Ryerson University Faculty of Science

Course Number	COE 328
Course Title	Digital Circuits
Semester/Year	Fall 2021
Instructor	Dr. Reza Sedaghat
TA Name	Sajjad Rostami Sani

Lab/Tutorial Report No.	6

Report Title	Simple General-Purpose Processor

Section No.	03
Submission Date	19-Dec-2021
Due Date	20-Dec-2021

Student Name	Student ID	Signature*
Ahmad El-Gohary	501011852	Ahmad El-Gohary

*By signing above, you attest that you have contributed to this submission and confirm that all work you have contributed to this submission is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a "0" on the work, an "F" in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at:

http://www.ryerson.ca/content/dam/senate/policies/pol60.pdf

Contents

Introduction

This lab was to design and construct a simple microprocessor that does an operation to 2 binary numbers based on the state of the Finite State Machine.

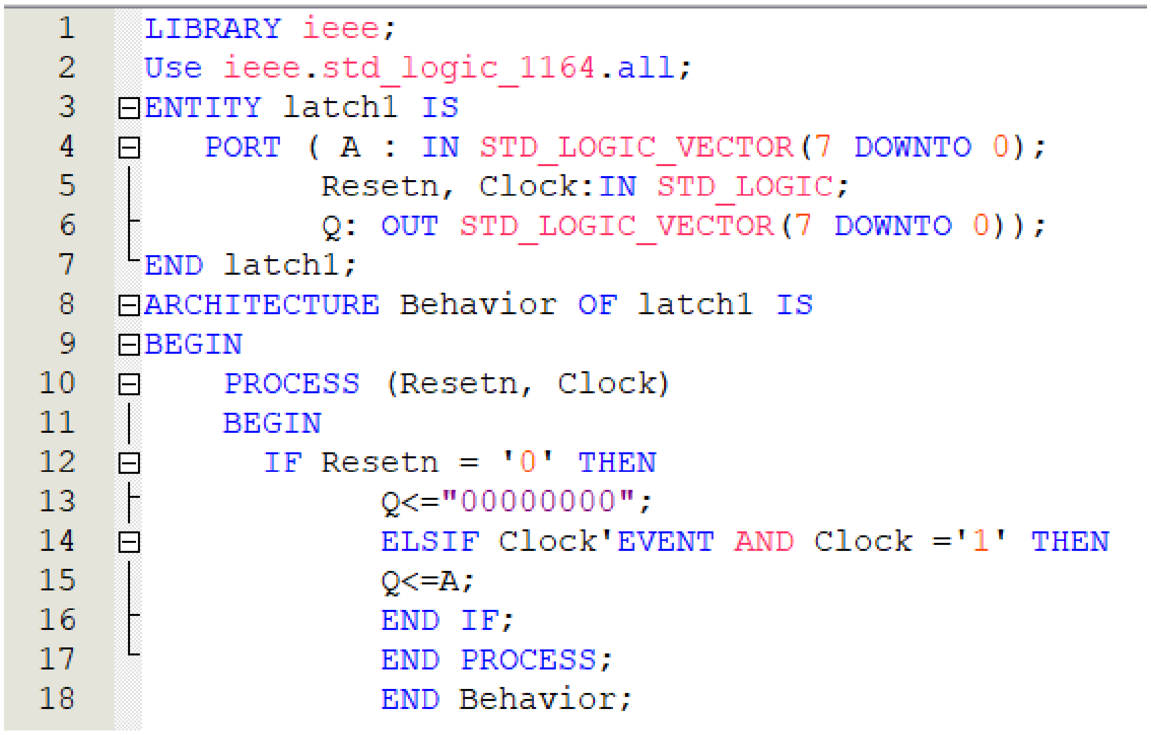
Components

- Basic latch

  The basic latch is the main component of the storage unit which takes in a binary number as an input and gives it out as an output for each cycle.

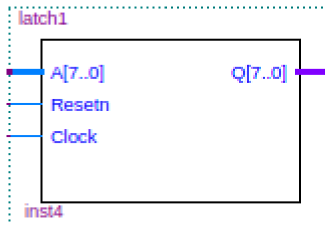
  The circuit uses 2 basic latches 1 for each binary number.

  Code for latch:

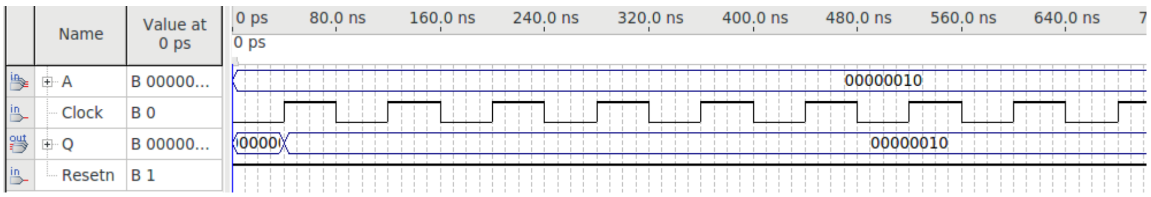


```
LIBRARY ieee;
Use ieee.std_logic_1164.all;
ENTITY latch1 IS
   PORT ( A : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         Resetn, Clock:IN STD_LOGIC;
         Q: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END latch1;
ARCHITECTURE Behavior OF latch1 IS
BEGIN
    PROCESS (Resetn, Clock)
    BEGIN
      IF Resetn = '0' THEN
            Q<="00000000";
            ELSIF Clock'EVENT AND Clock ='1' THEN
            Q<=A;
            END IF;
            END PROCESS;
            END Behavior;
```

Component’s Circuit Diagram:

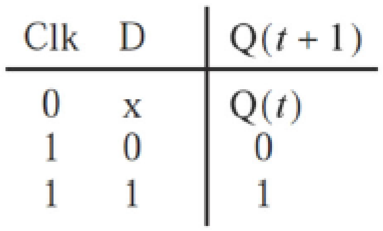


Waveform for Latch:



Name	Value at 0 ps
A	B 00000...
Clock	B 0
Q	B 00000...
Resetn	B 1

Truth table for latch:

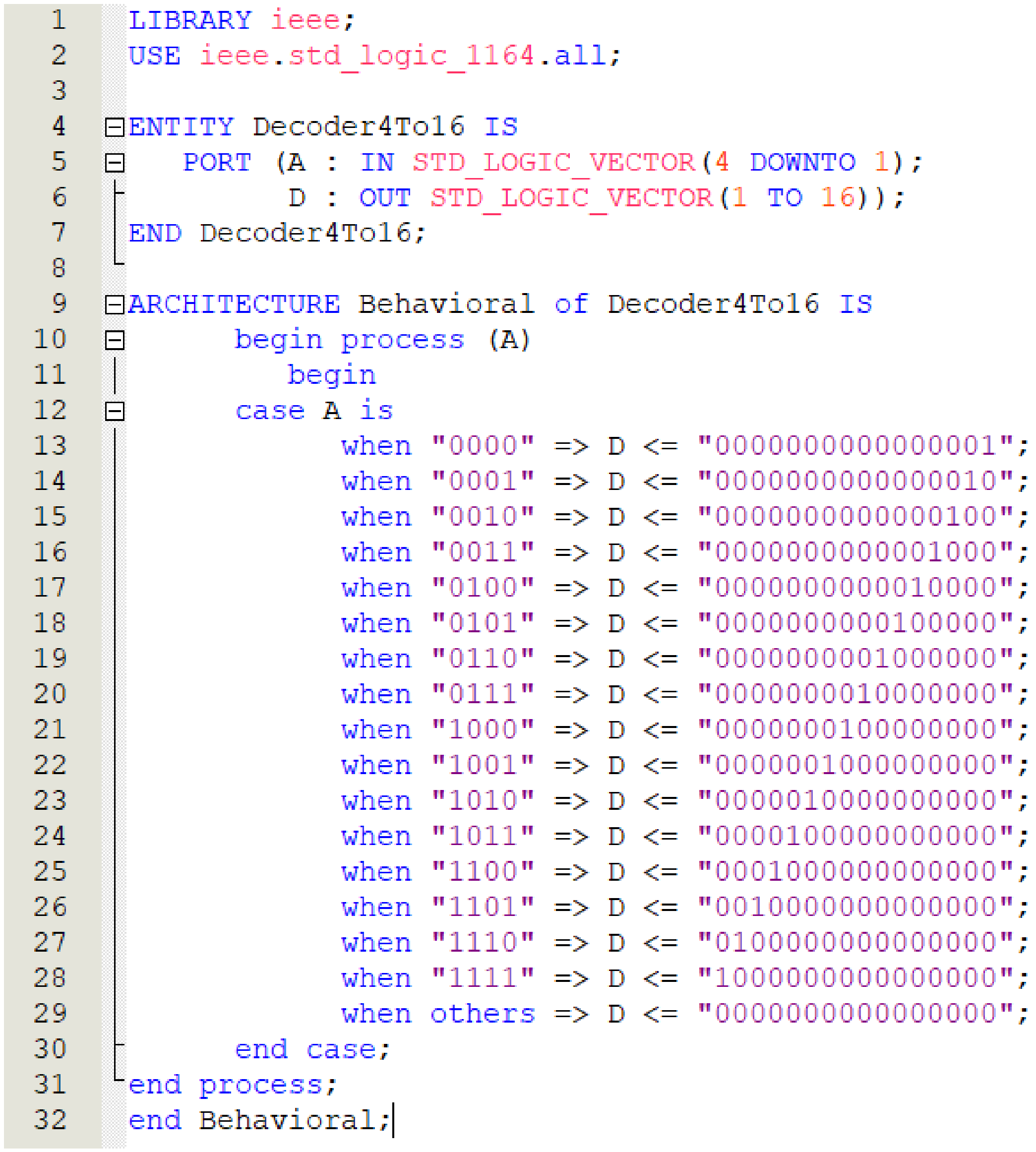


Clk	D	$Q(t+1)$
0	x	$Q(t)$
1	0	0
1	1	1

- 4:16 Decoder

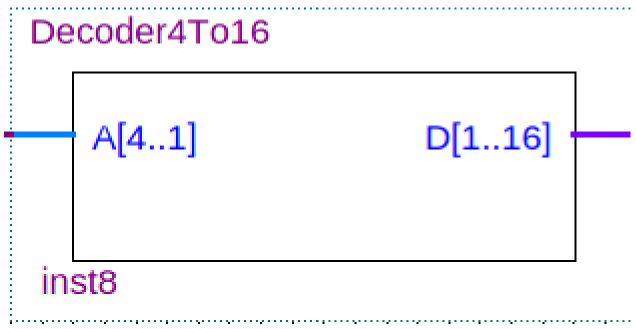
  The 4:16 decoder takes in the 4-bit state output of the FSM and gives out a unique 16-bit output for each state.

  Code for 4:16 decoder:

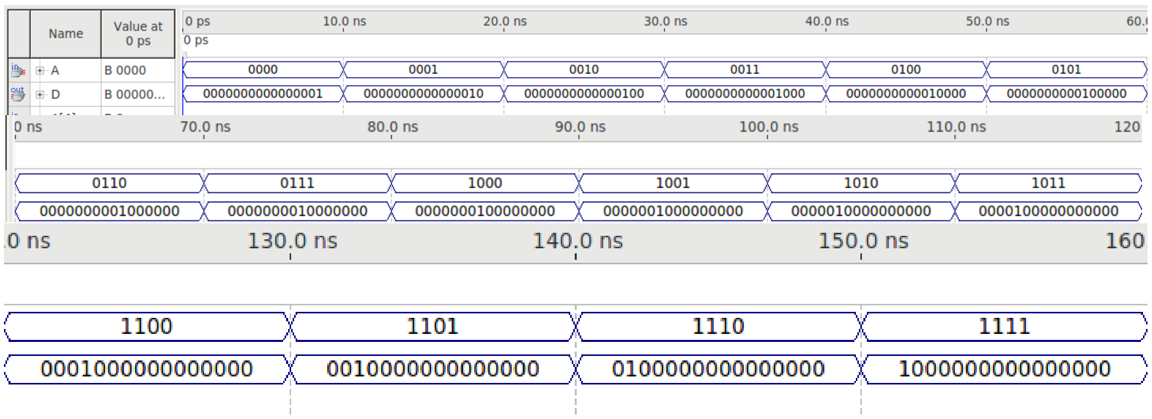


```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Decoder4To16 IS
    PORT (A : IN STD_LOGIC_VECTOR(4 DOWNTO 1);
            D : OUT STD_LOGIC_VECTOR(1 TO 16));
END Decoder4To16;

ARCHITECTURE Behavioral of Decoder4To16 IS
        begin process (A)
            begin
        case A is
                when "0000" => D <= "0000000000000001";
                when "0001" => D <= "0000000000000010";
                when "0010" => D <= "0000000000000100";
                when "0011" => D <= "0000000000001000";
                when "0100" => D <= "0000000000010000";
                when "0101" => D <= "0000000000100000";
                when "0110" => D <= "0000000001000000";
                when "0111" => D <= "0000000010000000";
                when "1000" => D <= "0000000100000000";
                when "1001" => D <= "0000001000000000";
                when "1010" => D <= "0000010000000000";
                when "1011" => D <= "0000100000000000";
                when "1100" => D <= "0001000000000000";
                when "1101" => D <= "0010000000000000";
                when "1110" => D <= "0100000000000000";
                when "1111" => D <= "1000000000000000";
                when others => D <= "0000000000000000";
        end case;
end process;
end Behavioral;
```

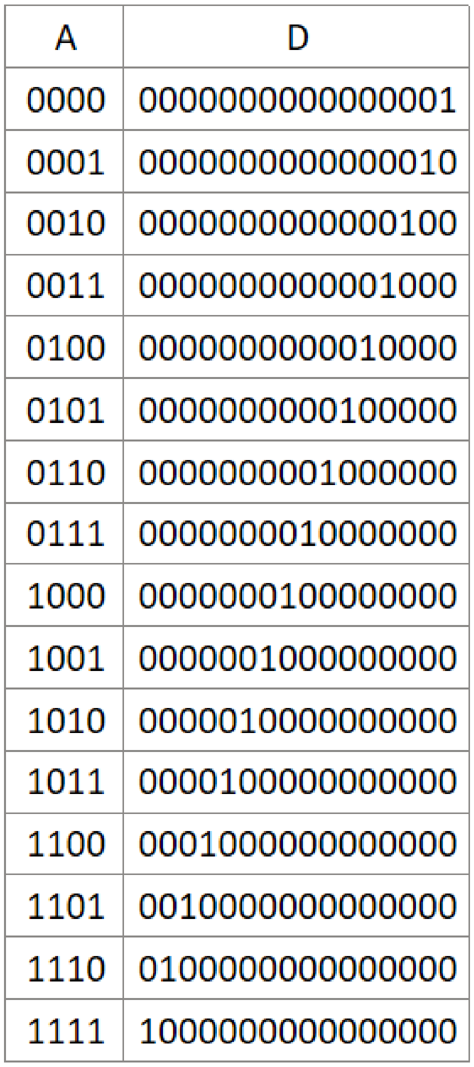
Component’s Circuit Diagram:



