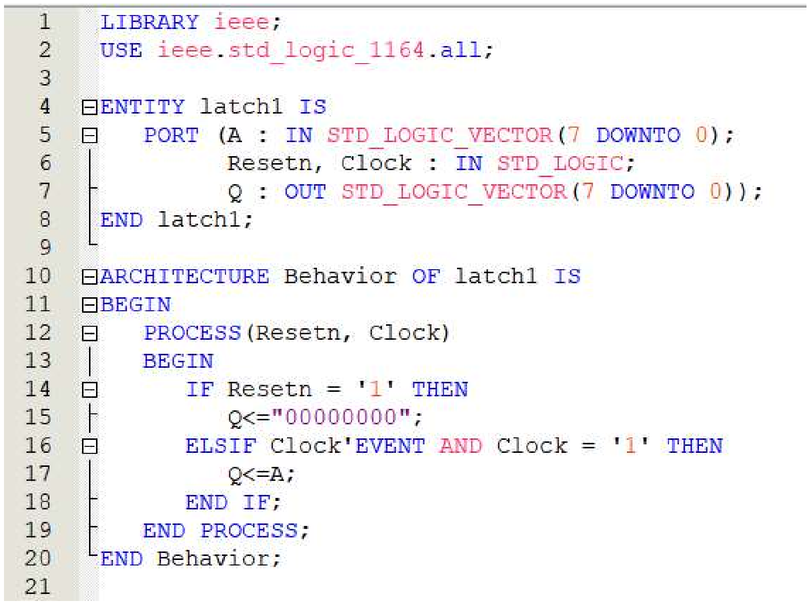
Table of Contents:

Introduction:

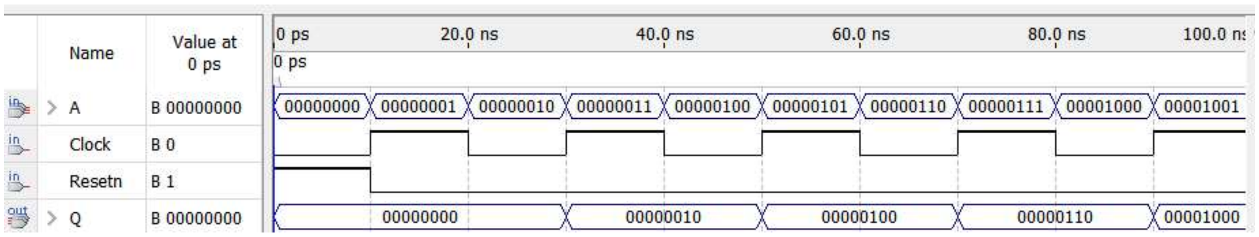
The purpose of this lab was to create a simple Arithmetic and Logic Unit (ALU) and gain a better understanding of how it works and how it functions. To make this ALU we had to construct other components as well including Latches, 4-16 Decoder and Finite State Machine (FSM). We also had to alter the ALU block 3 times to test it's functionality and therefore we had to make 3 different ALU devices. This was all done through VHDL code.

Latch:

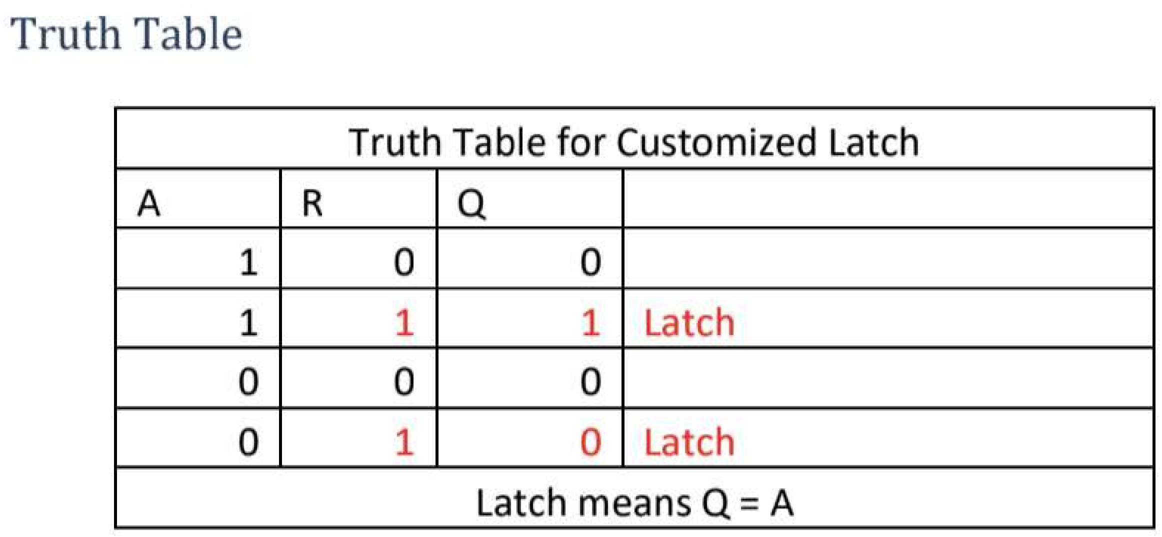
A latch is a memory storage device and output said input when it is turned on. Whether it is on or not is determined by what value the Reset asks for, either 1 or 0, if it's off then nothing will be outputted but if on then the memory that was inputted in it should be outputted and the purpose it served in this lab was to output the two numbers that the calculations would be carried out on in the ALU. Below the VHDL code for the Latch can be seen along with the corresponding waveform:



```
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 = '1' THEN
            Q<="00000000";
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q<=A;
        END IF;
    END PROCESS;
END Behavior;

```



Latch Truth Table:



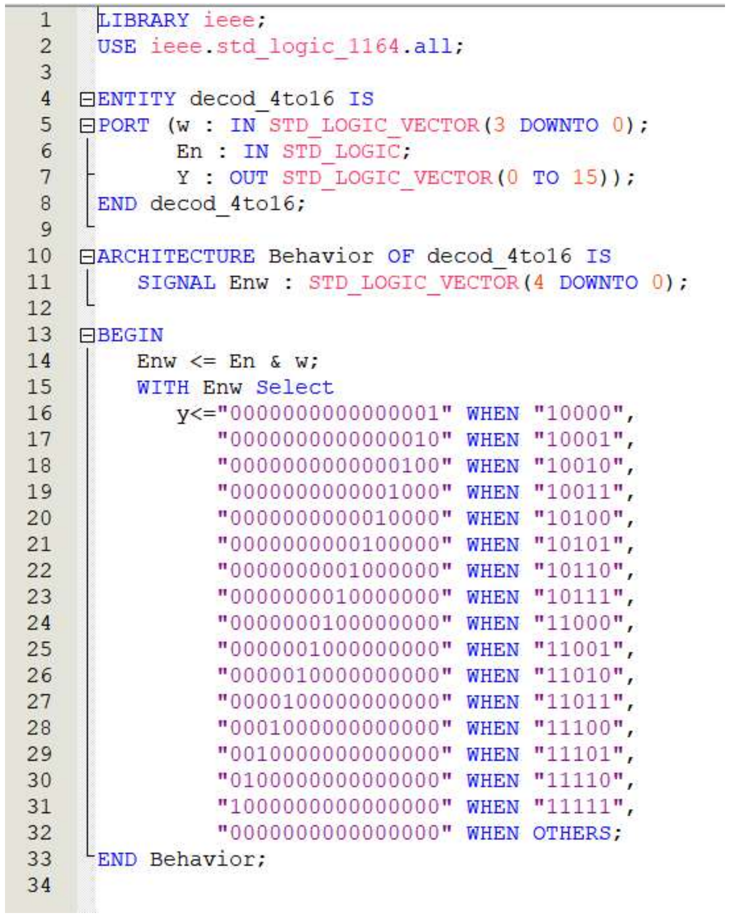
Truth Table

Truth Table for Customized Latch			
A	R	Q	
1	0	0	
1	1	1	Latch
0	0	0	
0	1	0	Latch
Latch means Q = A			

