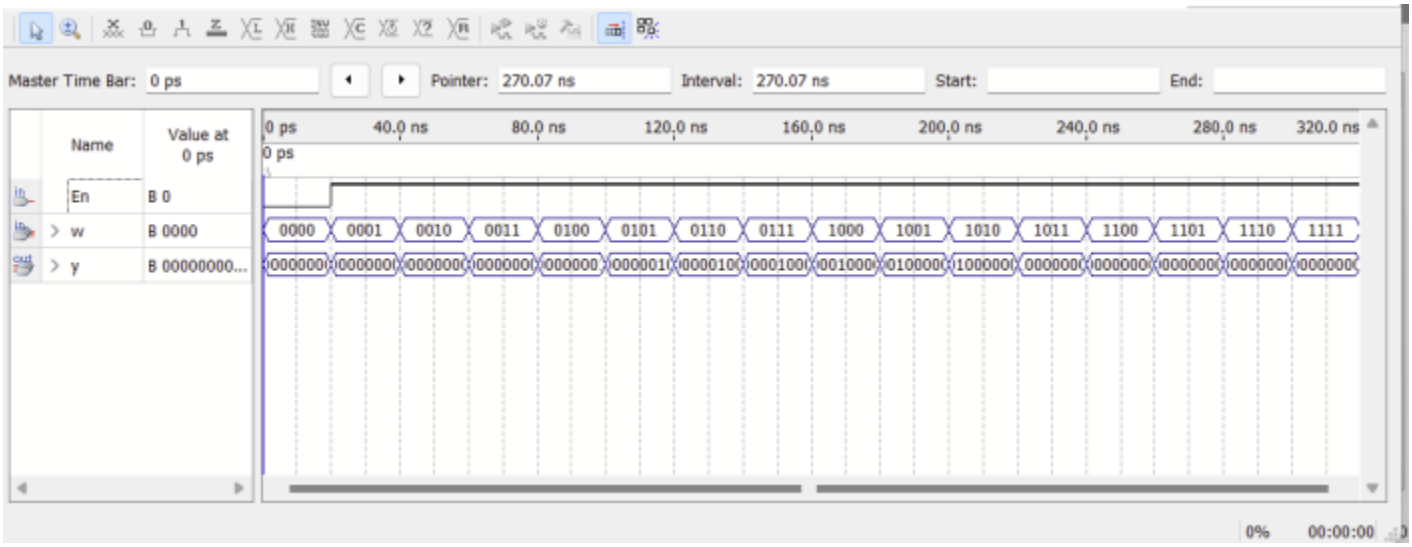
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY decoder IS
    PORT (
        A : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        En : IN STD_LOGIC;
        D : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
    );
END decoder;
ARCHITECTURE Behavior OF decoder IS
    SIGNAL EnA : STD_LOGIC_VECTOR(4 DOWNTO 0);
BEGIN
    EnA <= En & A;
    WITH EnA SELECT
    D <= "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;
```

This figure represents a VHDL program that simulates a decoder that is a logic gate-based combinational circuit. A decoder circuit converts a collection of digital input signals into its output's corresponding decimal code. A decoder produces 2N outputs for N inputs. A decoder is a logic gate-based combinational circuit. Moreover, here we see a 4x16 decoder and a 4x16 decoder has 4 inputs and 16 outputs, with the outputs turning high for each 4-bit input.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
E	A	B	C	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11	Y12	Y13	Y14	Y15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



These figures represent the truth table and circuit diagram of a decoder. We can see how the 4x16 decoder is implemented using logic gates.



This figure is the waveform of the decoder. We can see how there are lots more outputs than there are inputs. This is due to the logic gates and truth table shown above.

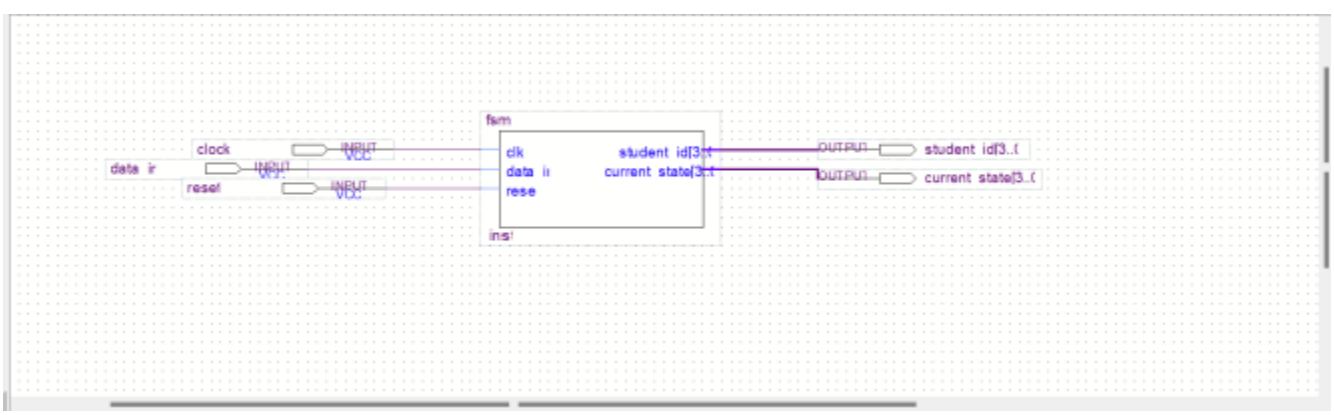
FSM