Waveform for 4:16 decoder:



Truth Table for 4:16 decoder:

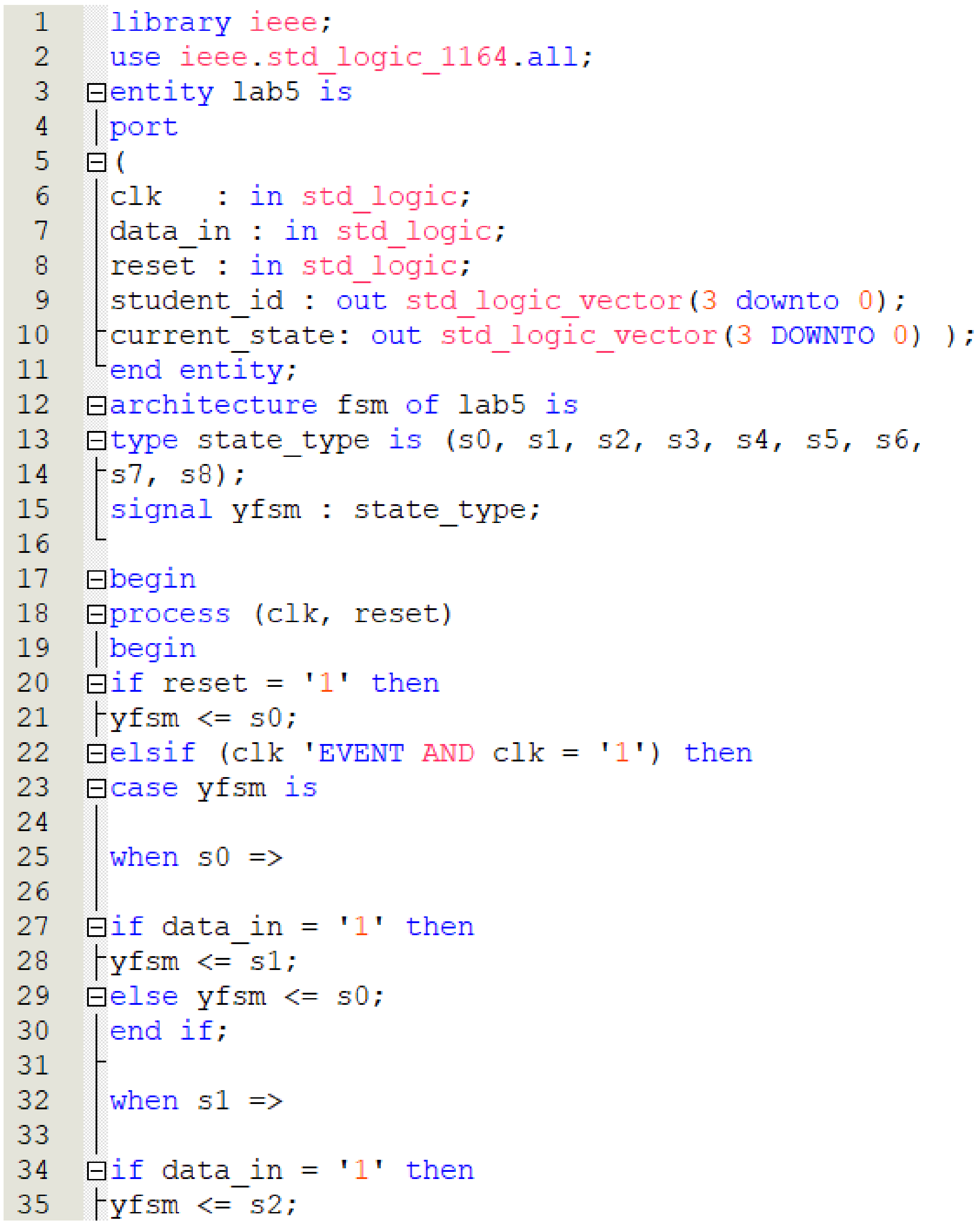


A	D
0000	0000000000000001
0001	0000000000000010
0010	0000000000000100
0011	0000000000001000
0100	0000000000010000
0101	0000000000100000
0110	0000000001000000
0111	0000000010000000
1000	0000000100000000
1001	0000001000000000
1010	0000010000000000
1011	0000100000000000
1100	0001000000000000
1101	0010000000000000
1110	0100000000000000
1111	1000000000000000

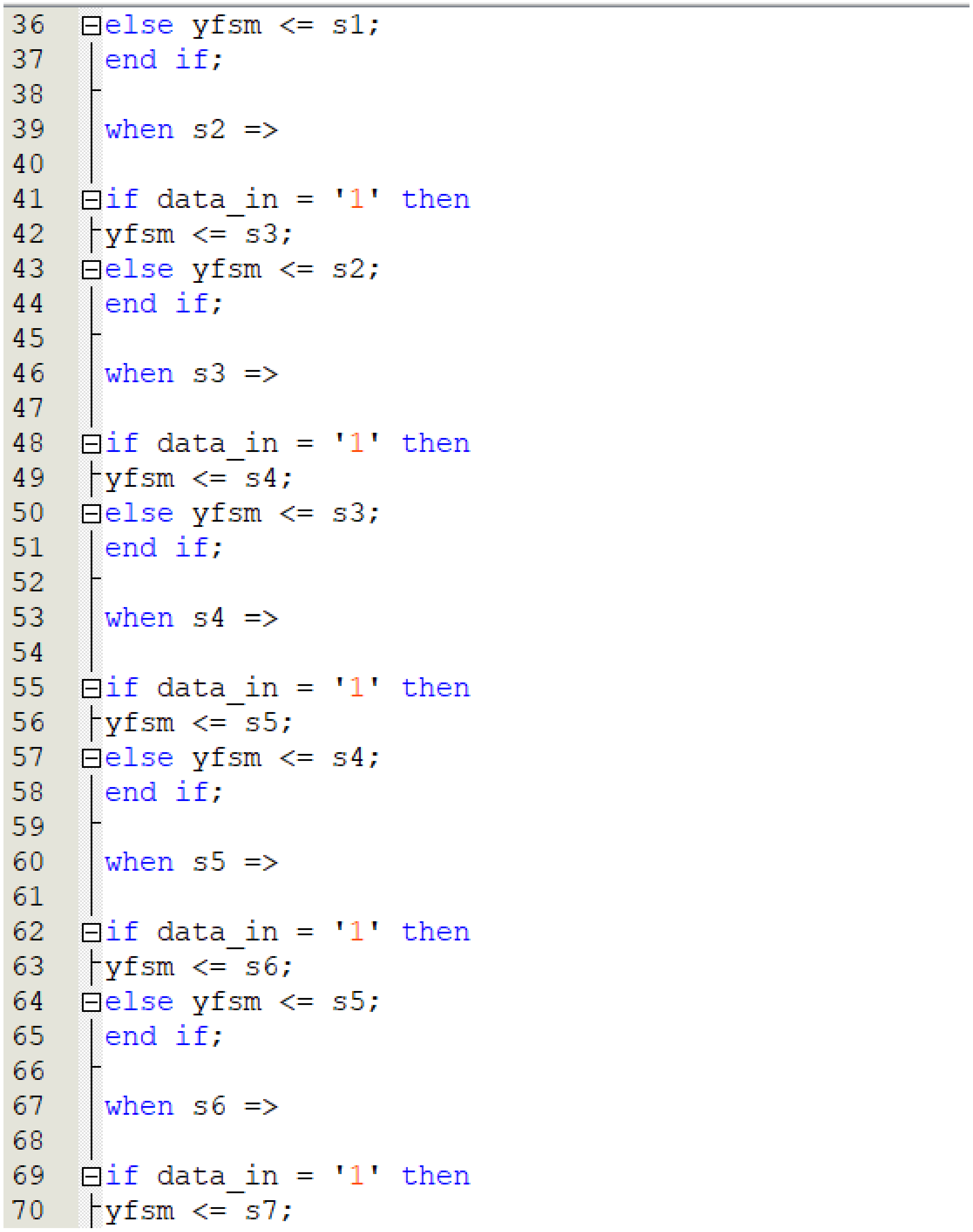
- Finite State Machine (FSM)

  The FSM is a Moore machine cycles through different states from S0 to S8, when the data input is 1 it moves from 1 state to the next and it gives a respective digit of my student ID for example the third sate would give the third digit of my student ID.

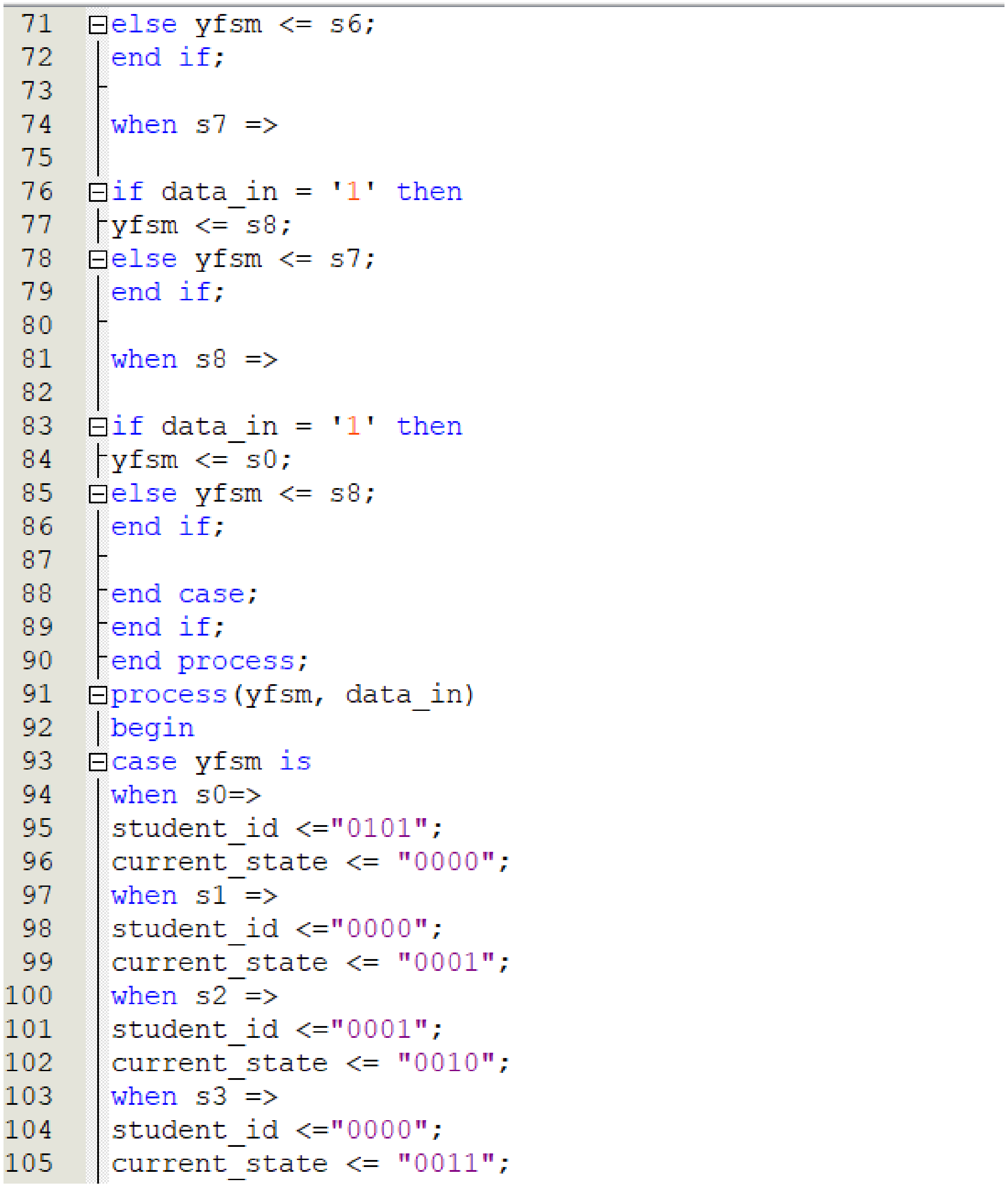
Code for FSM:



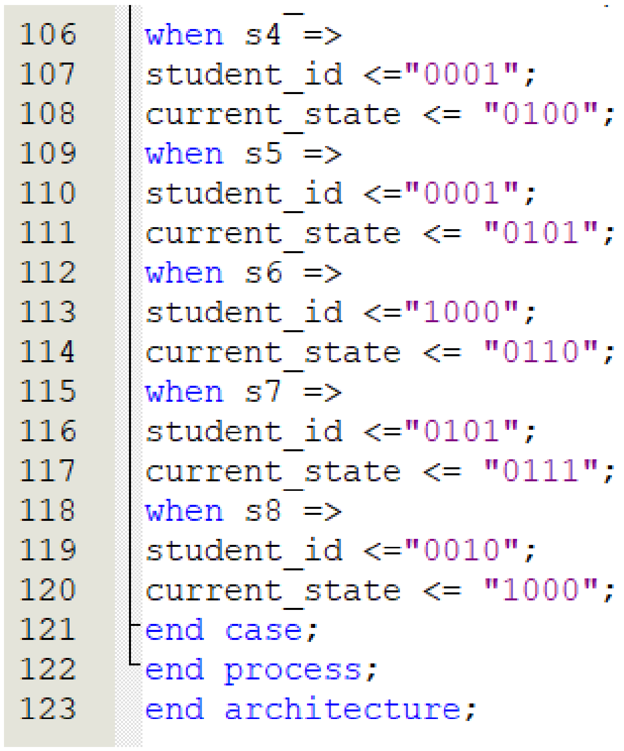
```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity lab5 is
port
(
clk    : in std_logic;
data_in : in std_logic;
reset : in std_logic;
student_id : out std_logic_vector(3 downto 0);
current_state: out std_logic_vector(3 DOWNTO 0) );
end entity;
architecture fsm of lab5 is
type state_type is (s0, 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;
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;
```