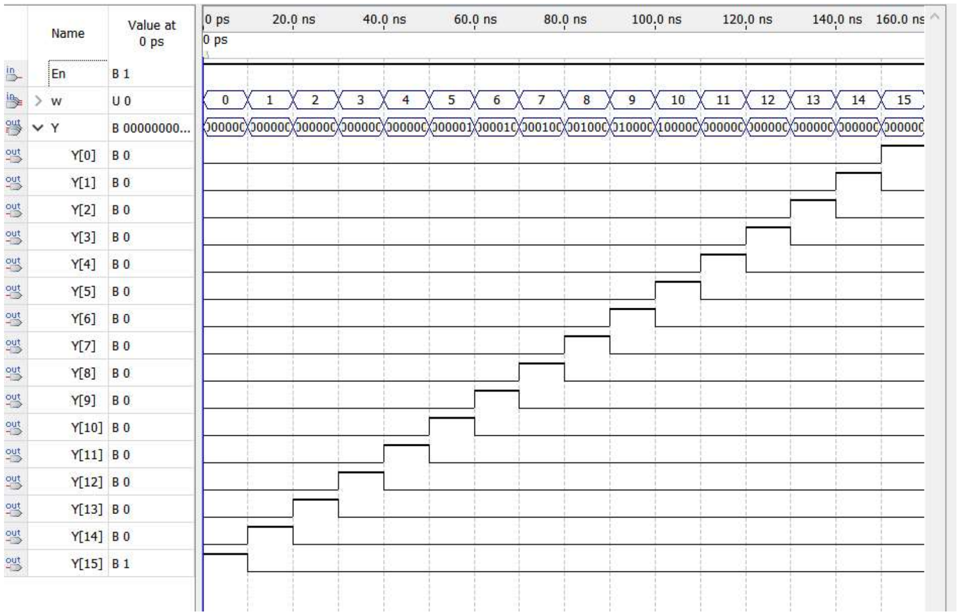
Above is an example of a latch truth table and as you can see from the truth table the waveform that was given by the VHDL implementation of the latch is correct, as when Reset is on there is a delay that results in the output being one after, which is present in the waveform and when when Reset is off the outputs are 0 which is present in our waveform as well.

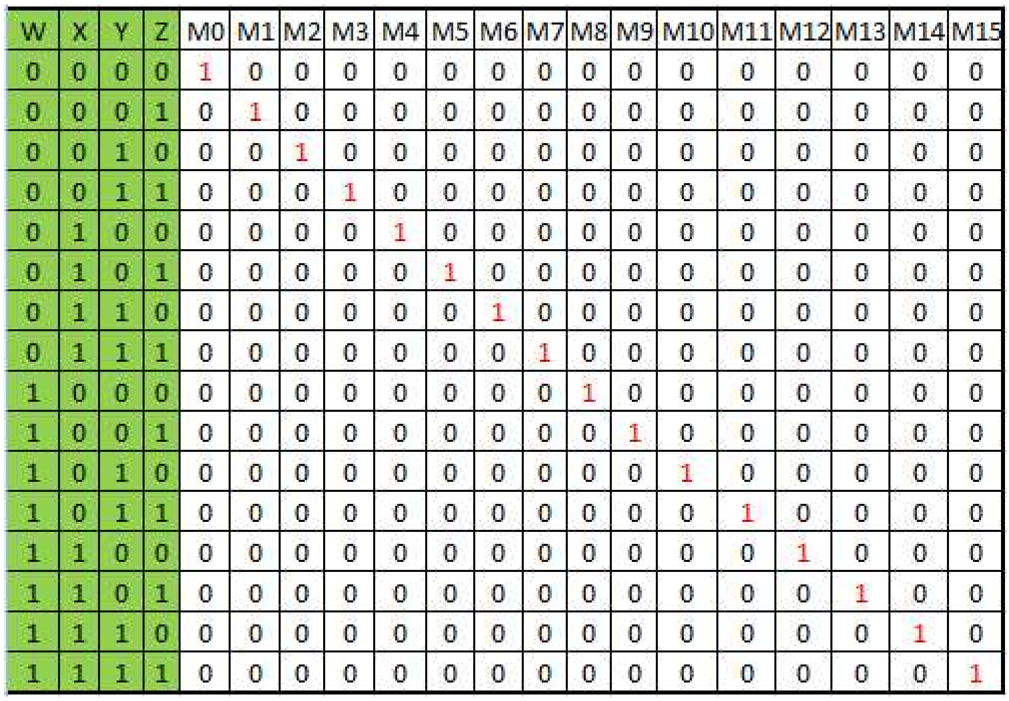
4:16 Decoder:

The purpose of the decoder was to take the 4 bit input of the states from the FSM and change it to 16 bit output that will be used as the Microcode for the ALU. To do this we simply altered our decoder code from Lab 4 and made it into a 4-16 decoder. The VHDL code, Waveform and Truth Table can be seen below:



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

ENTITY decod_4to16 IS
PORT (w : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      En : IN STD_LOGIC;
      Y : OUT STD_LOGIC_VECTOR(0 TO 15));
END decod_4to16;