```
library ieee;
use ieee.std_logic_1164.all;
entity Lab6_5 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 Lab6_5 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 <= 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;
```

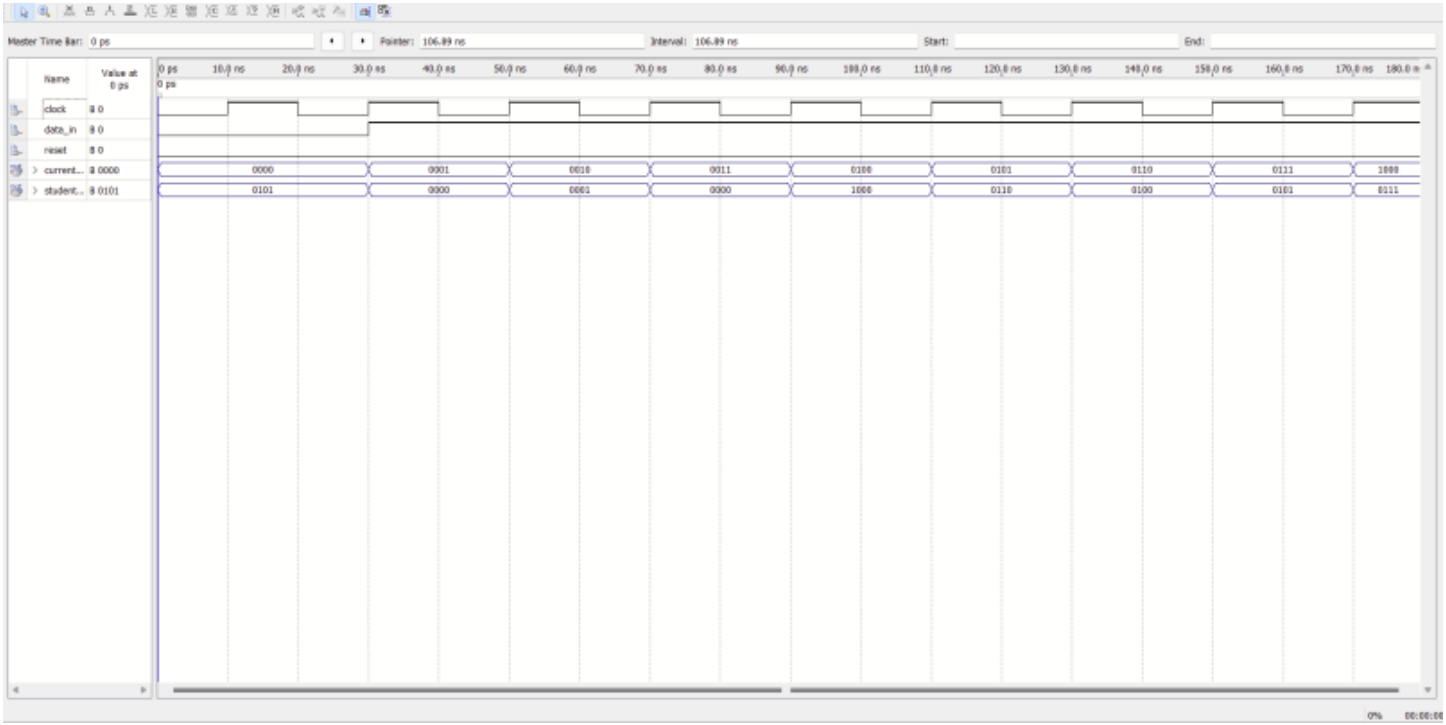
```
if data_in = '1' then
yfsm <= s3;
else yfsm <= s2;
end if;

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 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 <="0001";
current_state <= "0010";

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

when s4 =>
student_id <="1000";
current_state <= "0100";

when s5 =>
student_id <="0111";
current_state <= "0101";

when s6 =>
student_id <="1000";
current_state <= "0110";

when s7 =>
student_id <="0110";
current_state <= "0111";

when s8 =>
student_id <="0001";
current_state <= "1000";

end case;

end process;