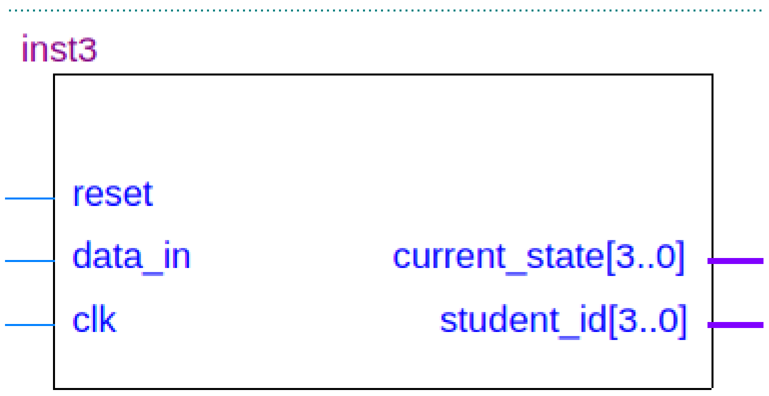


```
else yfsm <= s6;
end if;

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 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";
```

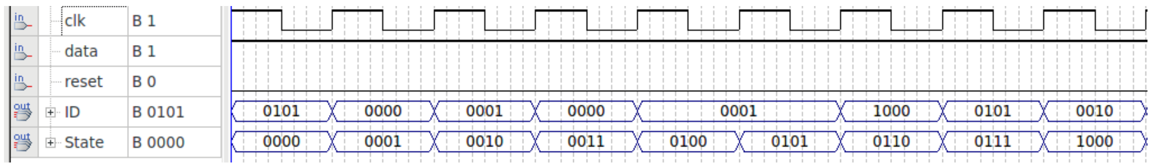


```
when s4 =>
student_id <="0001";
current_state <= "0100";
when s5 =>
student_id <="0001";
current_state <= "0101";
when s6 =>
student_id <="1000";
current_state <= "0110";
when s7 =>
student_id <="0101";
current_state <= "0111";
when s8 =>
student_id <="0010";
current_state <= "1000";
end case;
end process;
end architecture;
```

Component's Circuit Diagram:



Waveform for FSM:



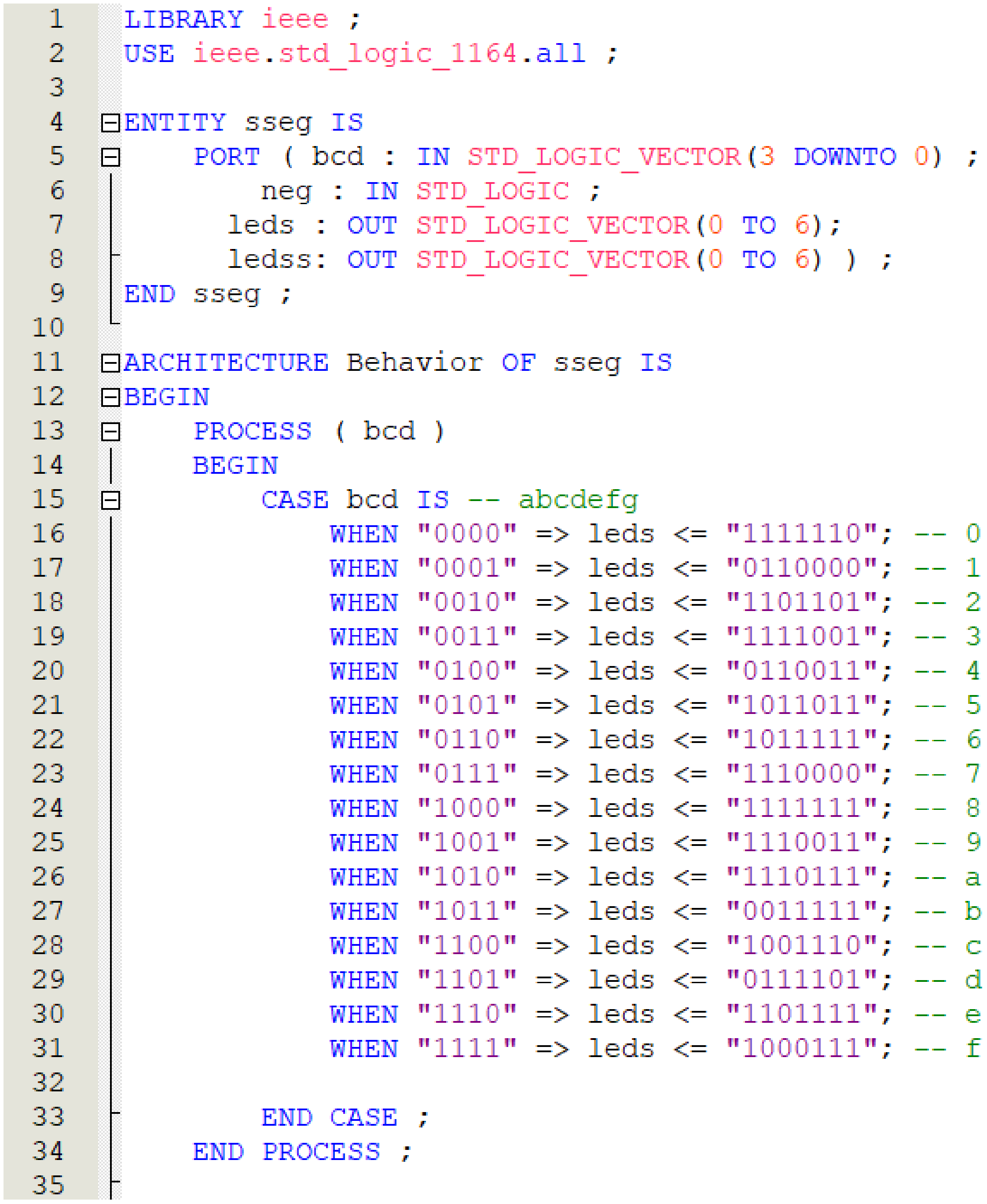
Truth Table for FSM:

Present State	Next State		Output student_id
	data_in=0	data_in=1	
0000	0000	0001	0101
0001	0001	0010	0000
0010	0010	0011	0001
0011	0011	0100	0000
0100	0100	0101	0001
0101	0101	0110	0001
0110	0110	0111	1000
0111	0111	1000	0101
1000	1000	0000	0010

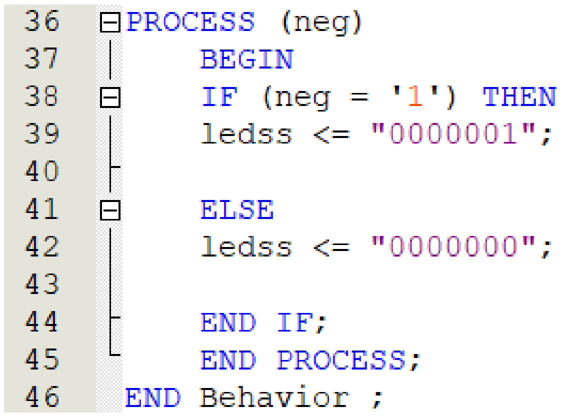
- 7 Segment display

The 7-segment display shows numerical output in hexadecimal starting from 0 to F, it can show up to 15 numbers on 1 7-segment display.

7 Segment display Code:

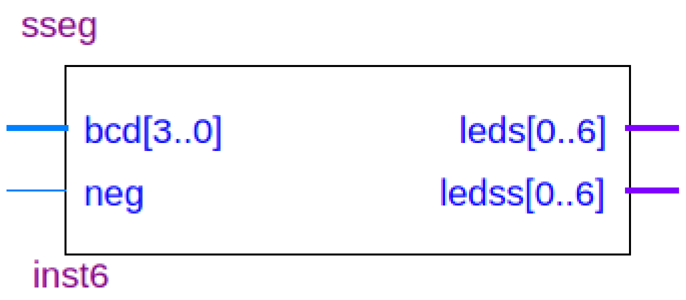


```vhdl
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY sseg IS
    PORT ( bcd : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
        neg : IN STD_LOGIC ;
      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 -- abcdefg
            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 <= "1110011"; -- 9
            WHEN "1010" => leds <= "1110111"; -- a
            WHEN "1011" => leds <= "0011111"; -- b
            WHEN "1100" => leds <= "1001110"; -- c
            WHEN "1101" => leds <= "0111101"; -- d
            WHEN "1110" => leds <= "1101111"; -- e
            WHEN "1111" => leds <= "1000111"; -- f

        END CASE ;
    END PROCESS ;

```

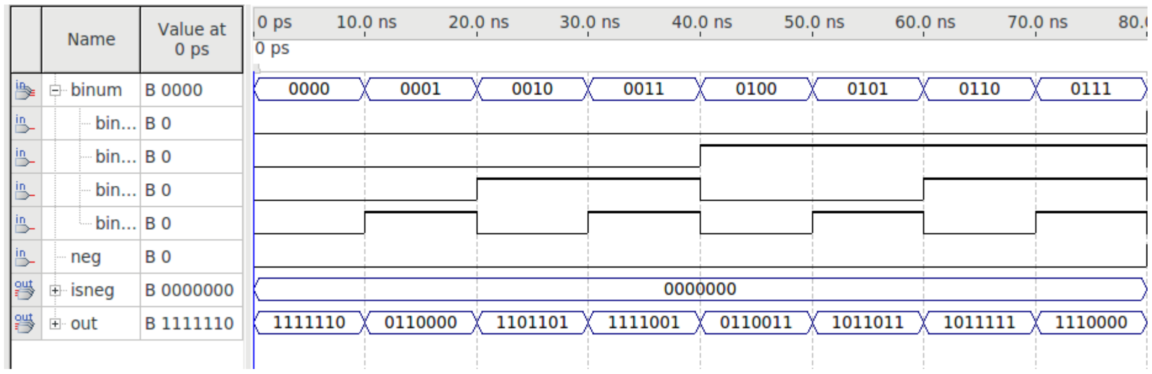


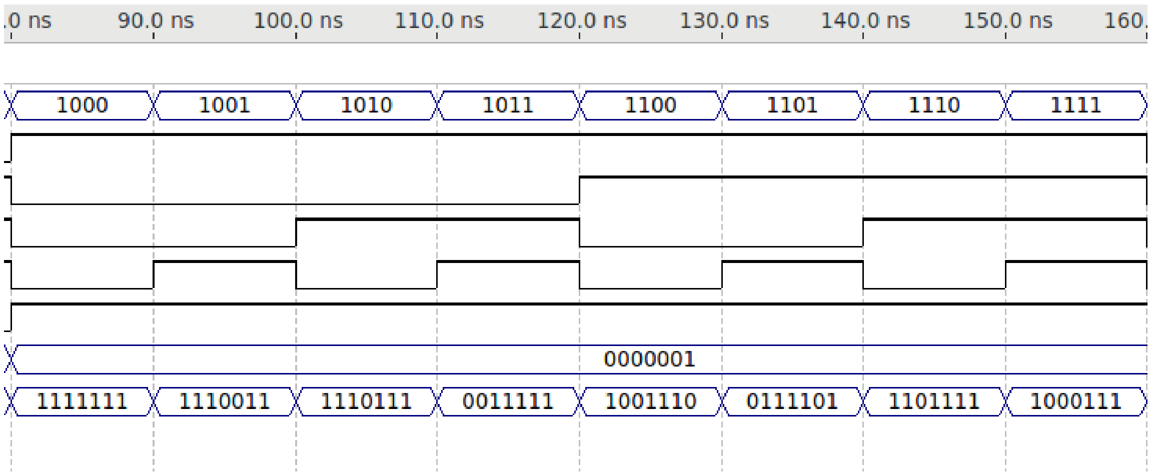
```
PROCESS (neg)
    BEGIN
    IF (neg = '1') THEN
    ledss <= "0000001";

    ELSE
    ledss <= "0000000";

    END IF;
    END PROCESS;
END Behavior ;
```

Component’s Circuit Diagram:



Waveform for 7-Segment display:



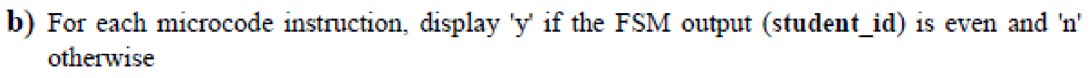


Truth Table for 7-segment display:

Input	leds
0000	1111110
0001	0110000
0010	1101101
0011	1111001
0100	0110011
0101	1011011
0110	1011111
0111	1110000
1000	1111111
1001	1110011
1010	1110111
1011	0011111
1100	1001110
1101	0111101
1110	1101111
1111	1000111