ARCHITECTURE Behavior OF decod_4to16 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 Behavior;

```





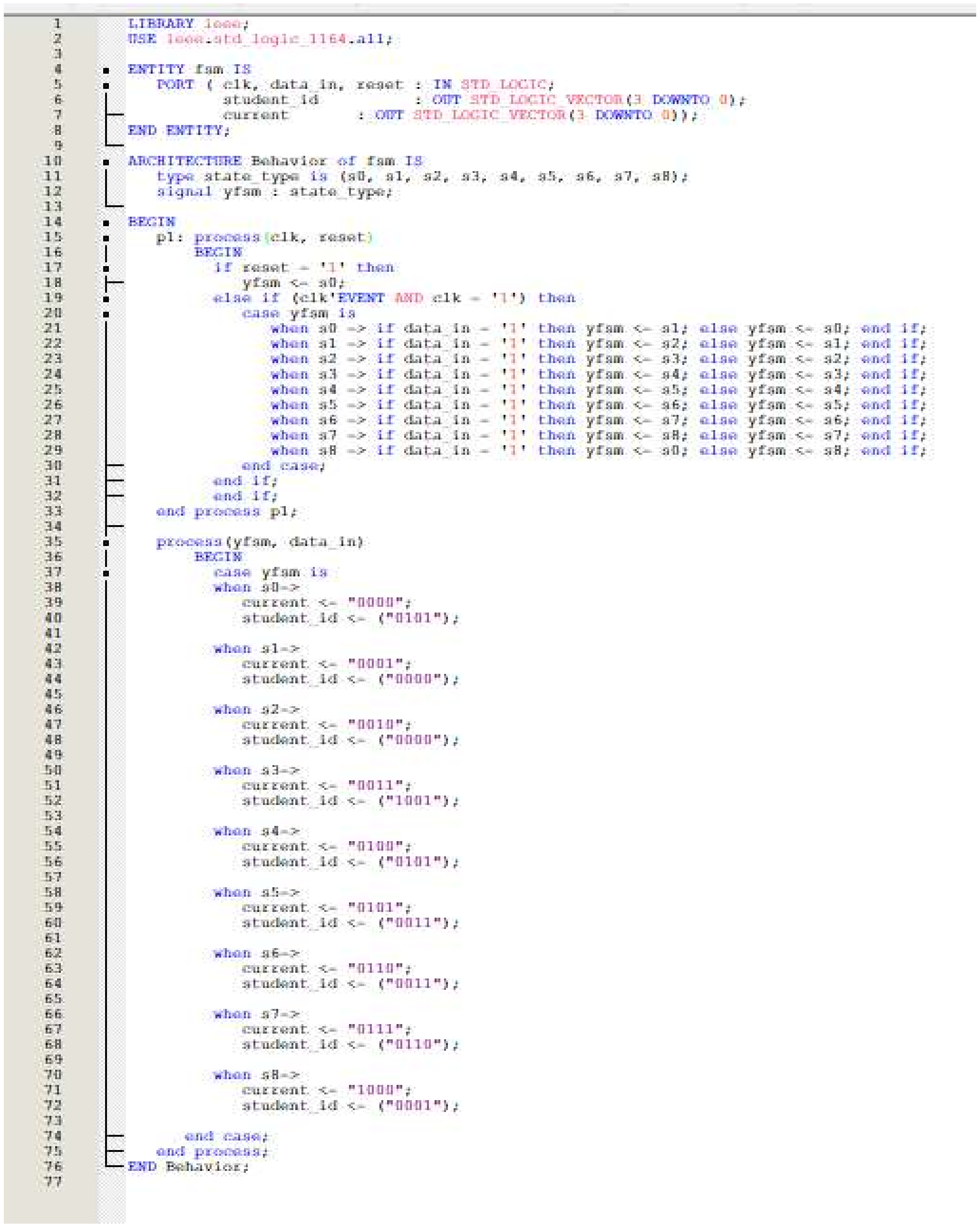
W	X	Y	Z	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	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	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

One thing that should be noted is that the waveform output for our decoder is flipped from that of the normal truth table, this was done because the microcodes start from bottom up rather than top down so by flipping the truth table outputs we were able to input the proper microcode into the ALU to make sure it functions properly.

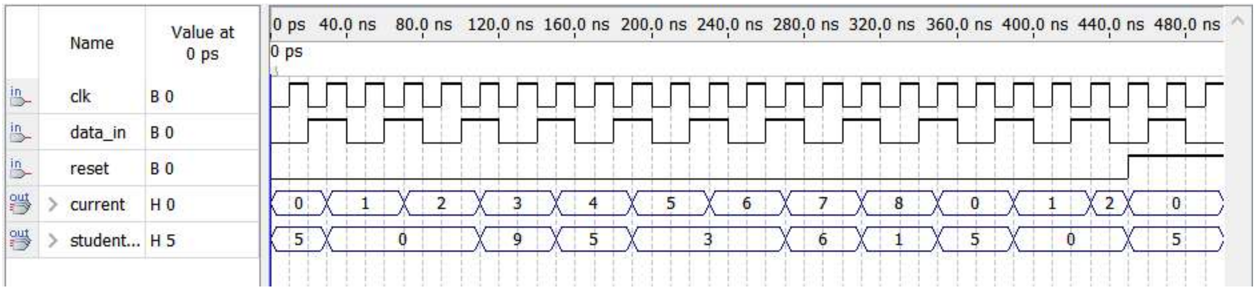
FSM:

The purpose of the FSM was to output our student number and also the states, the student number would be used in the third ALU problem whereas the states were the inputs for the 4-16 decoder. The FSM functioned the same way as the FSM in the previous lab with some differences. The first was that in the first lab we implemented a Mealy machine whereas in this case it's a Moore machine and the second difference is that the order of the states is a simple sequence from 0-8. The VHDL code, State Table and waveform are below:

Present State (Sn)	**Next State**		**Output(Number)**	**Output(State)**
	Data IN W=0	**Data IN W=1**	**Student ID**	**Current State**
S0	S0	S1	[5] d1=0101	0000 [0]
S2	S1	S2	[0] d2=0000	0001 [1]
S4	S2	S3	[0] d3=0000	0010 [2]
S1	S3	S4	[9] d4=1001	0011 [3]
S3	S4	S5	[5] d5=0101	0100 [4]
S8	S5	S6	[3] d6=0011	0101 [5]
S6	S6	S7	[3] d7=0011	0110 [6]
S5	S7	S8	[6] d8=0110	0111 [7]
S7	S8	S0	[1] d9=0001	1000 [8]

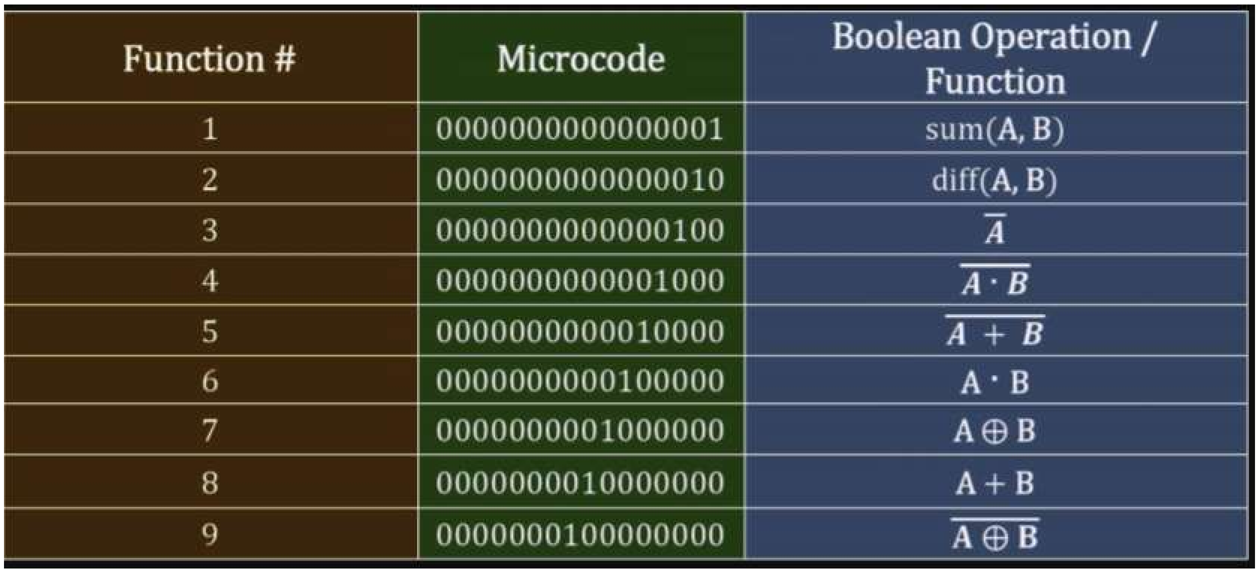