end architecture;
```

The VHDL code for the Finite State Machine is depicted in these pictures. We can see how we have four input clocks that will keep the outputs and inputs steady, and how clock pulses trigger flip-flops. We have data in, which changes the state to 0 or 1 based on the input, and reset, which puts the state machine in a predefined state. We have the student id output, which represents each digit of the student ID (501087861). Finally, we have the output current state, which will maintain track of the current state of the FSM and output the student id depending on that state. Later on, we find that we have state types of s0-s8, which reflect what the next state is when the mealy machine goes on when the data in is either 0 or 1. Following that are conditional expressions that will maintain track of the machine's current state depending on the FSM states and when data in input is either 0 or 1. If the input is 0, the state remains unchanged; if the input is 1, the state changes while showing the digit in the student ID at that time. The program's bottom section shows the binary representation of each digit in the student ID for each state. This will help the SSEG in understanding what needs to be outputted.



This figure represents the block diagram of the FSM, we can see how the input and outputs are connected.

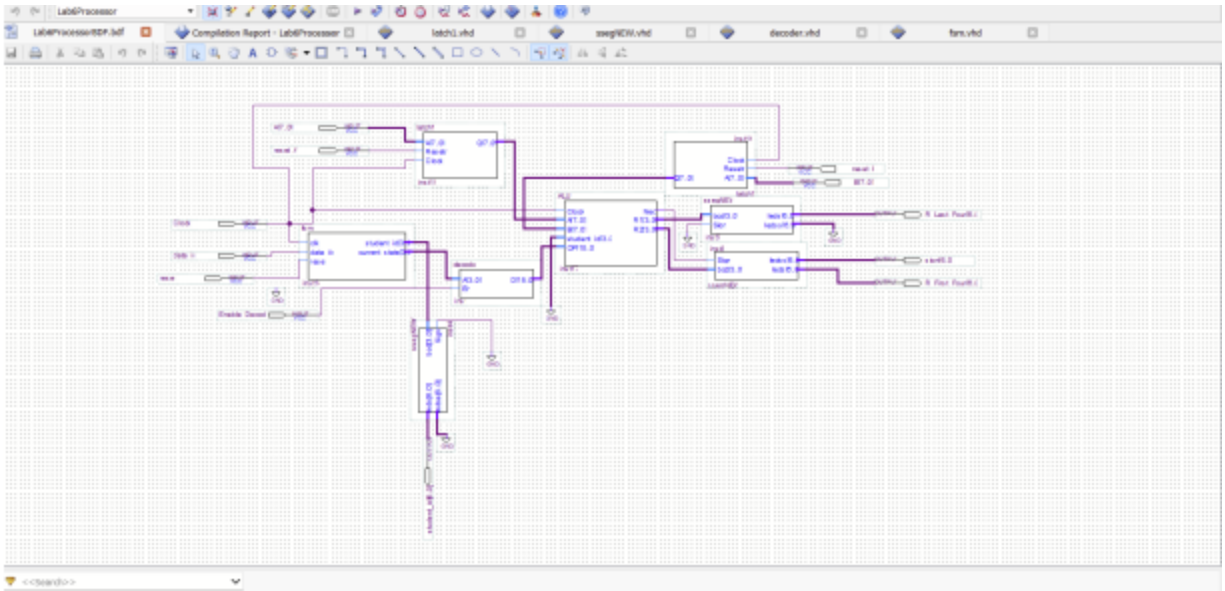


We can see that this figure is the correct waveform of the FSM since it displays the current states from 0-8, which we defined in the VHDL program shown above and at each state it represents the correct student ID binary value. For example, at start 0 it shows 0101(5). 2nd is 0000(0), 3rd is 0001(1) and so on. Which is the correct output shown in the VHDL program.

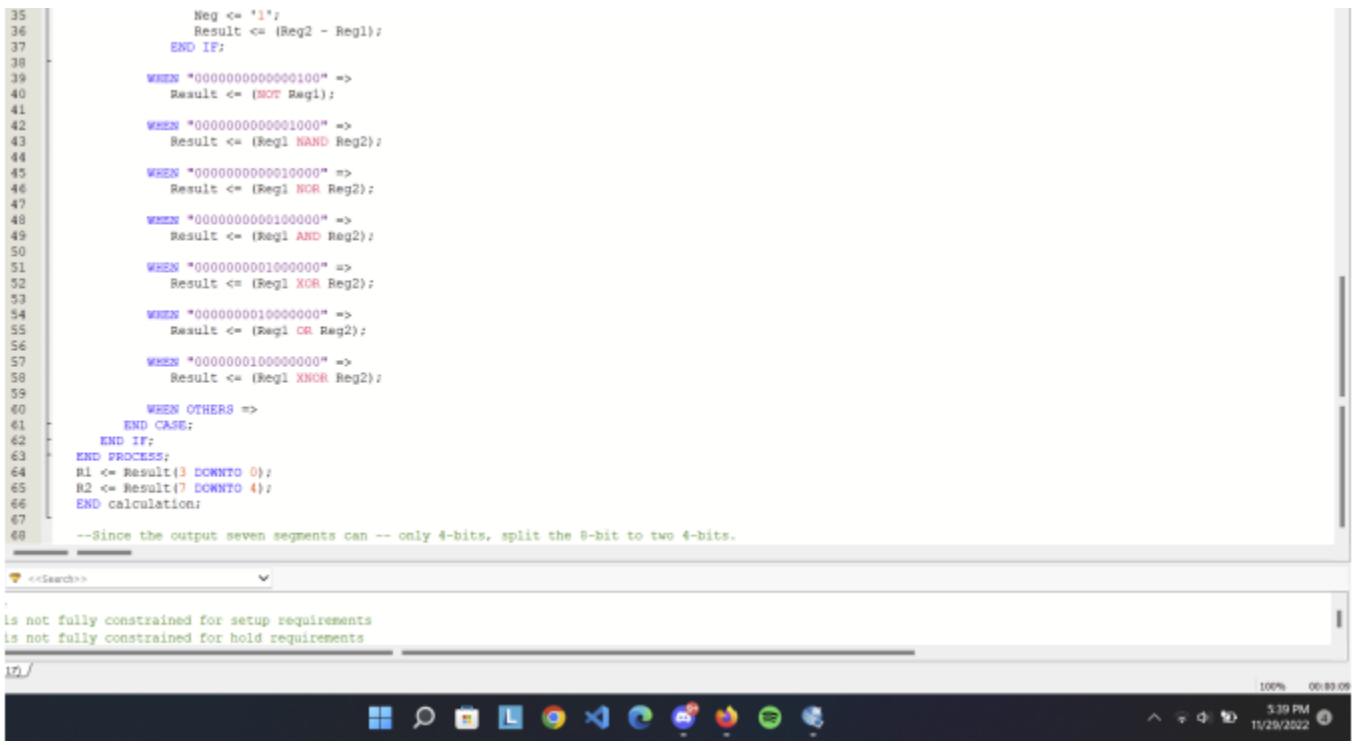
Part 1

To finish the processor for parts one to six, an ALU must be built. An ALU is a component that receives a 16-bit input (current state) from the decoder and performs a specific operation on A and B (two inputs generated by the latch outputs). Due to the present state switching to the next states, the operation that the ALU executes will likewise change, however, the student ID will stay constant. Finally, the outcome is then shown on the SSEG.