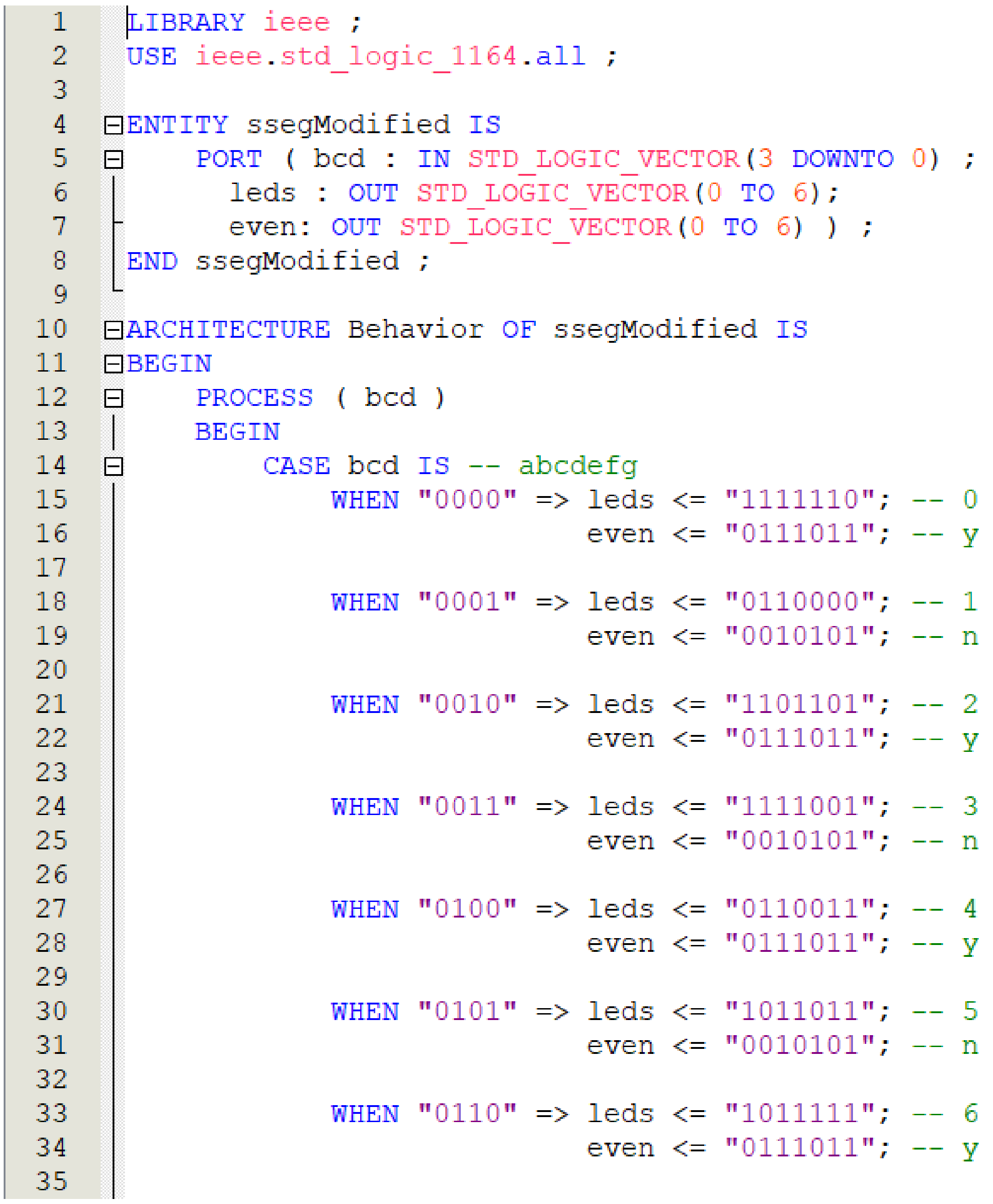
neg	Ledss
0	0000000
1	0000001

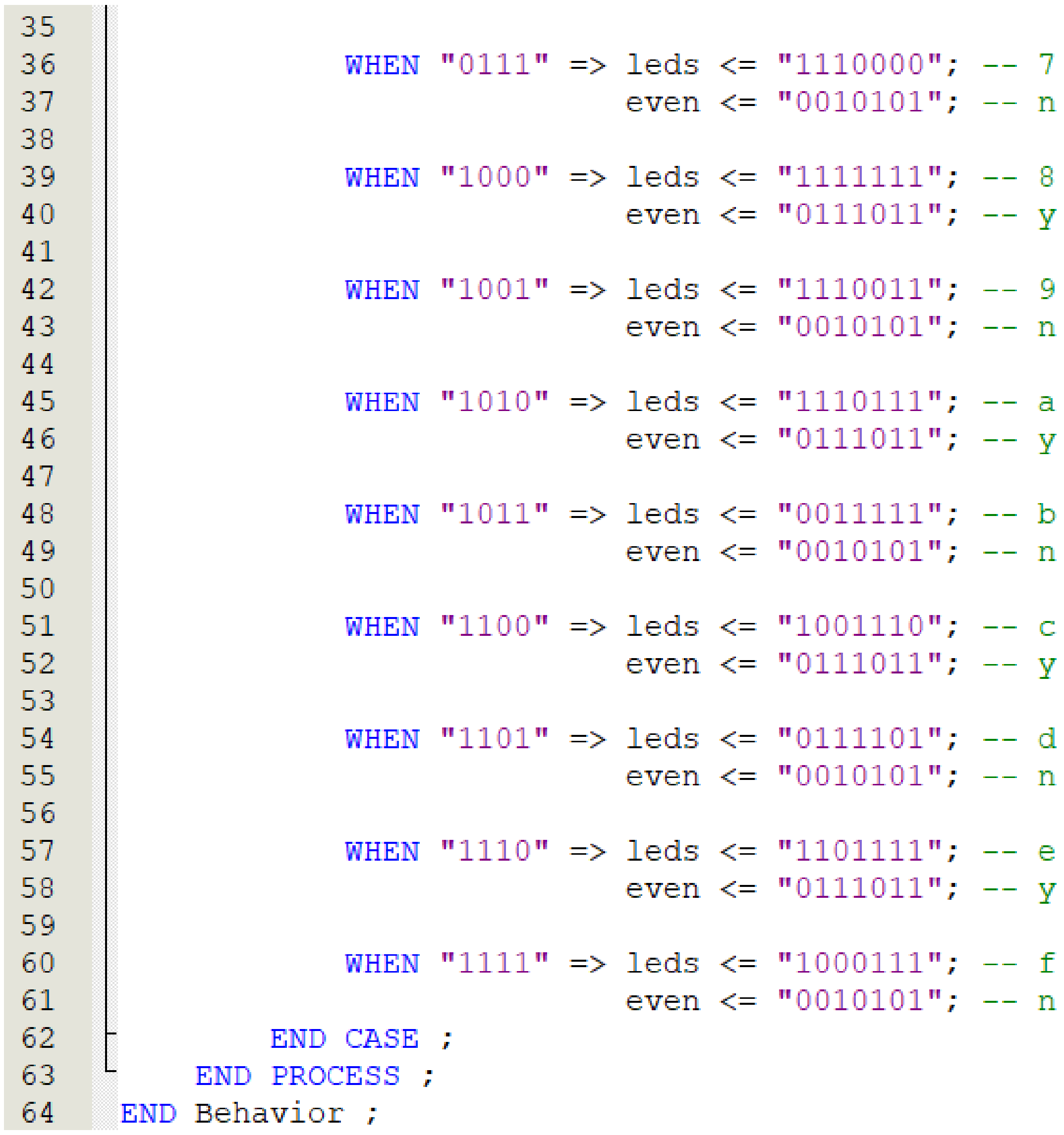
- Modified 7 Segment display (used in part 3 method 1):


b) For each microcode instruction, display 'y' if the FSM output (**student_id**) is even and 'n' otherwise

The 7-Segment checks if the number is even or odd and no changes needed for the ALU.

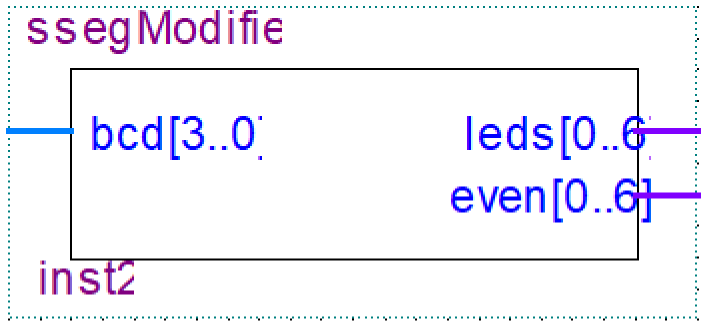
Code for 7-segment display:


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

ENTITY ssegModified IS
    PORT ( bcd : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
      leds : OUT STD_LOGIC_VECTOR(0 TO 6);
      even: OUT STD_LOGIC_VECTOR(0 TO 6) ) ;
END ssegModified ;

ARCHITECTURE Behavior OF ssegModified IS
BEGIN
    PROCESS ( bcd )
    BEGIN
        CASE bcd IS -- abcdefg
            WHEN "0000" => leds <= "1111110"; -- 0
                           even <= "0111011"; -- y

            WHEN "0001" => leds <= "0110000"; -- 1
                           even <= "0010101"; -- n

            WHEN "0010" => leds <= "1101101"; -- 2
                           even <= "0111011"; -- y

            WHEN "0011" => leds <= "1111001"; -- 3
                           even <= "0010101"; -- n

            WHEN "0100" => leds <= "0110011"; -- 4
                           even <= "0111011"; -- y

            WHEN "0101" => leds <= "1011011"; -- 5
                           even <= "0010101"; -- n

            WHEN "0110" => leds <= "1011111"; -- 6
                           even <= "0111011"; -- y

```

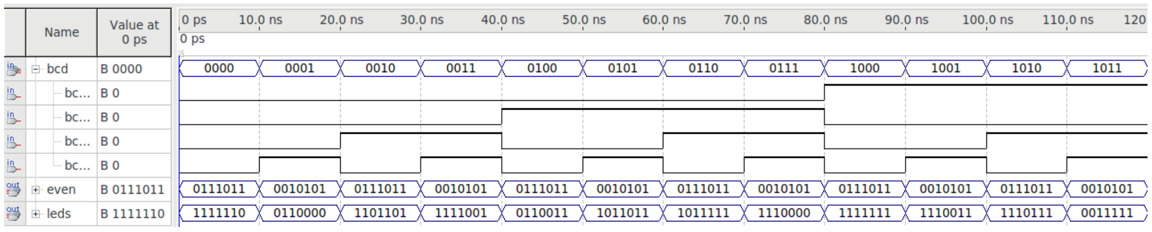


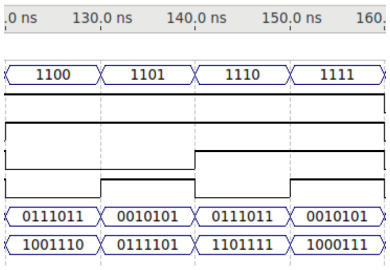
```

            WHEN "0111" => leds <= "1110000"; -- 7
                           even <= "0010101"; -- n

            WHEN "1000" => leds <= "1111111"; -- 8
                           even <= "0111011"; -- y

            WHEN "1001" => leds <= "1110011"; -- 9
                           even <= "0010101"; -- n

            WHEN "1010" => leds <= "1110111"; -- a
                           even <= "0111011"; -- y

            WHEN "1011" => leds <= "0011111"; -- b
                           even <= "0010101"; -- n

            WHEN "1100" => leds <= "1001110"; -- c
                           even <= "0111011"; -- y

            WHEN "1101" => leds <= "0111101"; -- d
                           even <= "0010101"; -- n

            WHEN "1110" => leds <= "1101111"; -- e
                           even <= "0111011"; -- y

            WHEN "1111" => leds <= "1000111"; -- f
                           even <= "0010101"; -- n
        END CASE ;
    END PROCESS ;
END Behavior ;
```

Component's Circuit Diagram:



Waveform for modified 7-segment display:





Truth Table for 7-segment display:

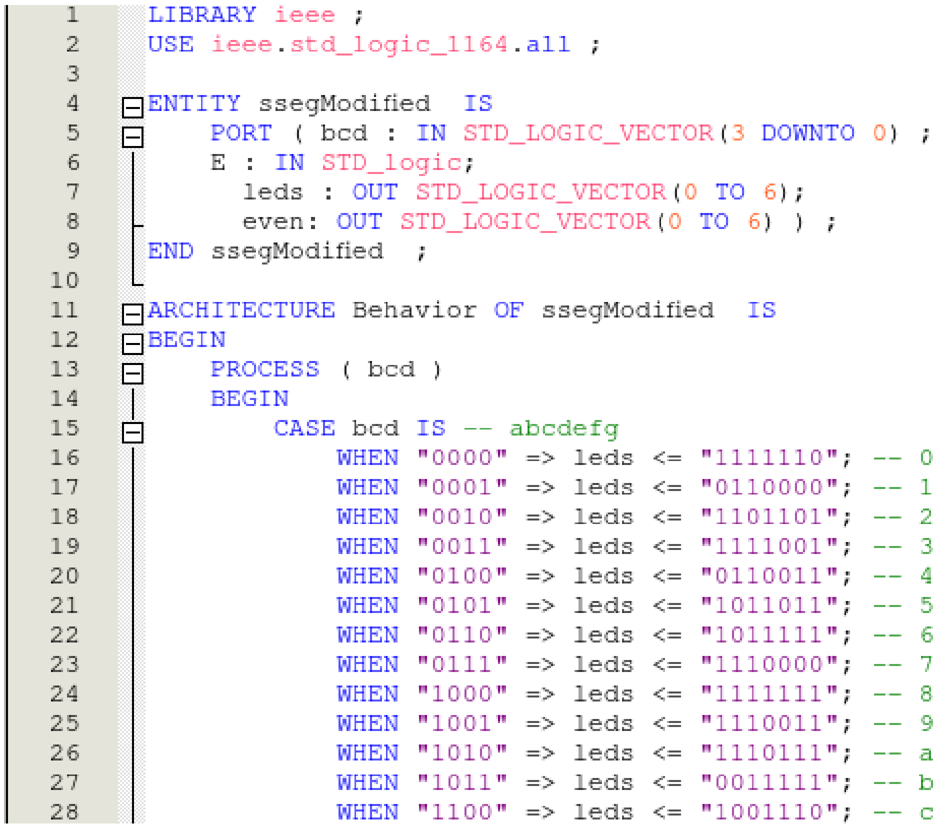
Input	leds	even
0000	1111110	0111011
0001	0110000	0010101
0010	1101101	0111011
0011	1111001	0010101
0100	0110011	0111011
0101	1011011	0010101
0110	1011111	0111011
0111	1110000	0010101
1000	1111111	0111011
1001	1110011	0010101
1010	1110111	0111011
1011	0011111	0010101
1100	1001110	0111011
1101	0111101	0010101
1110	1101111	0111011
1111	1000111	0010101

- Modified 7 Segment display (used in part 3 method2):

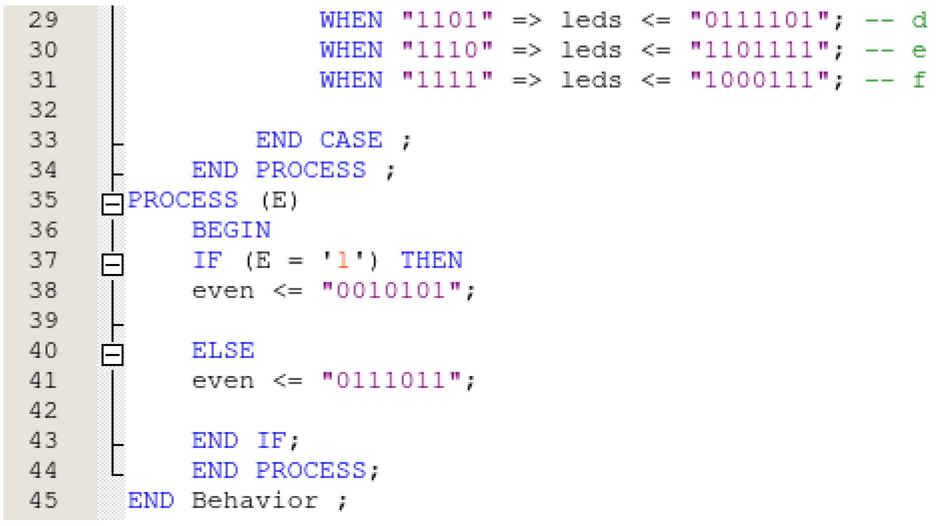
b) For each microcode instruction, display 'y' if the FSM output (**student_id**) is even and 'n' otherwise

ALU checks if the number is even or odd.

Code for 7-Segment display:

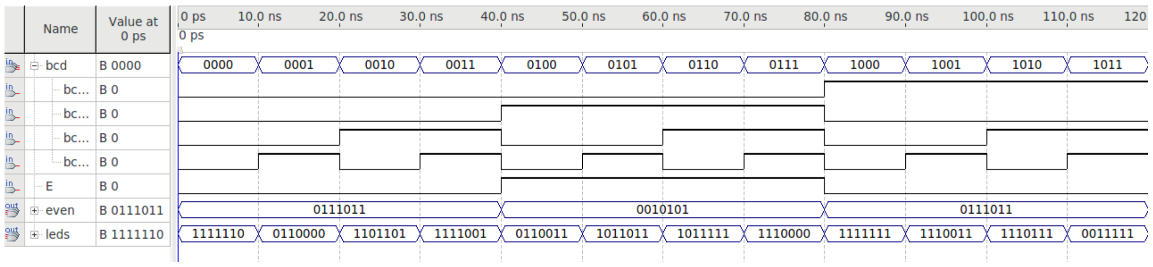


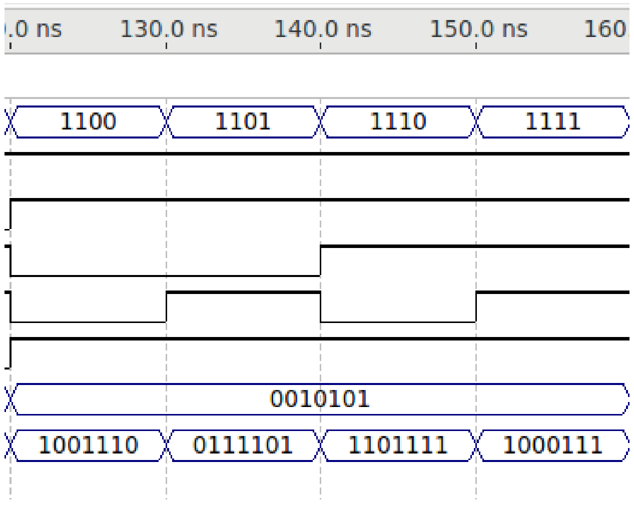
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY ssegModified  IS
    PORT ( bcd : IN STD_LOGIC_VECTOR(3 DOWNTO 0) ;
    E : IN STD_logic;
      leds : OUT STD_LOGIC_VECTOR(0 TO 6);
      even: OUT STD_LOGIC_VECTOR(0 TO 6) ) ;
END ssegModified  ;

ARCHITECTURE Behavior OF ssegModified  IS
BEGIN
    PROCESS ( bcd )
    BEGIN
        CASE bcd IS -- abcdefg
            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 <= "1110011"; -- 9
            WHEN "1010" => leds <= "1110111"; -- a
            WHEN "1011" => leds <= "0011111"; -- b
            WHEN "1100" => leds <= "1001110"; -- c
```



```
            WHEN "1101" => leds <= "0111101"; -- d
            WHEN "1110" => leds <= "1101111"; -- e
            WHEN "1111" => leds <= "1000111"; -- f

        END CASE ;
    END PROCESS ;
PROCESS (E)
    BEGIN
    IF (E = '1') THEN
    even <= "0010101";

    ELSE
    even <= "0111011";

    END IF;
    END PROCESS;
END Behavior ;
```