```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fsm IS
    PORT ( clk, data_in, reset : IN STD_LOGIC;
           student_id         : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
           current        : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END ENTITY;

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

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

    process(yfsm, data_in)
        BEGIN
          case yfsm is
          when s0=>
             current <= "0000";
             student_id <= ("0101");

          when s1=>
             current <= "0001";
             student_id <= ("0000");

          when s2=>
             current <= "0010";
             student_id <= ("0000");

          when s3=>
             current <= "0011";
             student_id <= ("1001");

          when s4=>
             current <= "0100";
             student_id <= ("0101");

          when s5=>
             current <= "0101";
             student_id <= ("0011");

          when s6=>
             current <= "0110";
             student_id <= ("0011");

          when s7=>
             current <= "0111";
             student_id <= ("0110");

          when s8=>
             current <= "1000";
             student_id <= ("0001");

        end case;
    end process;
END Behavior;
```



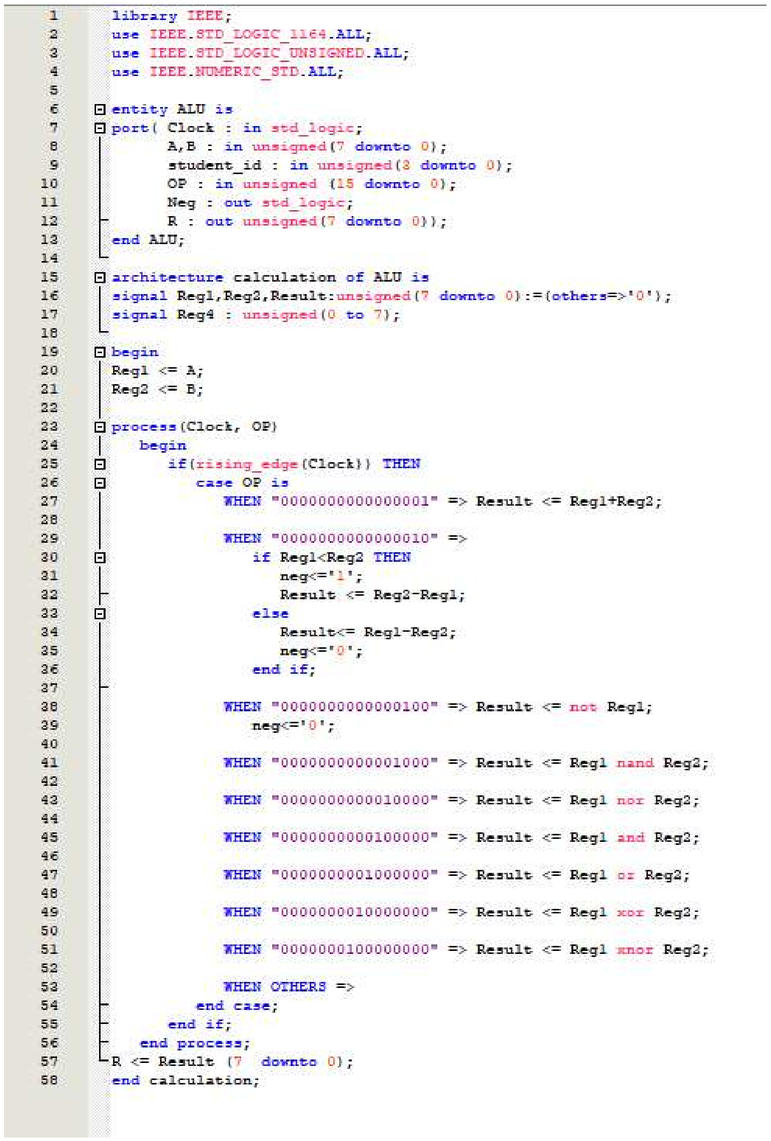
ALU:

The Arithmetic and Logic Unit or ALU is used commonly to carry out multiple calculations and functions on a pair of two inputs depending on the microcode. Depending on what the microcode is it will perform various functions as depicted by the table below:

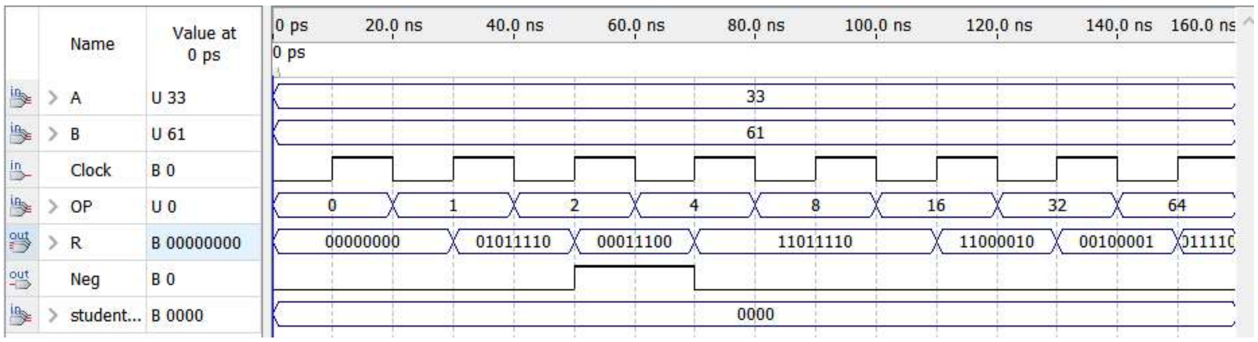


Function #	Microcode	Boolean Operation / Function
1	0000000000000001	sum(A, B)
2	0000000000000010	diff(A, B)
3	0000000000000100	$\overline{A}$
4	0000000000001000	$\overline{A \cdot B}$
5	0000000000010000	$\overline{A + B}$
6	0000000000100000	$\mathrm{A} \cdot \mathrm{B}$
7	0000000001000000	$\mathrm{A} \oplus \mathrm{B}$
8	0000000010000000	$\mathrm{A} + \mathrm{B}$
9	0000000100000000	$\overline{\mathrm{A} \oplus \mathrm{B}}$

The VHDL code along with the corresponding waveform looks as follows:



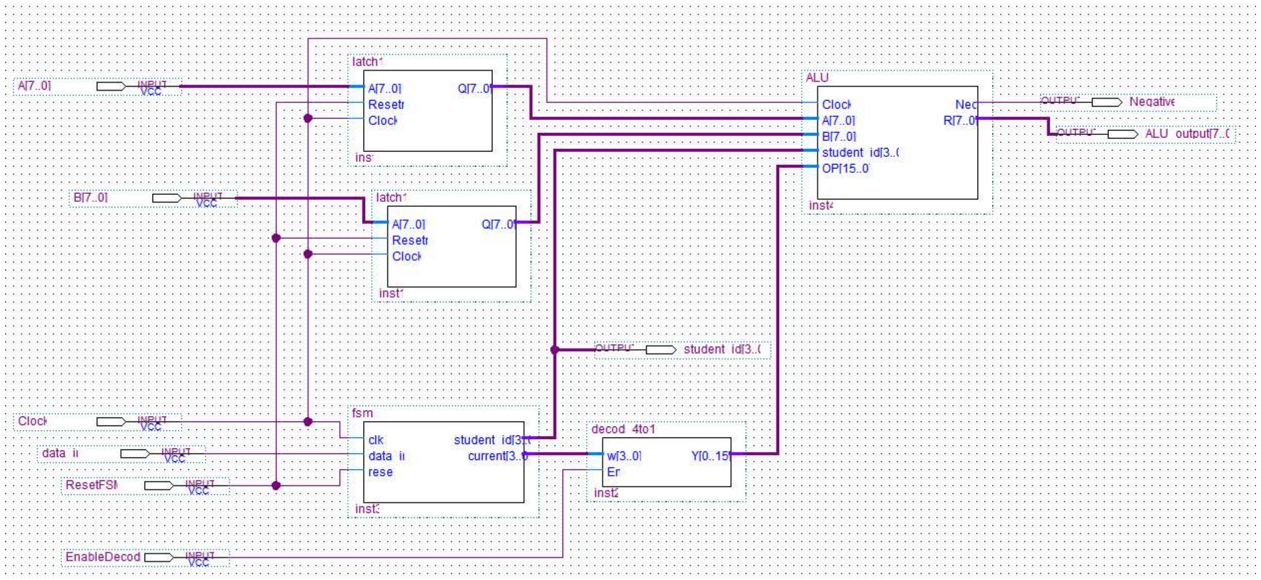
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ALU is
port( Clock : in std_logic;
      A,B : in unsigned(7 downto 0);
      student_id : in unsigned(3 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;

            WHEN "0000000000000010" =>
               if Reg1<Reg2 THEN
                  neg<='1';
                  Result <= Reg2-Reg1;
               else
                  Result<= Reg1-Reg2;
                  neg<='0';
               end if;

            WHEN "0000000000000100" => Result <= not Reg1;
               neg<='0';

            WHEN "0000000000001000" => Result <= Reg1 nand Reg2;

            WHEN "0000000000010000" => Result <= Reg1 nor Reg2;

            WHEN "0000000000100000" => Result <= Reg1 and Reg2;

            WHEN "0000000001000000" => Result <= Reg1 or Reg2;

            WHEN "0000000010000000" => Result <= Reg1 xor Reg2;

            WHEN "0000000100000000" => Result <= Reg1 xnor Reg2;

            WHEN OTHERS =>
         end case;
      end if;
   end process;
R <= Result (7  downto 0);
end calculation;
```