```
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; --input clock signal
      A, B : IN unsigned(7 DOWNTO 0); --8-bit inputs from latches A and B
      student_id : IN unsigned(3 DOWNTO 0); --4 bit student id from FSM
      OP : IN unsigned(15 DOWNTO 0); --16-bit selector for Operation from Decoder
      Neg : OUT std_logic; --is the result negative ? Set-ve bit output
      R1 : OUT unsigned(3 DOWNTO 0); -- lower 4-bits of 8-bit Result Output
   R2 : OUT unsigned(3 DOWNTO 0)); -- higher 4-bits of 8-bit Result Output
END ALU;

ARCHITECTURE calculation OF ALU IS --temporary signal declarations.
   SIGNAL Reg1, Reg2, Result : unsigned(7 DOWNTO 0) := (OTHERS => '0');
   SIGNAL Reg4 : unsigned (7 DOWNTO 0);
BEGIN
   Reg1 <= A; --temporarily store A in Reg1 local variable
   Reg2 <= B; --temporarily store B in Reg2 local variable
   PROCESS (Clock, OP)
   BEGIN
      IF (rising_edge(Clock)) THEN --Do the calculation @ positive edge of clock cycle.
         CASE OP IS
            WHEN "0000000000000001" =>
               Result <= (Reg1 + Reg2);

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



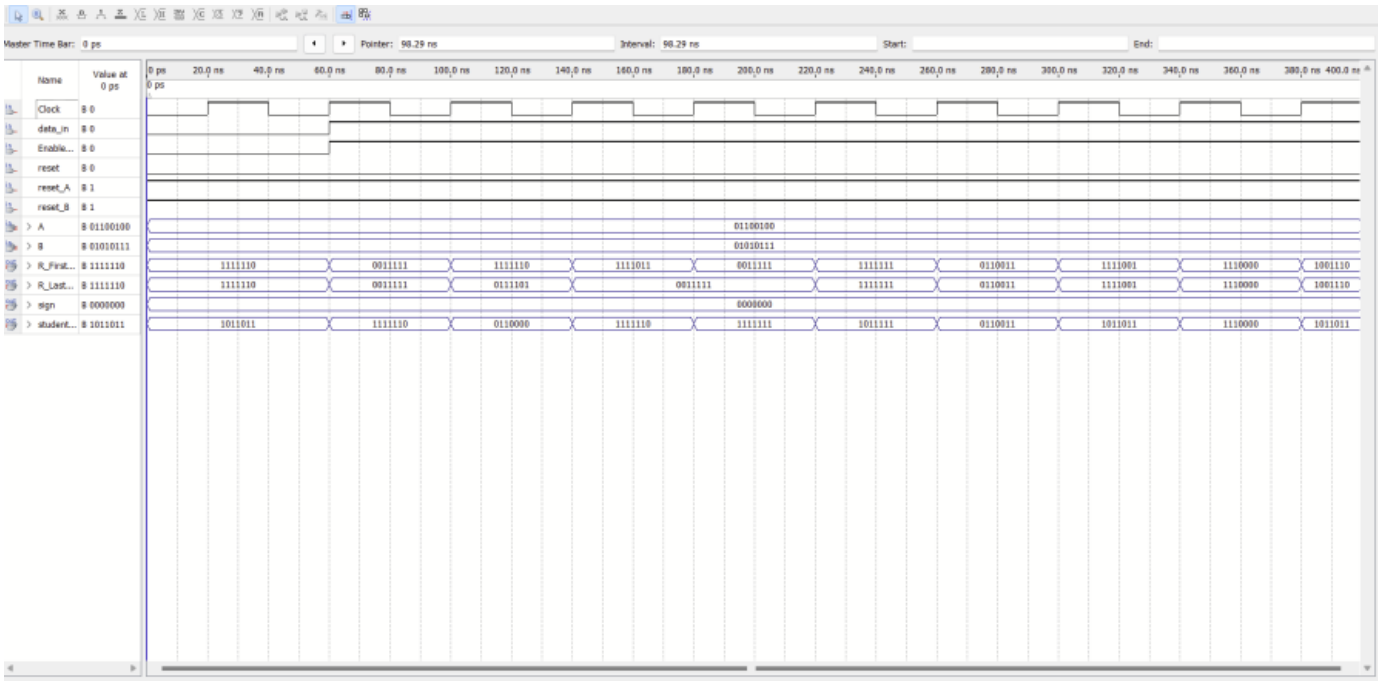
This figure represents the Block Diagram for problems 1 to 6, where we connect the components together to get the desired output.



```
                Neg <= '1';
                Result <= (Reg2 - Reg1);
            END IF;

        WHEN "00000000000000100" =>
            Result <= (NOT Reg1);

        WHEN "00000000000001000" =>
            Result <= (Reg1 NAND Reg2);

        WHEN "00000000000010000" =>
            Result <= (Reg1 NOR Reg2);

        WHEN "00000000000100000" =>
            Result <= (Reg1 AND Reg2);

        WHEN "00000000001000000" =>
            Result <= (Reg1 XOR Reg2);

        WHEN "00000000010000000" =>
            Result <= (Reg1 OR Reg2);

        WHEN "00000000100000000" =>
            Result <= (Reg1 XNOR Reg2);

        WHEN OTHERS =>
    END CASE;
  END IF;
END PROCESS;
R1 <= Result(3 DOWNTO 0);
R2 <= Result(7 DOWNTO 4);
END calculation;

--Since the output seven segments can -- only 4-bits, split the 8-bit to two 4-bits.
```

```
is not fully constrained for setup requirements
is not fully constrained for hold requirements
```