Waveform for 7-segment display:



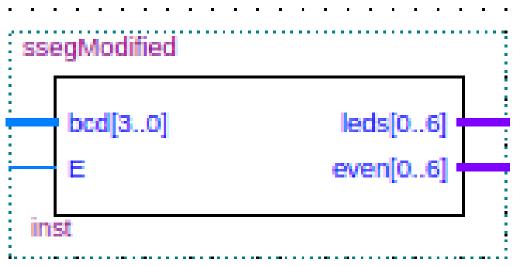


Truth Table for 7-segment display:

Input	leds
0000	1111110
0001	0110000
0010	1101101
0011	1111001
0100	0110011
0101	1011011
0110	1011111
0111	1110000
1000	1111111
1001	1110011
1010	1110111
1011	0011111
1100	1001110
1101	0111101
1110	1101111
1111	1000111

E	Even
0	0111011
1	0010101

Component's Circuit Diagram:



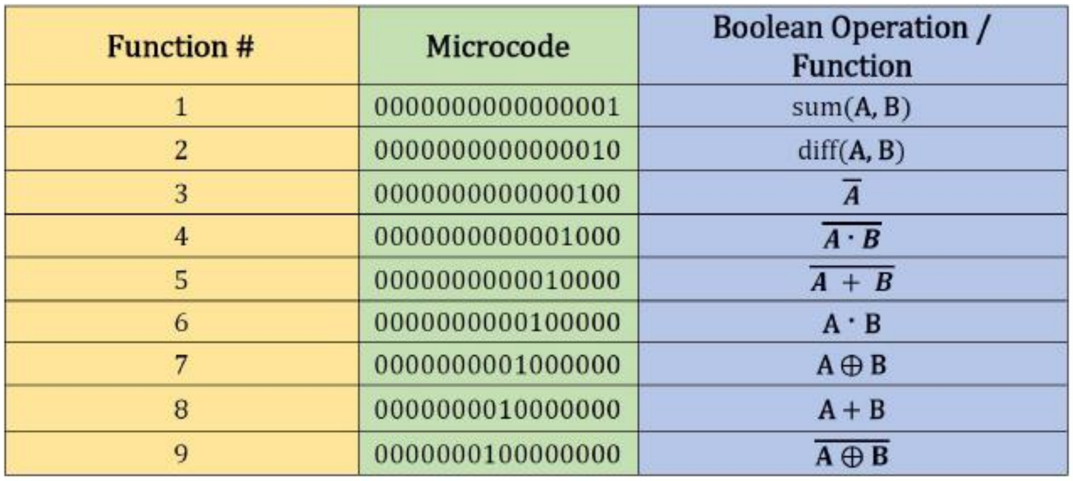
Arithmetic logical unit (ALU)

- ALU for Part 1

The ALU is the processor that decides what operation to do on the inputs based on the state of the FSM. It has 5 inputs:

1. Clock
2. A
3. B
4. student_id
5. OP

Input A and B are binary numbers which are the input of the storage unit as long as the data_in is 1. The clock input alternates between 1 and 0, when the clock input changes from 0 to 1 (rising edge) the ALU checks the value OP input (which is the output of the decoder) and runs the switch statement matching the value of the OP input to the case and doing its respective operation on A and B according to the table in the lab manual. The student_id input has no purpose in part.



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	$A \cdot B$
7	0000000001000000	$A \oplus B$
8	0000000010000000	$A + B$
9	0000000100000000	$\overline{A \oplus B}$

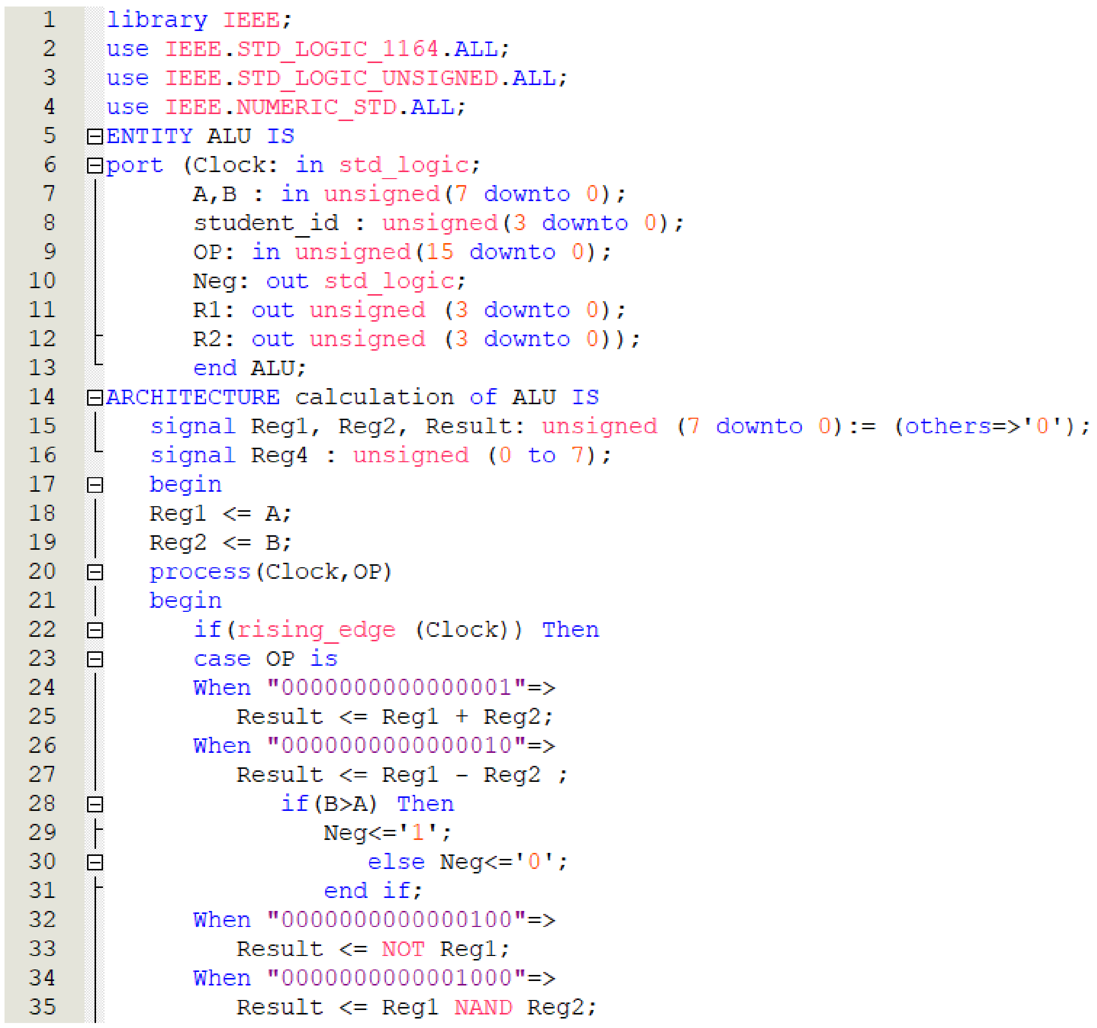
The ALU has 3 outputs:

1. Neg
2. R1
3. R2

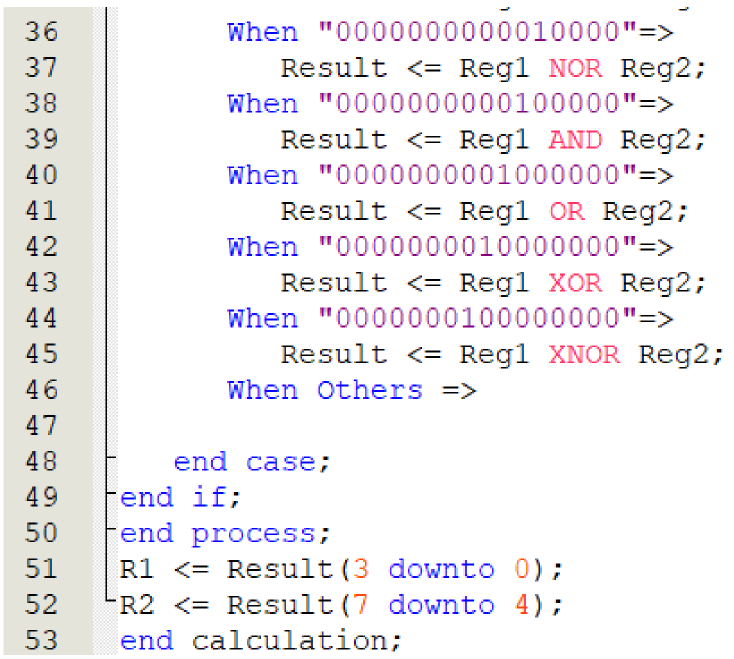
Neg outputs 1 if the number is negative and lights up the g led on the 7-segment display

R1 and R2 are 4-bit outputs when combined gives an 8-bit output each one goes to a 7-segment display to display the binary output but in hexadecimal.

Code for ALU part 1:

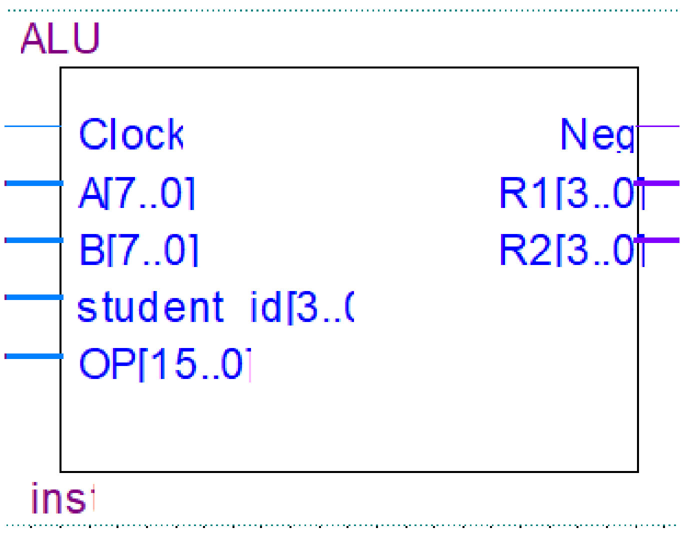


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
ENTITY ALU IS
port (Clock: in std_logic;
      A,B : in unsigned(7 downto 0);
      student_id : unsigned(3 downto 0);
      OP: in unsigned(15 downto 0);
      Neg: out std_logic;
      R1: out unsigned (3 downto 0);
      R2: out unsigned (3 downto 0));
      end ALU;
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"=>
         Result <= Reg1 - Reg2 ;
            if(B>A) Then
               Neg<='1';
                  else Neg<='0';
               end if;
      When "0000000000000100"=>
         Result <= NOT Reg1;
      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;
R1 <= Result(3 downto 0);
R2 <= Result(7 downto 4);
end calculation;
```

Component’s Circuit Diagram:



- ALU for part 2:

The ALU is the processor that decides what operation to do on the inputs based on the state of the FSM. It has 5 inputs:

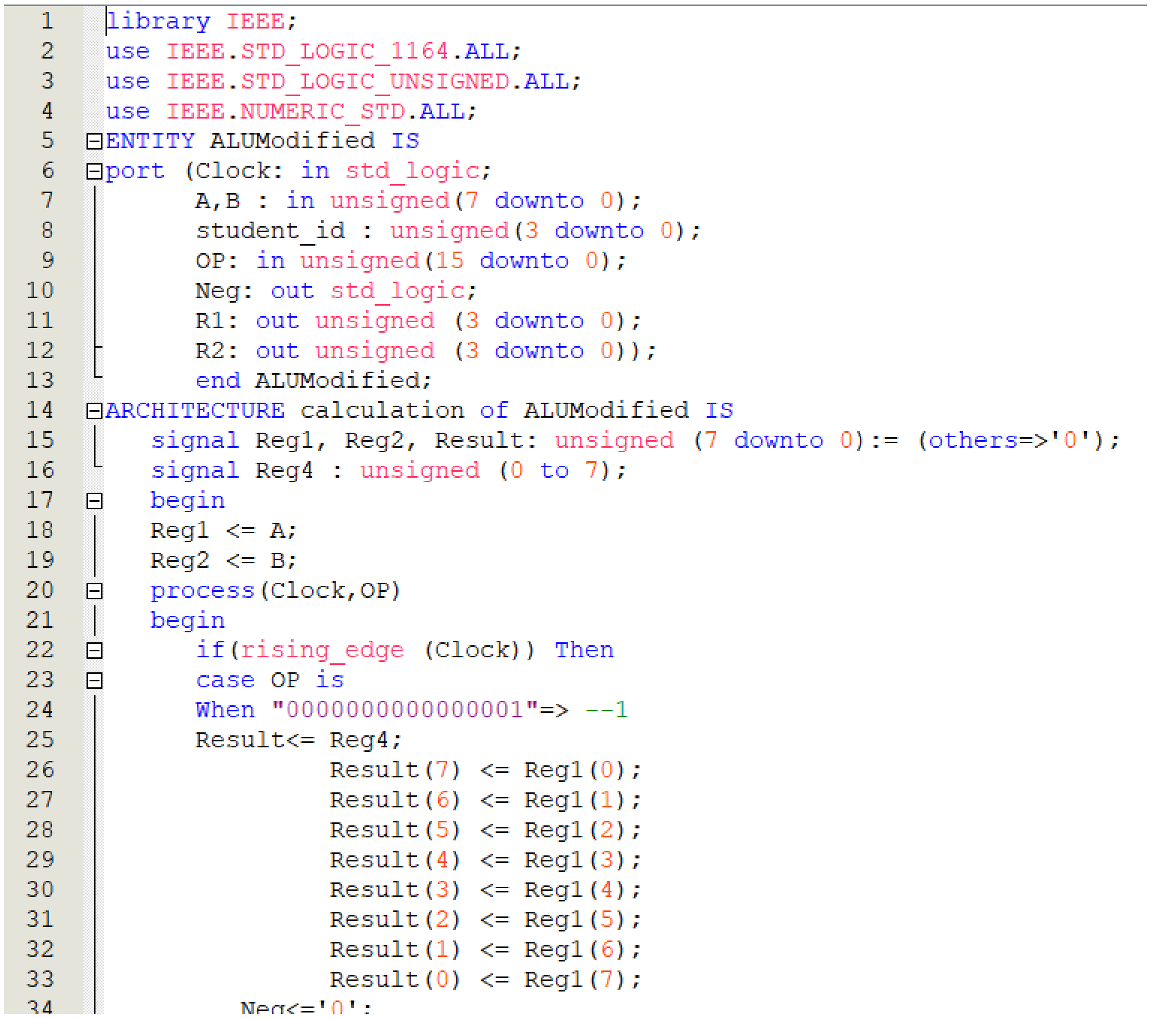
1. Clock
2. A
3. B
4. student_id
5. OP

Input A and B are binary numbers which are the input of the storage unit as long as the data_in is 1. The clock input alternates between 1 and 0, when the clock input changes from 0 to 1 (rising edge) the ALU checks the value OP input (which is the output of the decoder) and runs the switch statement matching the value of the OP input to the case and doing its respective operation on A and B according to the table in the lab manual. The student_id input has no purpose in part.

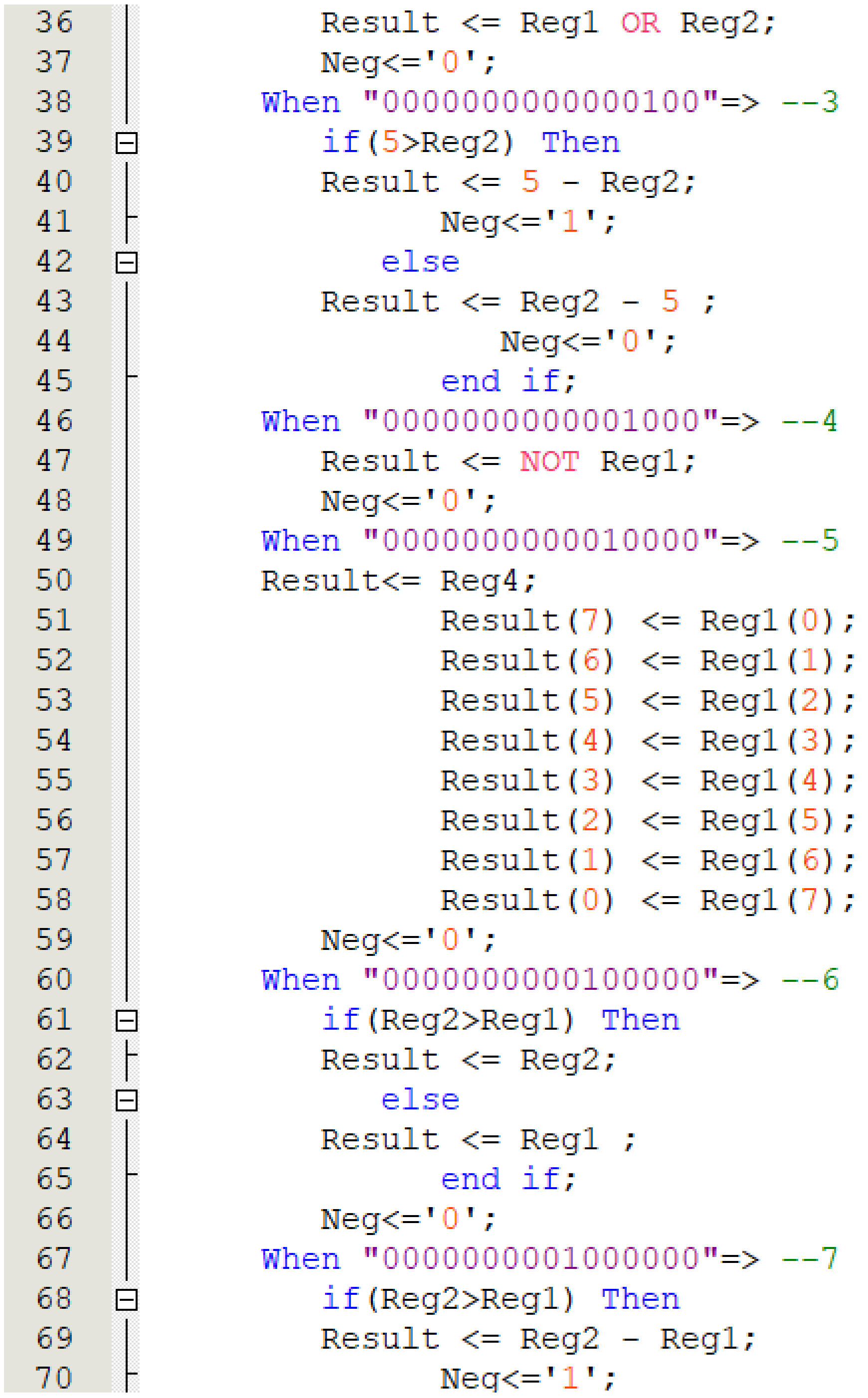
Function #	Operation / Function
1	Swap the lower and upper 4 bits of A
2	Produce the result of ORing A and B
3	Decrement **B** by 5
4	Invert all bits of A
5	Invert the bit-significance order of **A**
6	Find the greater value of **A** and **B** and produce the results (Max(**A**,**B**))
7	Produce the difference between **A** and **B**
8	Produce the result of XNORing **A** and **B**
9	Rotate **B** to left by three bits (ROL)

For the third function (difference) the output would normally be in 2’s compliment however in my approach I decided to make the output be in decimal for simplicity, so instead of giving -FF for -3 I decided to make it -3, by adding an if statement.

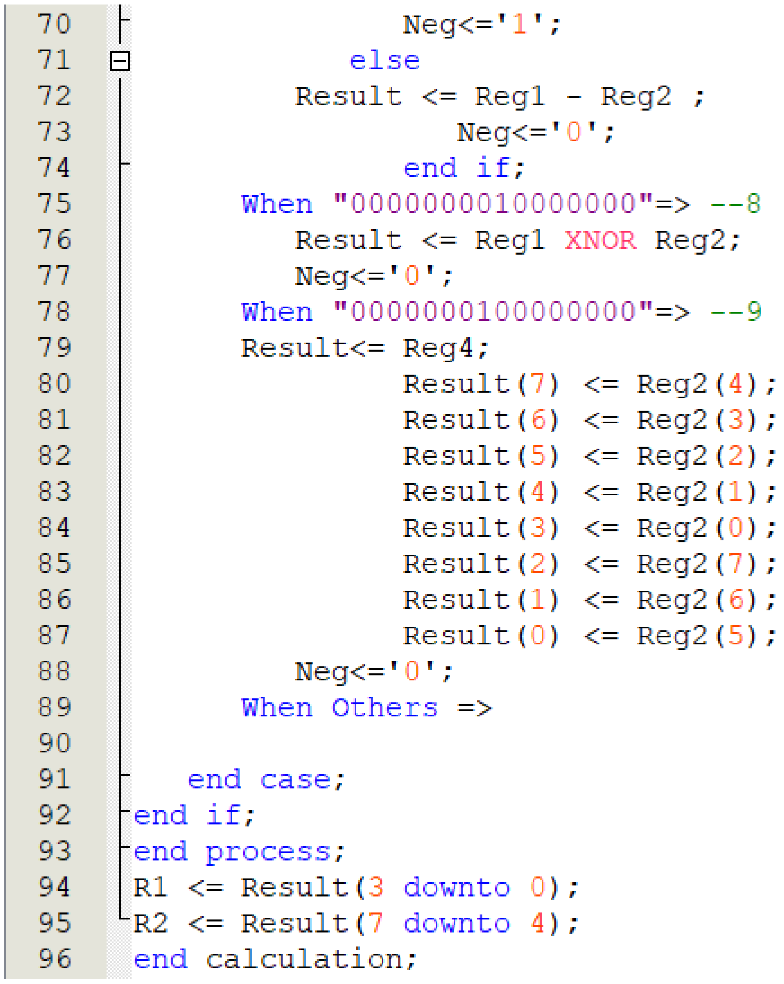
Code for ALU part 2:



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
ENTITY ALUModified IS
port (Clock: in std_logic;
      A,B : in unsigned(7 downto 0);
      student_id : 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 ALUModified;
ARCHITECTURE calculation of ALUModified 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"=> --1
      Result<= Reg4;
               Result(7) <= Reg1(0);
               Result(6) <= Reg1(1);
               Result(5) <= Reg1(2);
               Result(4) <= Reg1(3);
               Result(3) <= Reg1(4);
               Result(2) <= Reg1(5);
               Result(1) <= Reg1(6);
               Result(0) <= Reg1(7);
         Neg<='0';
```

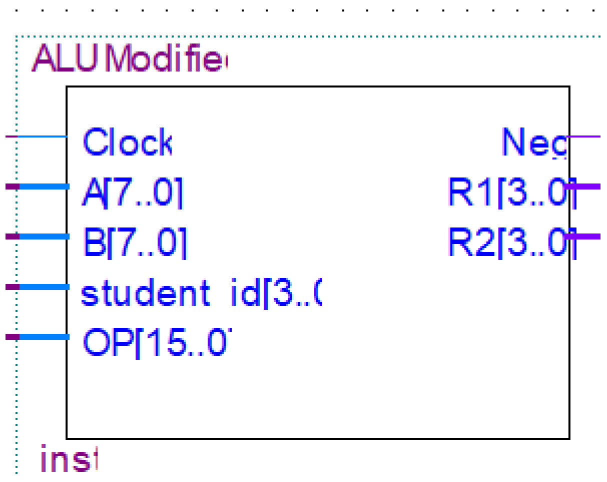


```
   Result <= Reg1 OR Reg2;
   Neg<='0';
When "0000000000000100"=> --3
   if(5>Reg2) Then
   Result <= 5 - Reg2;
         Neg<='1';
      else
   Result <= Reg2 - 5 ;
            Neg<='0';
         end if;
When "0000000000001000"=> --4
   Result <= NOT Reg1;
   Neg<='0';
When "0000000000010000"=> --5
Result<= Reg4;
         Result(7) <= Reg1(0);
         Result(6) <= Reg1(1);
         Result(5) <= Reg1(2);
         Result(4) <= Reg1(3);
         Result(3) <= Reg1(4);
         Result(2) <= Reg1(5);
         Result(1) <= Reg1(6);
         Result(0) <= Reg1(7);
   Neg<='0';
When "0000000000100000"=> --6
   if(Reg2>Reg1) Then
   Result <= Reg2;
      else
   Result <= Reg1 ;
         end if;
   Neg<='0';
When "0000000001000000"=> --7
   if(Reg2>Reg1) Then
   Result <= Reg2 - Reg1;
         Neg<='1';
```



```
               Neg<='1';
            else
         Result <= Reg1 - Reg2 ;
                  Neg<='0';
               end if;
      When "0000000010000000"=> --8
         Result <= Reg1 XNOR Reg2;
         Neg<='0';
      When "0000000100000000"=> --9
      Result<= Reg4;
               Result(7) <= Reg2(4);
               Result(6) <= Reg2(3);
               Result(5) <= Reg2(2);
               Result(4) <= Reg2(1);
               Result(3) <= Reg2(0);
               Result(2) <= Reg2(7);
               Result(1) <= Reg2(6);
               Result(0) <= Reg2(5);
         Neg<='0';
      When Others =>

   end case;
end if;
end process;
R1 <= Result(3 downto 0);
R2 <= Result(7 downto 4);
end calculation;
```