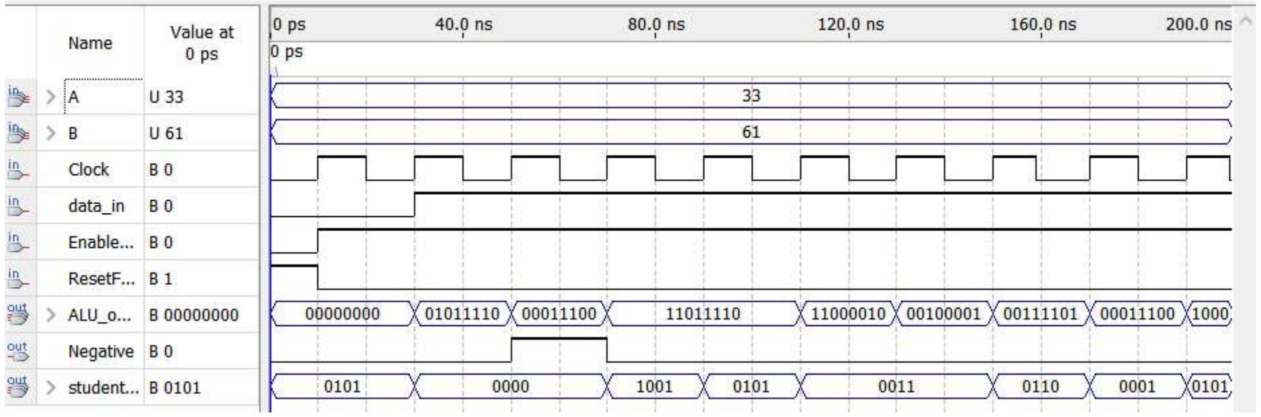


The waveform above was checked by hand by me and through various online calculators for functions 4-9 and they were all correct. On top of that the negative output is also functional as it's high when subtraction is done and low in every other case. The first output is also a 0 because of the delay from the latch and the data_in inputs.

<u>Part 1:</u>

This is a combination of all previous components we've made into a massive circuit that takes inputs. The inputs for the ALU to function are the values of A and B from the latches and the output from the 4-16 decoder for the microcodes and the decoder itself takes inputs from the FSM to function, and finally the student_id input would be determined by the student number output in the ALU. The block schematic for the first problem looked as follows along with the corresponding waveform:



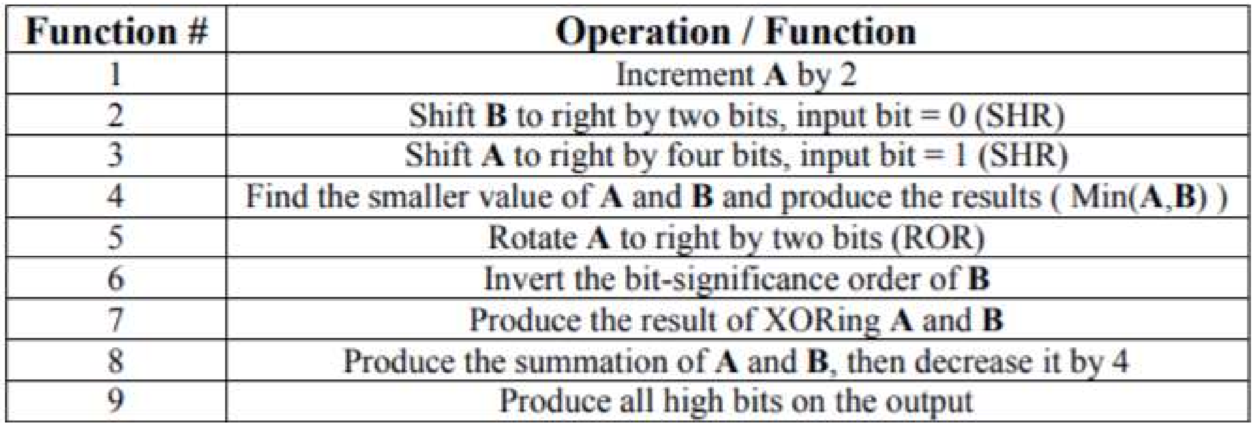


Name	Value at 0 ps
A	U 33
B	U 61
Clock	B 0
data_in	B 0
Enable...	B 0
ResetF...	B 1
ALU_o...	B 00000000
Negative	B 0
student...	B 0101

We can be sure that the waveform for part 1 is fine as it matches with the normal ALU waveform.