This figure represents an ALU. This is where all the computation occurs and where we enter the input for each case and get the output once the system is connected. This means its is the portion of a central processing unit that performs arithmetic and logic operations for problem 1.



This figure represents the waveform for the block diagram.

Part 2

We were given particular functions with varied microcode instructions for the ALU to compute in the second set. This ALU works in the same way as the first. Similar to the first diagram, this diagram takes two latch outputs as inputs and then, based on the decoder output, executes the relevant operation on the two latch outputs. Therefore, the only components that were modified were the ALU and the Block Diagram. Thus, the result is then divided into two outputs, each representing the first and last 4 digits of the binary value.

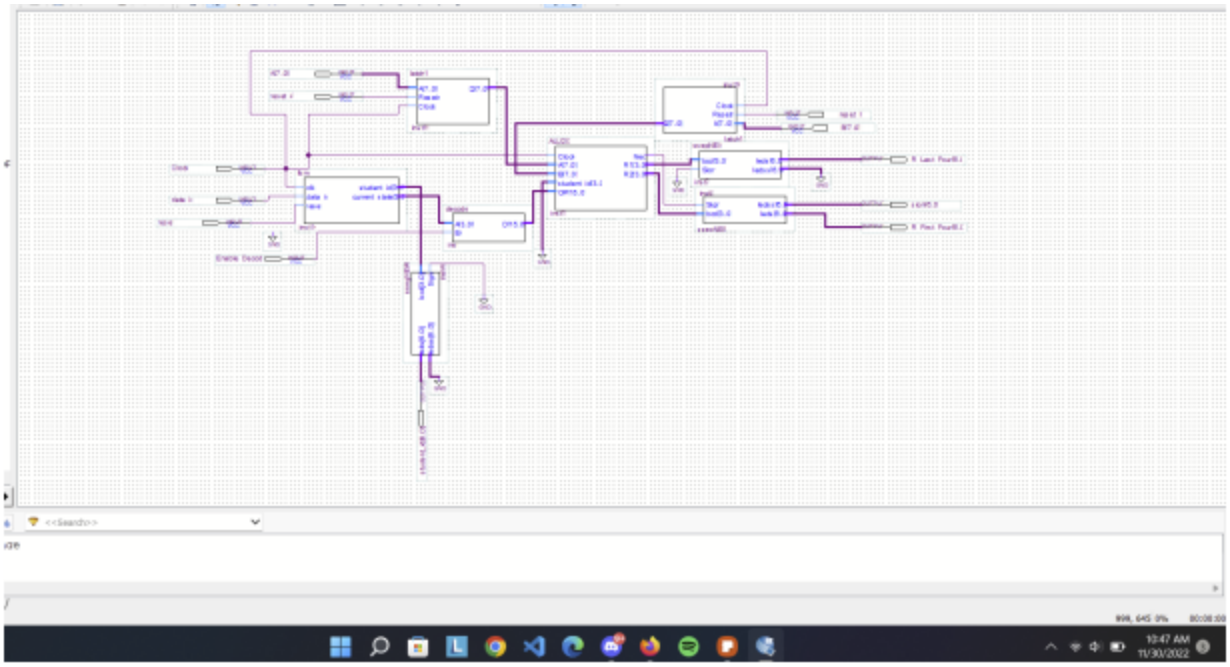
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY ALU2G IS
   PORT (
      Clk : IN std_logic;
      A, B : IN unsigned(7 DOWNTO 0);
      student_id : IN unsigned(3 DOWNTO 0); --4 bit student id from FSM
      OP : IN unsigned(15 DOWNTO 0);
      Neg : OUT std_logic;
      R1 : OUT unsigned(3 DOWNTO 0);
      R2 : OUT unsigned(3 DOWNTO 0)
   );
END ALU2G;
ARCHITECTURE calculation OF ALU2G IS
   SIGNAL Reg1, Reg2, Result : unsigned(7 DOWNTO 0) := (OTHERS => '0');
   SIGNAL Reg4 : unsigned(0 TO 7);
BEGIN
   Reg1 <= A;
   Reg2 <= B;
   PROCESS (Clk, OP)
   BEGIN
      IF (rising_edge(Clk)) THEN
         CASE OP IS