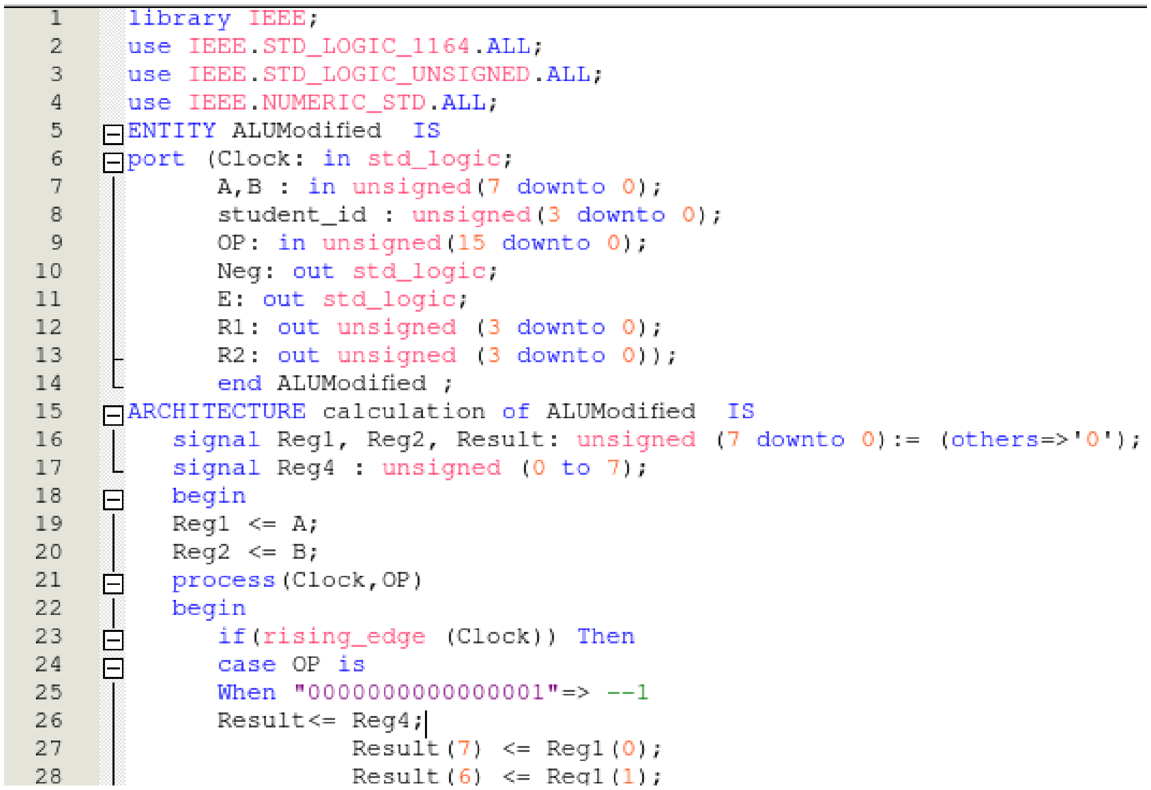
Component’s Circuit Diagram:



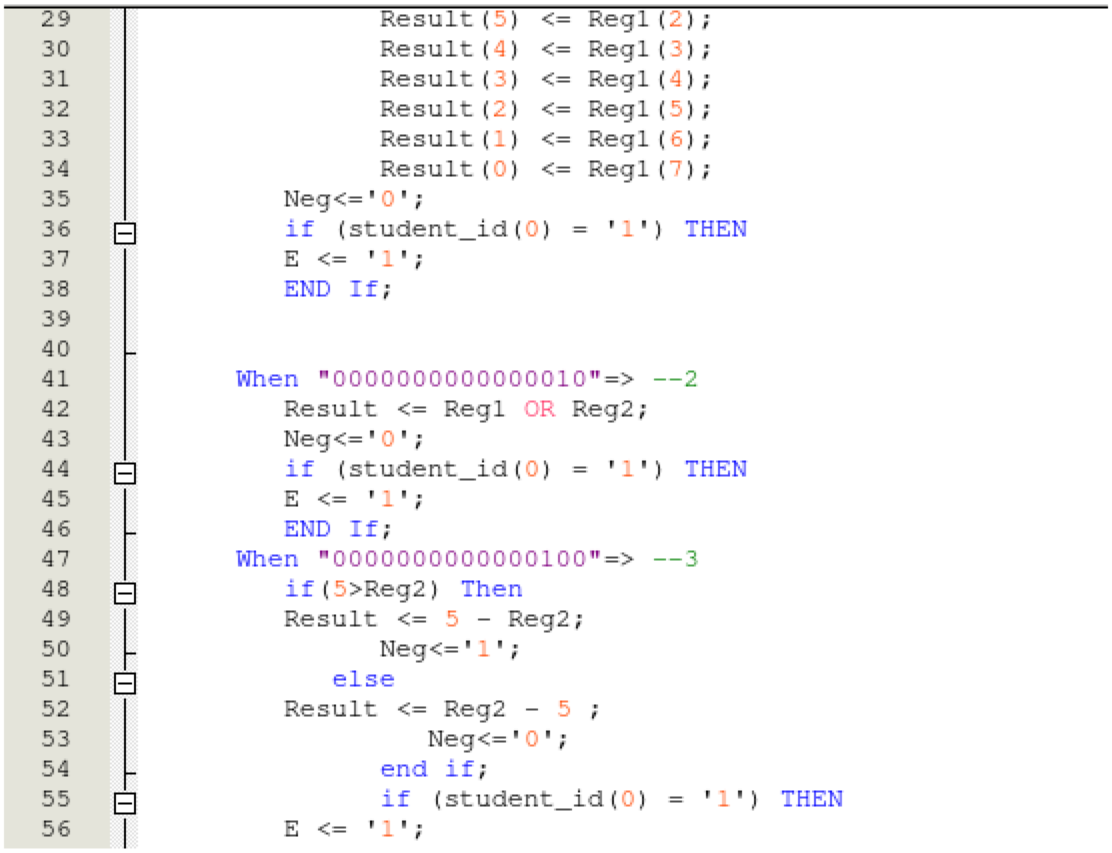
- ALU for part 3 (method 2):

In this part the student_id input is used and the ALU checks if its LSB is 1 or 0 since 1 means it is an odd number and 0 means it is an even number. The ALU also has a new output E that sends 1 if the number is odd and 0 if even which then goes into the 7-segment display and goes through an if statement to show either y for even or n for odd.

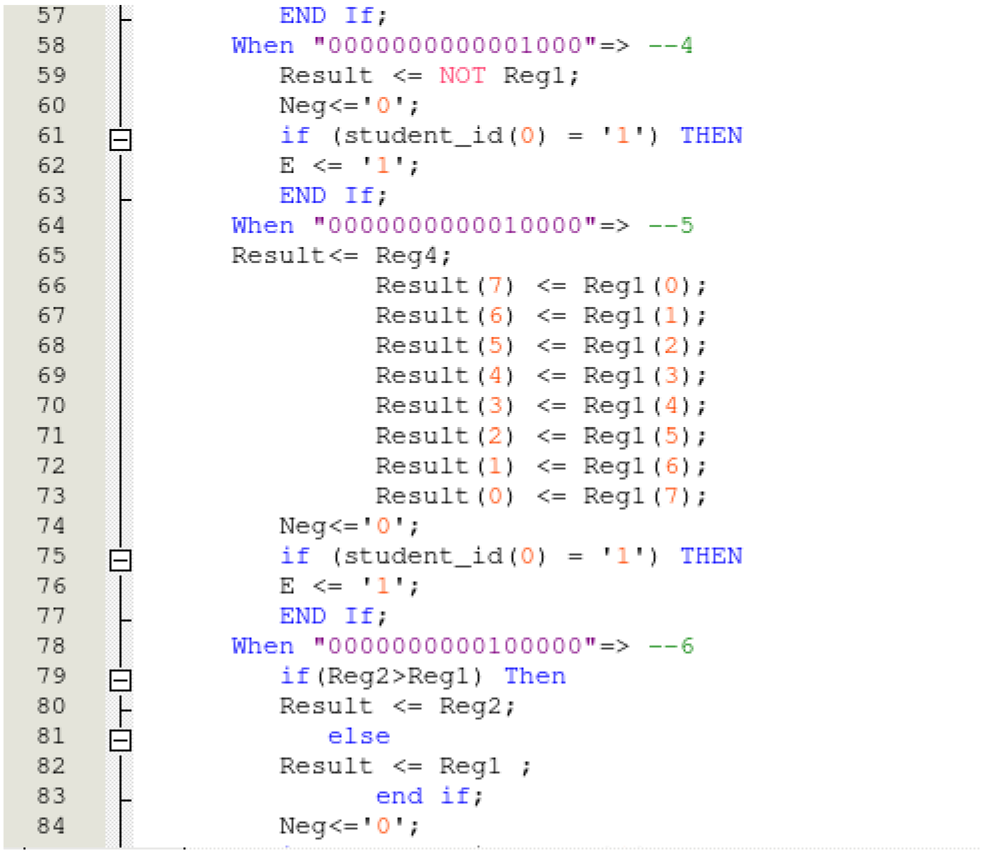
Code for ALU:



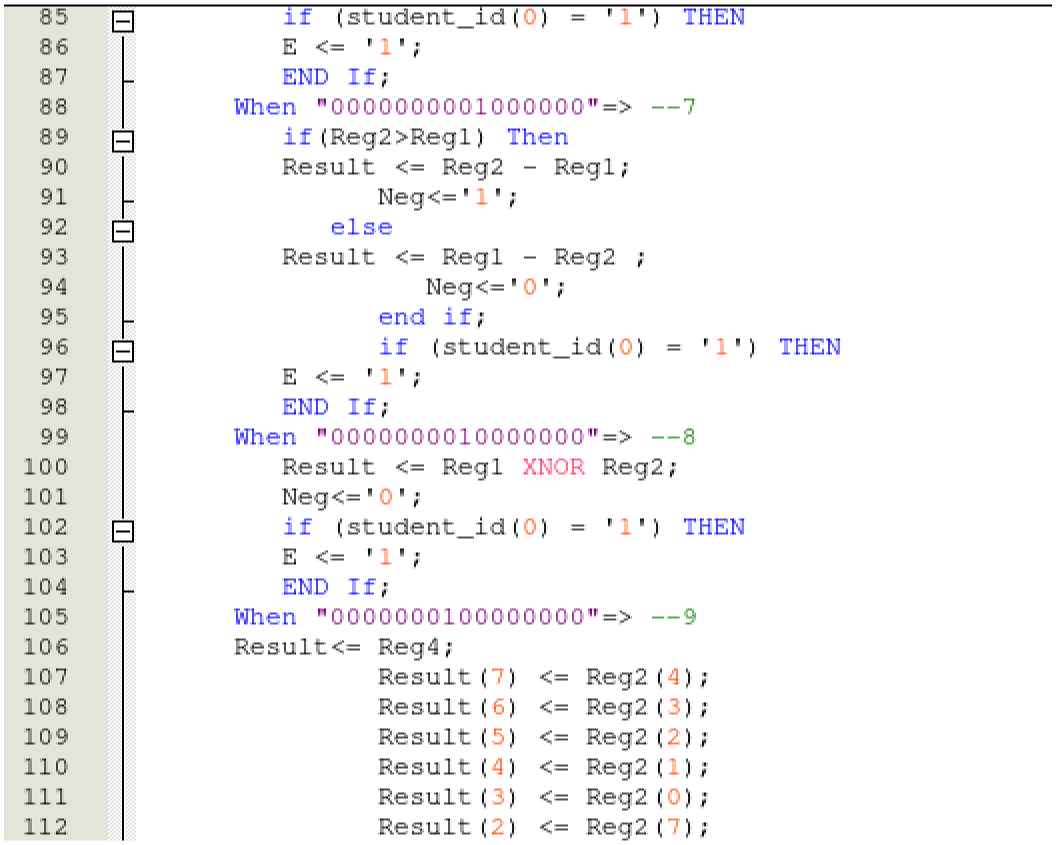
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
ENTITY ALUModified  IS
port  (Clock: in std_logic;
      A,B : in unsigned(7 downto 0);
      student_id : unsigned(3 downto 0);
      OP: in unsigned(15 downto 0);
      Neg: out std_logic;
      E: out std_logic;
      R1: out unsigned (3 downto 0);
      R2: out unsigned (3 downto 0));
      end ALUModified ;
ARCHITECTURE calculation of ALUModified  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"=> --1
      Result<= Reg4;
               Result(7) <= Reg1(0);
               Result(6) <= Reg1(1);
```



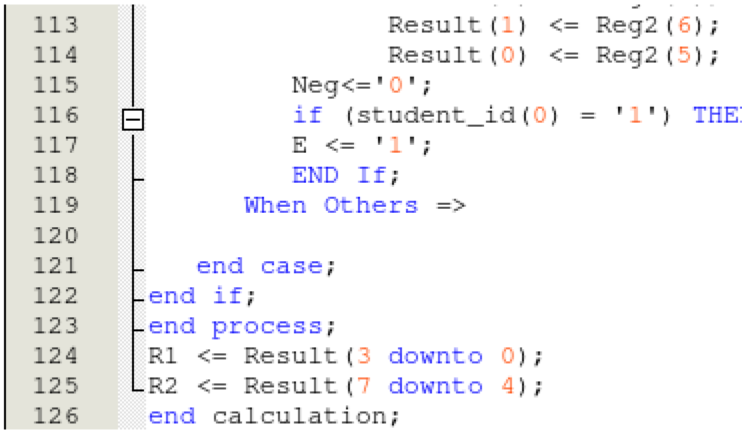
```
             Result(5) <= Reg1(2);
             Result(4) <= Reg1(3);
             Result(3) <= Reg1(4);
             Result(2) <= Reg1(5);
             Result(1) <= Reg1(6);
             Result(0) <= Reg1(7);
       Neg<='0';
       if (student_id(0) = '1') THEN
       E <= '1';
       END If;


    When "0000000000000010"=> --2
       Result <= Reg1 OR Reg2;
       Neg<='0';
       if (student_id(0) = '1') THEN
       E <= '1';
       END If;
    When "0000000000000100"=> --3
       if(5>Reg2) Then
       Result <= 5 - Reg2;
             Neg<='1';
          else
       Result <= Reg2 - 5 ;
                Neg<='0';
             end if;
             if (student_id(0) = '1') THEN
       E <= '1';
```



```
       END If;
    When "0000000000001000"=> --4
       Result <= NOT Reg1;
       Neg<='0';
       if (student_id(0) = '1') THEN
       E <= '1';
       END If;
    When "0000000000010000"=> --5
    Result<= Reg4;
             Result(7) <= Reg1(0);
             Result(6) <= Reg1(1);
             Result(5) <= Reg1(2);
             Result(4) <= Reg1(3);
             Result(3) <= Reg1(4);
             Result(2) <= Reg1(5);
             Result(1) <= Reg1(6);
             Result(0) <= Reg1(7);
       Neg<='0';
       if (student_id(0) = '1') THEN
       E <= '1';
       END If;
    When "0000000000100000"=> --6
       if(Reg2>Reg1) Then
       Result <= Reg2;
          else
       Result <= Reg1 ;
             end if;
       Neg<='0';
```

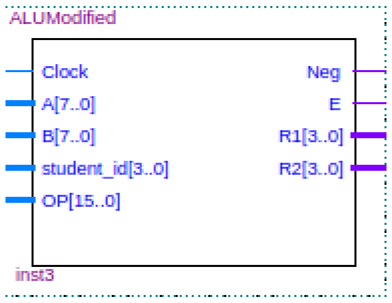