Part 2:

For the second part or the second problem we had to modify the ALU VHDL code to match what the problem we were assigned asked for. In my case it was problem a which asked for the following:

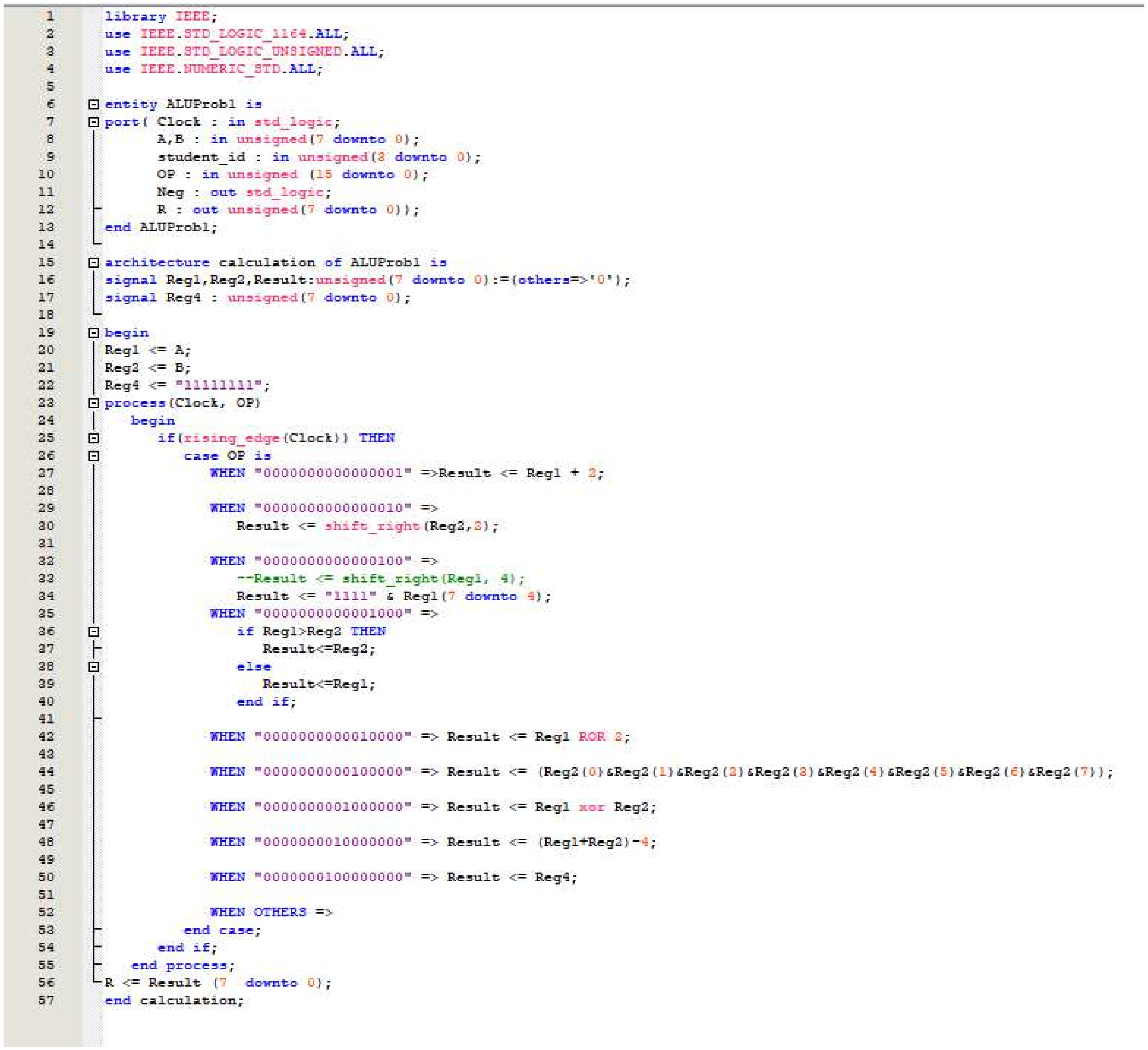


Function #	Operation / Function
1	Increment **A** by 2
2	Shift **B** to right by two bits, input bit = 0 (SHR)
3	Shift **A** to right by four bits, input bit = 1 (SHR)
4	Find the smaller value of **A** and **B** and produce the results (Min(**A**,**B**))
5	Rotate **A** to right by two bits (ROR)
6	Invert the bit-significance order of **B**
7	Produce the result of XORing **A** and **B**
8	Produce the summation of **A** and **B**, then decrease it by 4
9	Produce all high bits on the output

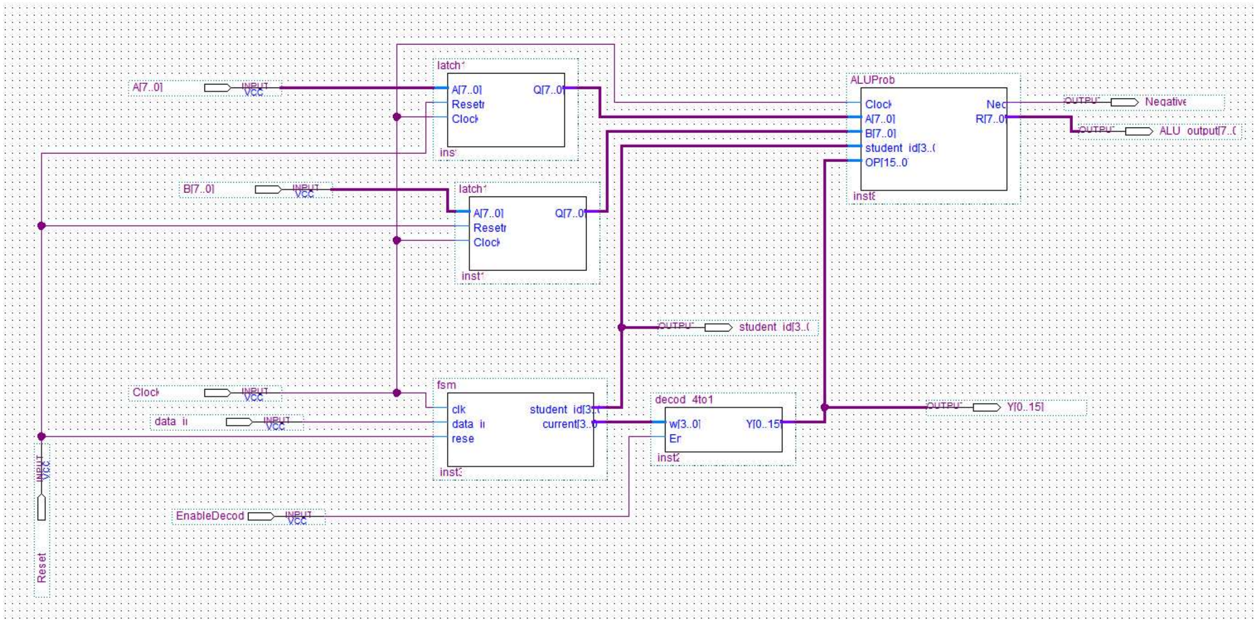
Some of the things asked for are self explanatory such as function 1, 4, 7 and 8 while the rest are a bit more complicated. For function 2 and 3 they ask us to shift a bit by number by a certain number of bits, for example the number 1011 shifted right by 2 would be 0010 and if the input bit is 1 then it would be 1110, there is a built in function for quartus to do this and that was used for this problem and also a built in function was used for problem 5 as well that asks us to rotate A by two bits. What rotating a bit means simply changing the position of the bit with it's adjacent bit.