            WHEN "0000000000000001" =>
               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 "0000000000000010" =>

            Result <= Reg1 SLL 3;

         WHEN "0000000000000100" =>
            Result(7) <= NOT Reg2(7);
            Result(6) <= NOT Reg2(6);
            Result(5) <= NOT Reg2(5);
            Result(4) <= NOT Reg2(4);
            Result(3) <= Reg2(3);
            Result(2) <= Reg2(2);
            Result(1) <= Reg2(1);
            Result(0) <= Reg2(0);


         WHEN "0000000000001000" =>
            IF (Reg1 <= Reg2) THEN
               Result <= Reg1;
            ELSE
               Result <= Reg2;
            END IF;

         WHEN "0000000000010000" => Result <= (Reg1 + Reg2) + 4;

         WHEN "0000000000100000" => Result <= Reg1 + "00000011";

         WHEN "0000000001000000" =>
            Result(0) <= Reg1(0);
            Result(1) <= Reg2(1);
```

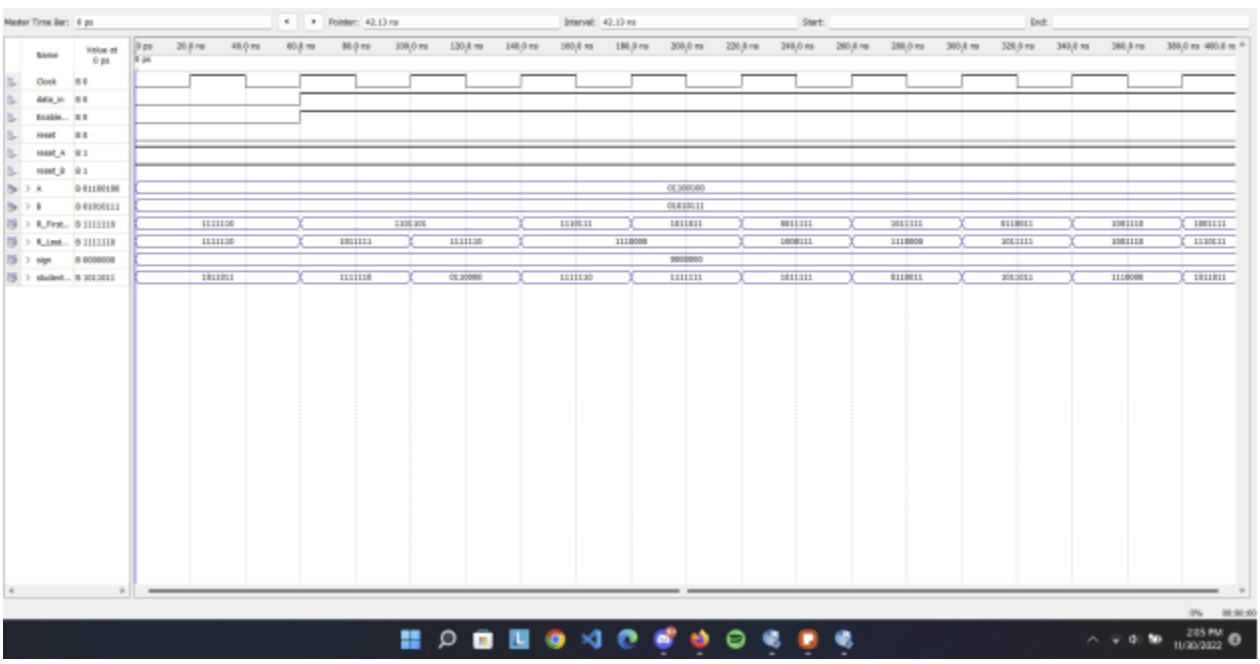
```
            Result(2) <= Reg1(2);
            Result(3) <= Reg2(3);
            Result(4) <= Reg1(4);
            Result(5) <= Reg2(5);
            Result(6) <= Reg1(6);
            Result(7) <= Reg2(7);
            Neg <= '0';

         WHEN "0000000010000000" => Result <= (Reg1 XNOR Reg2);

         WHEN "0000000100000000" => Result <= Reg2 ROR 3;

         WHEN OTHERS =>
            Result <= "--------";


         END CASE;
      END IF;
   END PROCESS;

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

```
ase minimum pulse width slack is -1.380
ected device family is not supported by the report metastability command.
```

This figure represents an ALU. This is where all the computation occurs and where we enter the input for each case and get the output once the system is connected. This means this is the portion of a central processing unit that performs arithmetic and logic operations for problem 2.