```
         if (student_id(0) = '1') THEN
         E <= '1';
         END If;
      When "0000000001000000"=> --7
         if(Reg2>Reg1) Then
         Result <= Reg2 - Reg1;
               Neg<='1';
            else
         Result <= Reg1 - Reg2 ;
                  Neg<='0';
               end if;
               if (student_id(0) = '1') THEN
         E <= '1';
         END If;
      When "0000000010000000"=> --8
         Result <= Reg1 XNOR Reg2;
         Neg<='0';
         if (student_id(0) = '1') THEN
         E <= '1';
         END If;
      When "0000000100000000"=> --9
      Result<= Reg4;
               Result(7) <= Reg2(4);
               Result(6) <= Reg2(3);
               Result(5) <= Reg2(2);
               Result(4) <= Reg2(1);
               Result(3) <= Reg2(0);
               Result(2) <= Reg2(7);
```



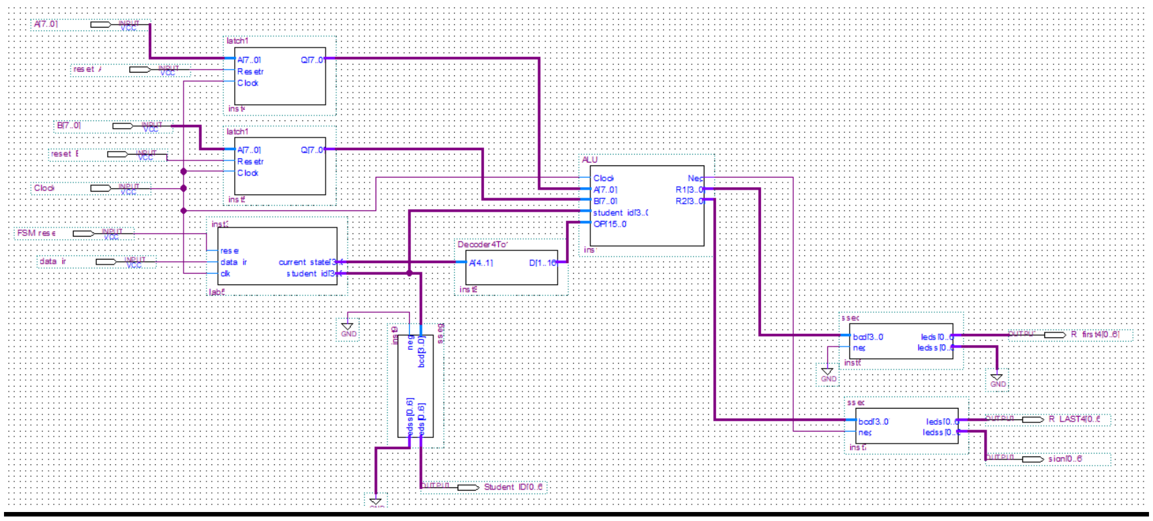
```
               Result(1) <= Reg2(6);
               Result(0) <= Reg2(5);
         Neg<='0';
         if (student_id(0) = '1') THEN
         E <= '1';
         END If;
      When Others =>

   end case;
end if;
end process;
R1 <= Result(3 downto 0);
R2 <= Result(7 downto 4);
end calculation;
```

Component's Circuit Diagram:

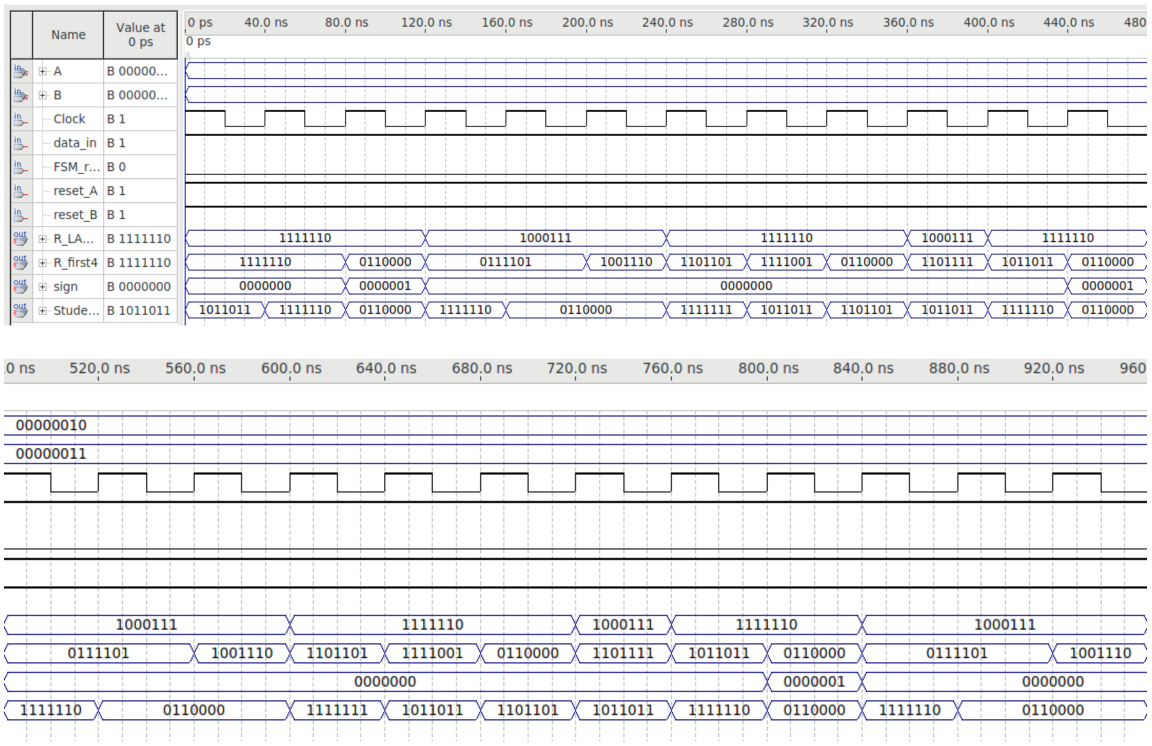


Circuit Diagrams

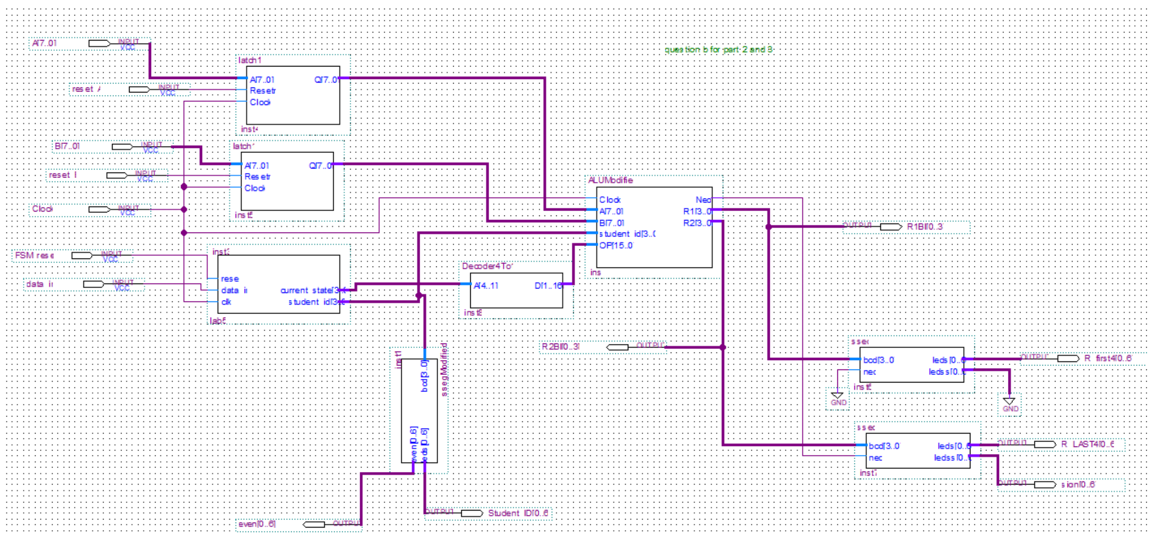
Circuit for part 1:



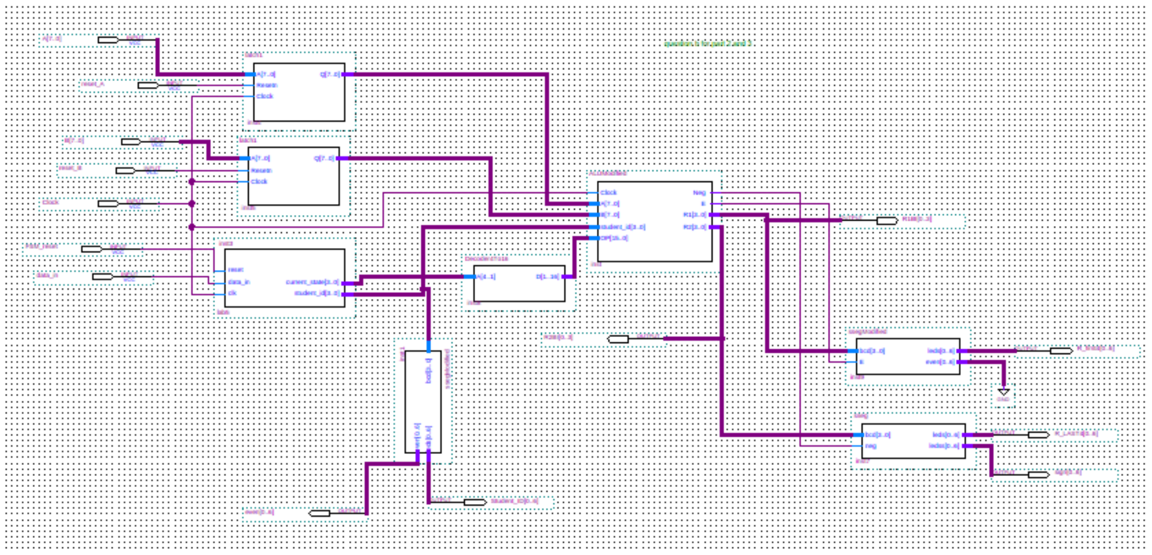
Waveform output for part 1:



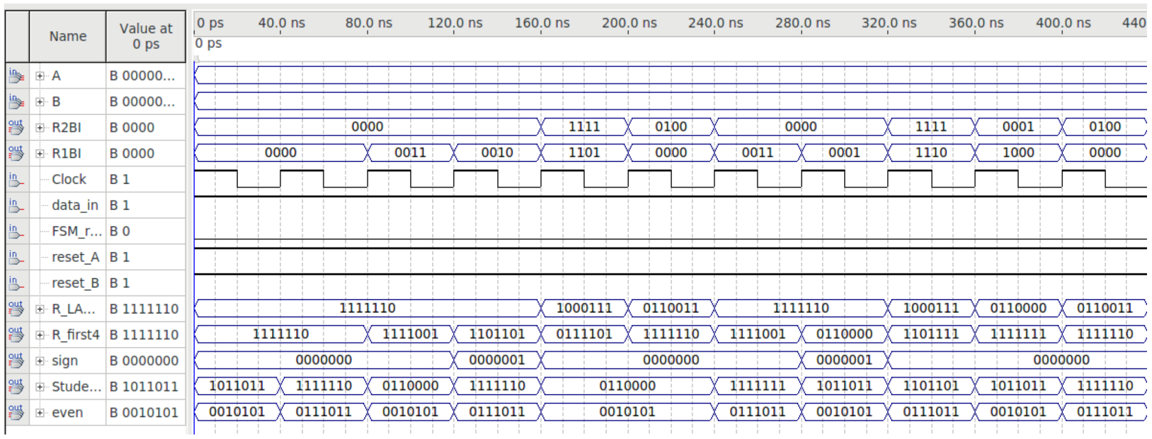
Circuit for part 2 and 3(method1):

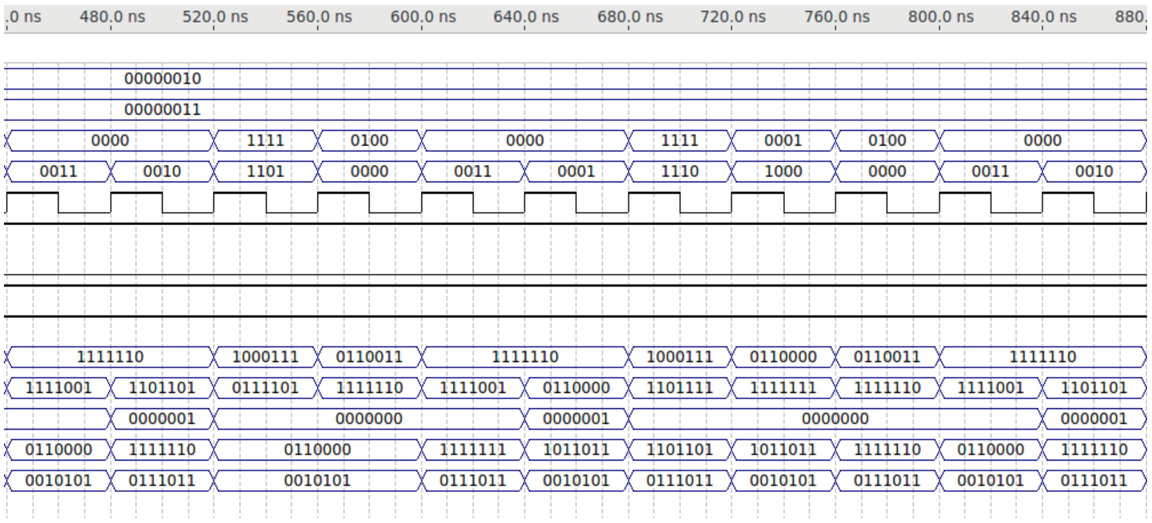


Circuit for part 2 and 3(method2):



Waveform output for part 2 and 3:





References

Brown, S. D., & Vranesic, Z. G. (2009). *Fundamentals of Digital Logic with VHDL Design*. New York, United States: McGraw-Hill Education.