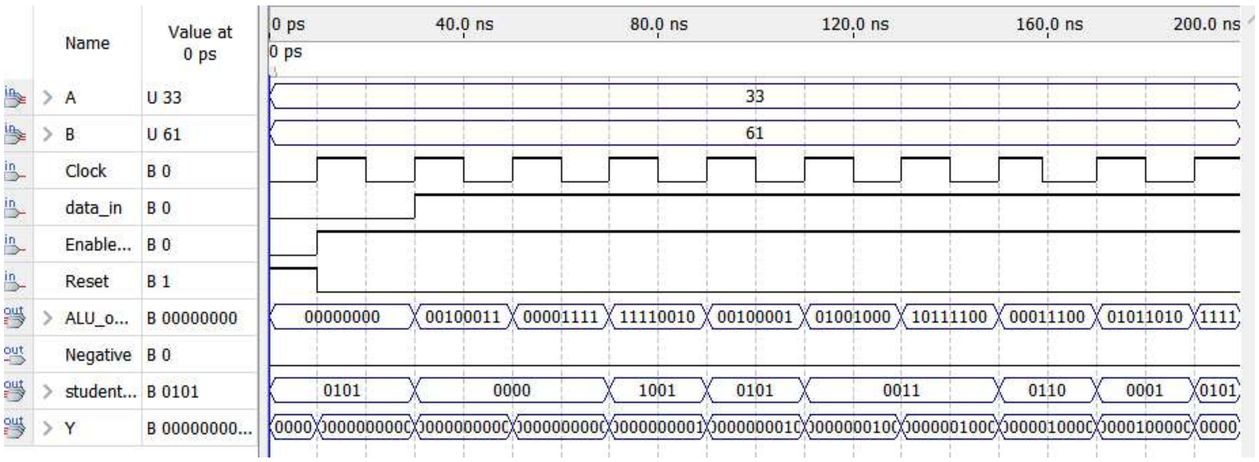
For function 6 it asks to invert bit significance and that simply means to reverse a bit so if it's 1011 then it would become 1101 and so on and finally for function 9 it asks to produce all high bits and that just means to output only 1s.

The altered ALU VHDL along with the circuit and waveform looked as follows:



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

entity ALUProb1 is
port( Clock : in std_logic;
      A,B : in unsigned(7 downto 0);
      student_id : in unsigned(3 downto 0);
      OP : in unsigned (15 downto 0);
      Neg : out std_logic;
      R : out unsigned(7 downto 0));
end ALUProb1;

architecture calculation of ALUProb1 is
signal Reg1,Reg2,Result:unsigned(7 downto 0):=(others=>'0');
signal Reg4 : unsigned(7 downto 0);

begin
Reg1 <= A;
Reg2 <= B;
Reg4 <= "11111111";
process(Clock, OP)
   begin
      if(rising_edge(Clock)) THEN
         case OP is
            WHEN "0000000000000001" =>Result <= Reg1 + 2;

            WHEN "0000000000000010" =>
               Result <= shift_right(Reg2,3);

            WHEN "0000000000000100" =>
               --Result <= shift_right(Reg1, 4);
               Result <= "1111" & Reg1(7 downto 4);
            WHEN "0000000000001000" =>
               if Reg1>Reg2 THEN
                  Result<=Reg2;
               else
                  Result<=Reg1;
               end if;

            WHEN "0000000000010000" => Result <= Reg1 ROR 2;

            WHEN "0000000000100000" => Result <= (Reg2(0)&Reg2(1)&Reg2(2)&Reg2(3)&Reg2(4)&Reg2(5)&Reg2(6)&Reg2(7));

            WHEN "0000000001000000" => Result <= Reg1 xor Reg2;

            WHEN "0000000010000000" => Result <= (Reg1+Reg2)-4;

            WHEN "0000000100000000" => Result <= Reg4;