This figure represents the Block Diagram for problem 2, where we connect the components together to get the desired output.



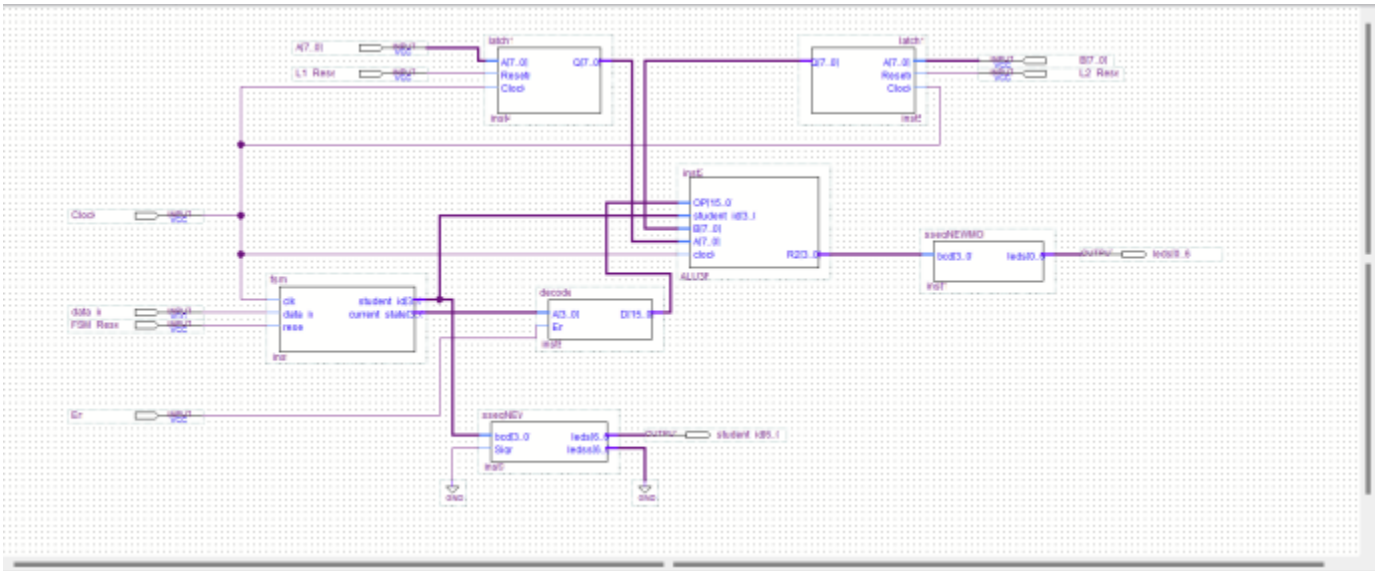
This figure represents the waveform for the block diagram.

Part 3

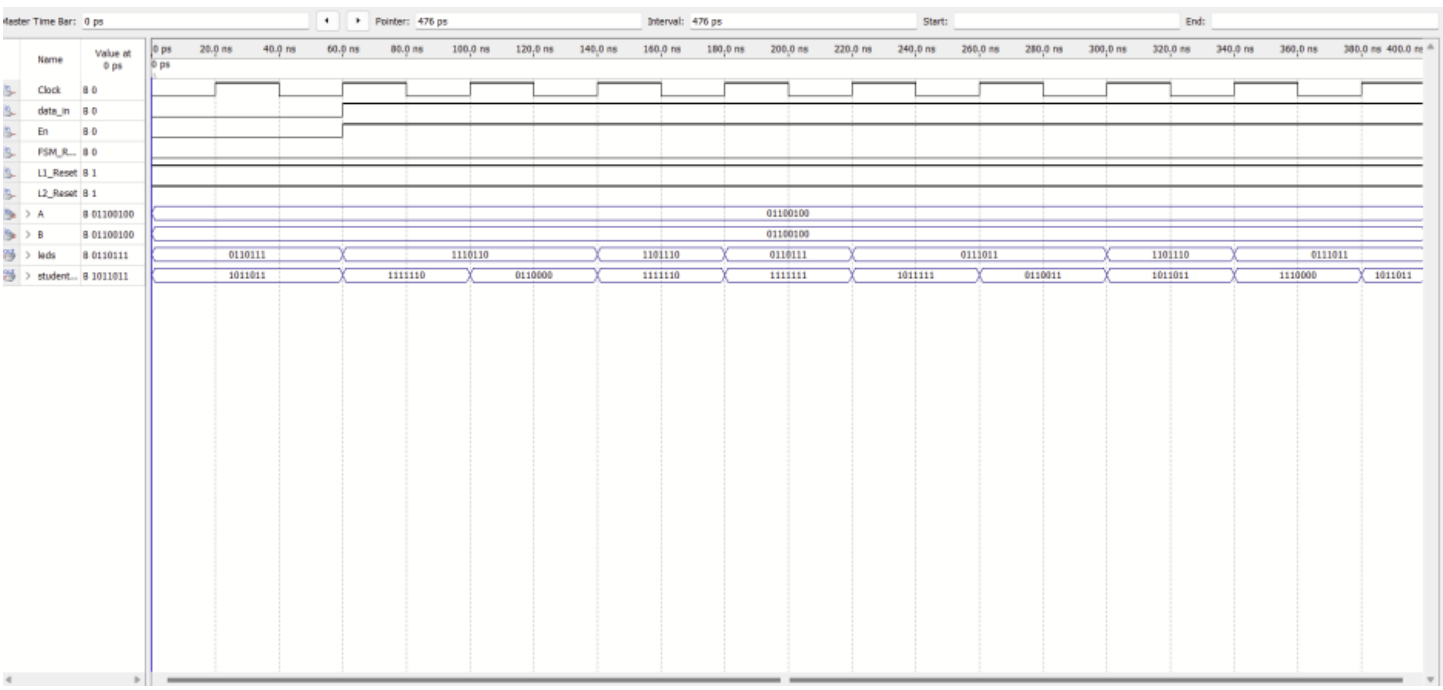
The student ID supplied by the decoder will also have an effect on the output of this ALU. In this case, function F was given problem set 3. Meaning, the ALU must specifically verify if the latch output (A) is less than the student id digit, and then output "0001" if true and "0000" if false. In addition, a modified seven-segment display is implemented, which displays the characters Y and N.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;