            WHEN OTHERS =>
         end case;
      end if;
   end process;
R <= Result (7  downto 0);
end calculation;
```



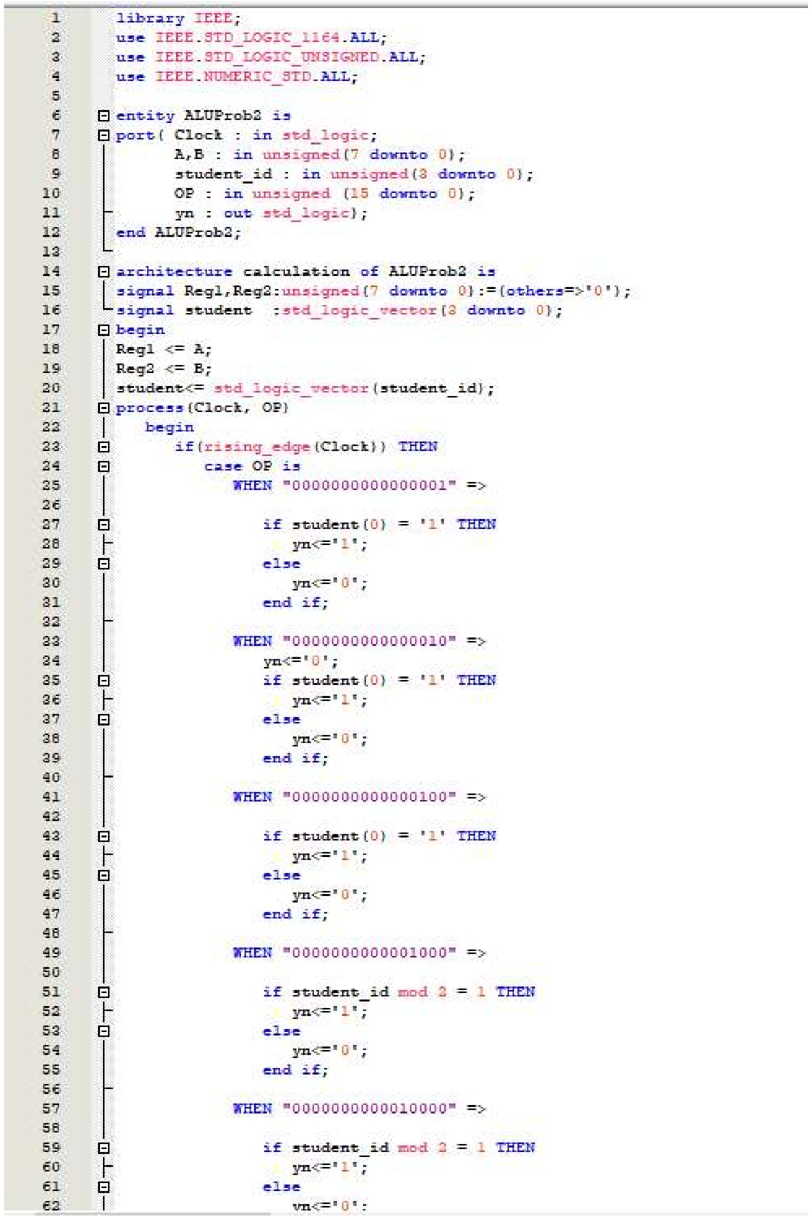


The waveform was checked by hand and online calculators and everything is correctly done.

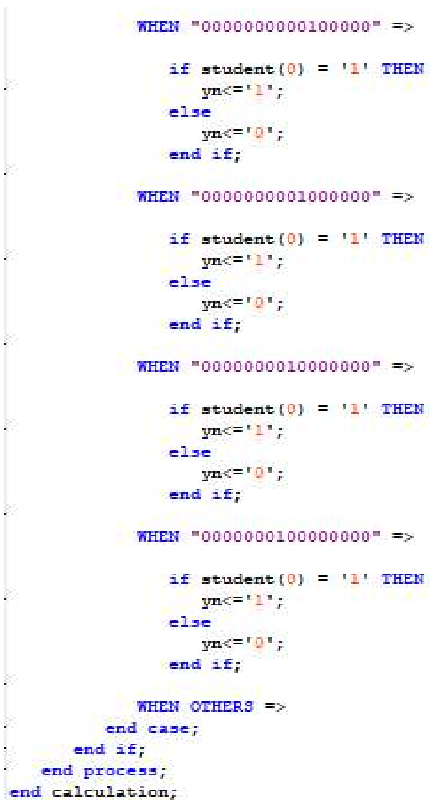
<u>Part 3:</u>

Part 3 or problem 3 is similar to the first 2 but in this one the ALU is modified more so than part 2. The modifications in this part are related to inputs, outputs and the functionality. For the inputs, A and B are no longer the main focus; it's instead student numbers input from the FSM and for the output there are no longer calculations being done but rather a check for odd or even for the bits in the student number. Finally the functionality of the ALU is changed similarly to how it was changed in part 2.

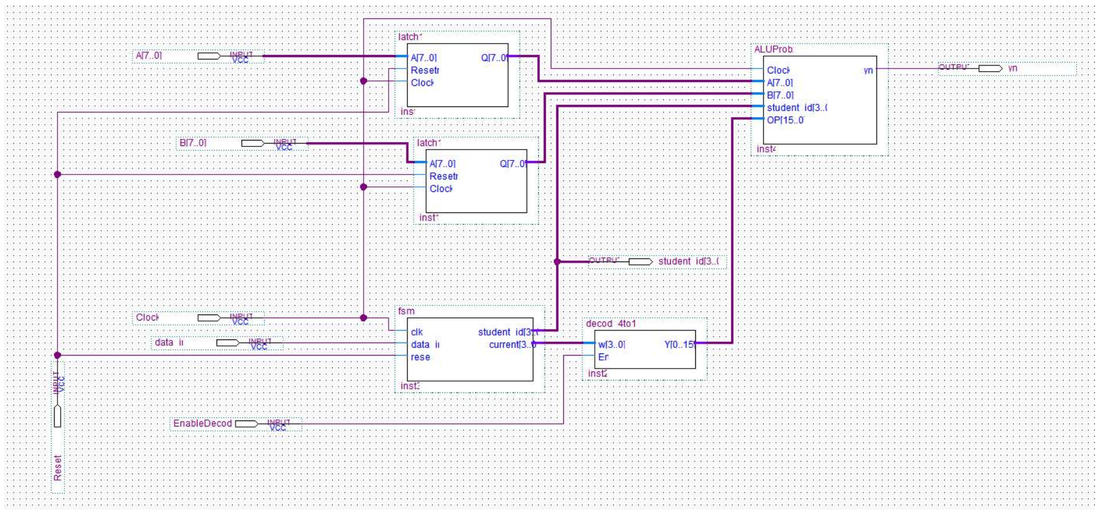
The final ALU VHDL along with the corresponding waveform and circuit are below:

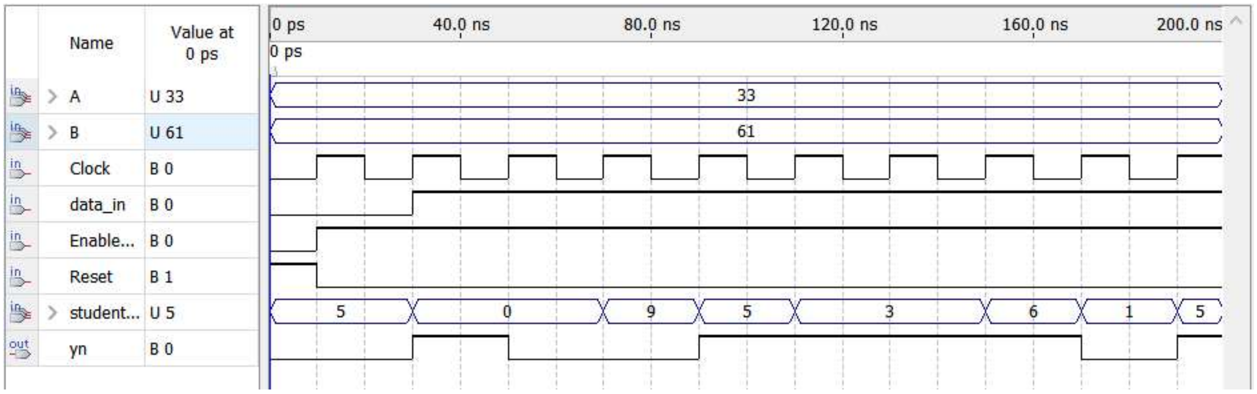


```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ALUProb2 is
port( Clock : in std_logic;
      A,B : in unsigned(7 downto 0);
      student_id : in unsigned(3 downto 0);
      OP : in unsigned (15 downto 0);
      yn : out std_logic);
end ALUProb2;

architecture calculation of ALUProb2 is
signal Reg1,Reg2:unsigned(7 downto 0):=(others=>'0');
signal student  :std_logic_vector(3 downto 0);
begin
Reg1 <= A;
Reg2 <= B;
student<= std_logic_vector(student_id);
process(Clock, OP)
   begin
      if(rising_edge(Clock)) THEN
         case OP is
            WHEN "0000000000000001" =>

               if student(0) = '1' THEN
                  yn<='1';
               else
                  yn<='0';
               end if;

            WHEN "0000000000000010" =>
               yn<='0';
               if student(0) = '1' THEN
                  yn<='1';
               else
                  yn<='0';
               end if;

            WHEN "0000000000000100" =>

               if student(0) = '1' THEN
                  yn<='1';
               else
                  yn<='0';
               end if;

            WHEN "0000000000001000" =>

               if student_id mod 2 = 1 THEN
                  yn<='1';
               else
                  yn<='0';
               end if;

            WHEN "0000000000010000" =>

               if student_id mod 2 = 1 THEN
                  yn<='1';
               else
                  yn<='0';
```



```
            WHEN "0000000000100000" =>

                if student(0) = '1' THEN
                    yn<='1';
                else
                    yn<='0';
                end if;

            WHEN "0000000001000000" =>

                if student(0) = '1' THEN
                    yn<='1';
                else
                    yn<='0';
                end if;

            WHEN "0000000010000000" =>

                if student(0) = '1' THEN
                    yn<='1';
                else
                    yn<='0';
                end if;

            WHEN "0000000100000000" =>

                if student(0) = '1' THEN
                    yn<='1';
                else
                    yn<='0';
                end if;

            WHEN OTHERS =>
        end case;
    end if;
  end process;
end calculation;
```





The waveform is supposed to output a high at YN when the student number is odd and at first glance the output is not correct but in reality it's just shifted to the right due to the inherent delay which means the output and waveform is in fact correct.

Conclusion:

The lab helped give a better understanding of how ALU functions and also the various components needed to make it function, we learned more of how an ALU works in relation to the microcodes. This lab also helped clarify more concepts from previous labs as it was an amalgamation of everything we had done before along with newer concepts of this lab. This lab also showed how various different components come together to make something much larger through the implementation of our block diagram and also how these different components affect each other in various ways.