ENTITY ALU3F 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);
      R2 : OUT unsigned(3 DOWNTO 0)
   );
END ALU3F;

ARCHITECTURE aluCalc OF ALU3F IS
   SIGNAL reg1, result : unsigned(7 DOWNTO 0) := (OTHERS => '0');
   SIGNAL reg2, reg3 : unsigned(0 TO 7);
BEGIN
   reg1 <= A;
   reg2 <= Reg1 REM 100; -- this is the digit in the ones place
   reg3 <= ((Reg1 REM 100)/10); -- this is the digit in the tens place
   PROCESS (clock, OP)
   BEGIN
      IF (rising_edge(clock)) THEN

         --In this case "00000001"=Yes, and "00000000"=No
         CASE OP IS
            WHEN "0000000000000001" =>
               IF (reg2 < student_id OR reg3 < student_id) THEN
                  result <= "00000001";
               ELSE
                  result <= "00000000";
               END IF;

            WHEN "0000000000000010" =>
               IF (reg2 < student_id OR reg3 < student_id) THEN
                  result <= "00000001";
               ELSE
                  result <= "00000000";
               END IF;

            WHEN "0000000000000100" =>
               IF (reg2 < student_id OR reg3 < student_id) THEN
                  result <= "00000001";
               ELSE
                  result <= "00000000";
               END IF;

            WHEN "0000000000001000" =>
               IF (reg2 < student_id OR reg3 < student_id) THEN
                  result <= "00000001";
               ELSE
                  result <= "00000000";
               END IF;

            WHEN "0000000000010000" =>
               IF (reg2 < student_id OR reg3 < student_id) THEN
                  result <= "00000001";
               ELSE
                  result <= "00000000";
               END IF;

            WHEN "0000000000100000" =>
               IF (reg2 < student_id OR reg3 < student_id) THEN
                  result <= "00000001";
               ELSE
                  result <= "00000000";
               END IF;

            WHEN "0000000001000000" =>
               IF (reg2 < student_id OR reg3 < student_id) THEN
                  result <= "00000001";
               ELSE
                  result <= "00000000";
               END IF;

            WHEN "0000000010000000" =>
               IF (reg2 < student_id OR reg3 < student_id) THEN
                  result <= "00000001";
               ELSE
                  result <= "00000000";
               END IF;

            WHEN "0000000100000000" =>
               IF (reg2 < student_id OR reg3 < student_id) THEN
                  result <= "00000001";
               ELSE
                  result <= "00000000";
               END IF;

            WHEN OTHERS =>
               result <= "00000000";
         END CASE;
      END IF;
   END PROCESS;
   R2 <= result(3 DOWNTO 0);
END aluCalc;
```

This figure represents an ALU. This is where all the computation occurs and where we enter the input for each case and get the output once the system is connected. This means this is the portion of a central processing unit that performs arithmetic and logic operations for problem 3.



This figure represents the Block Diagram for problems 1 to 6, where we connect the components together to get the desired output.



This figure represents the waveform for the block diagram.

Conclusion

In conclusion, this lab experiment broadened my understanding of general processor fundamentals and the theory behind machines. The generic processor is a device that performs certain operations using two binary values supplied via latches. The processor for the lab switches between 9 states from the finite state machine, allowing different tasks to be done. A decoder is also used to transform the finite state machine's 4-bit output to a 16-bit output for the ALU to understand. A computer has an ALU built inside to perform necessary calculations.

The difficulty faced in this face was messing up connecting the block diagram for problem 1, therefore, re-making the block diagram with problem configuration solved the problem and produced the correct output.

References

Agarwal, T. (2019, October 15). How to design a 4 to 16 decoder using 3 to 8 decoder. ElProCus. Retrieved December 1, 2022, from
https://www.elprocus.com/designing-4-to-16-decoder-using-3-to-8-decoder/