Ryerson University

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

Course Title:	Digital Systems
Course Number:	COE328
Semester/Year (e.g.F2016)	F2023

Instructor:	Arghavan Asad
T.A	Seham Al Abdul Wahid

Assignment/Lab Number:	Lab 6
Assignment/Lab Title:	Design of a Simple General-Purpose Processor

Submission Date:	Monday, December 4, 2023
Due Date:	Monday, December 4, 2023

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Babar	**Umar**	**501116508**	**9**	[illegible]

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report 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/senate/current/pol60.pdf

Contents

Introduction

An essential part of the central processing unit, the ALU (Arithmetic Logic Unit) is dedicated to performing arithmetic and logical operations. Functioning with two latches and a control unit, each latch has two inputs and one output and acts as a storage unit. When the enable signal is active, these latches continuously sample inputs and temporarily store 8-bit values (A and B) that correspond to the final four digits of a nine-digit student ID. The data is then sent to the ALU. The control unit, which functions as the primary operations selector, plays an important role in identifying the microcode required by the ALU. The control unit consists of two sub-components: 4-to-16 decoders and the FSM (Finite State Machine), which is utilized as a model of computation to simulate sequential logic. At the same time, the decoder, a circuit that converts codes into signal sets, improves the control unit's overall efficiency.

Components: Latch1, Latch2, 4:16 Decoder, FSM

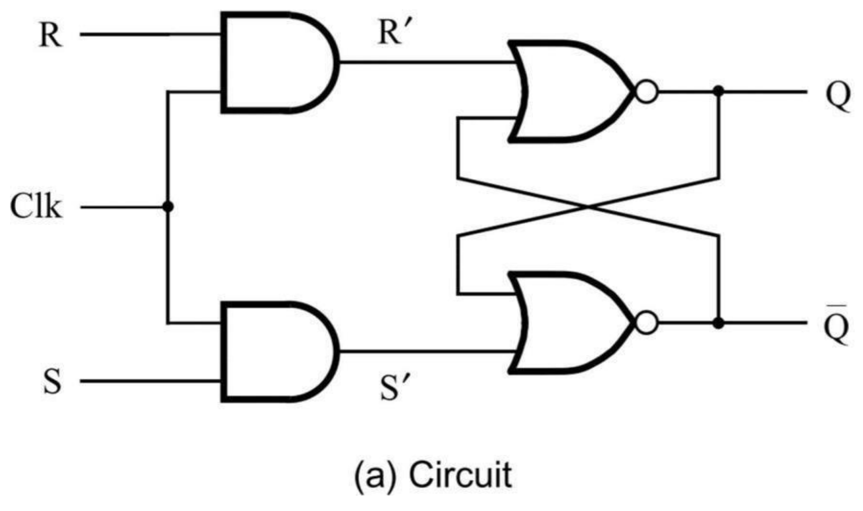
Description of components

Eight-bit input values are temporarily stored in latch1 and latch2 before being transmitted to the ALU for use in Boolean calculations. On consecutive rising edges, these latches record and output 8-bit Boolean values; a high reset input clears the stored values, setting the output to zero for every bit. They can store and output values again after resetting when the reset value decreases. The 4:16 decoder supplies the ALU with a 16-bit output that indicates the action to be taken. The 4:16 decoder receives the current state from the FSM, which up-counts from 0 to 8, and its output directs the ALU to choose one of its nine functions based on the inputs.

Latch1 & Latch2

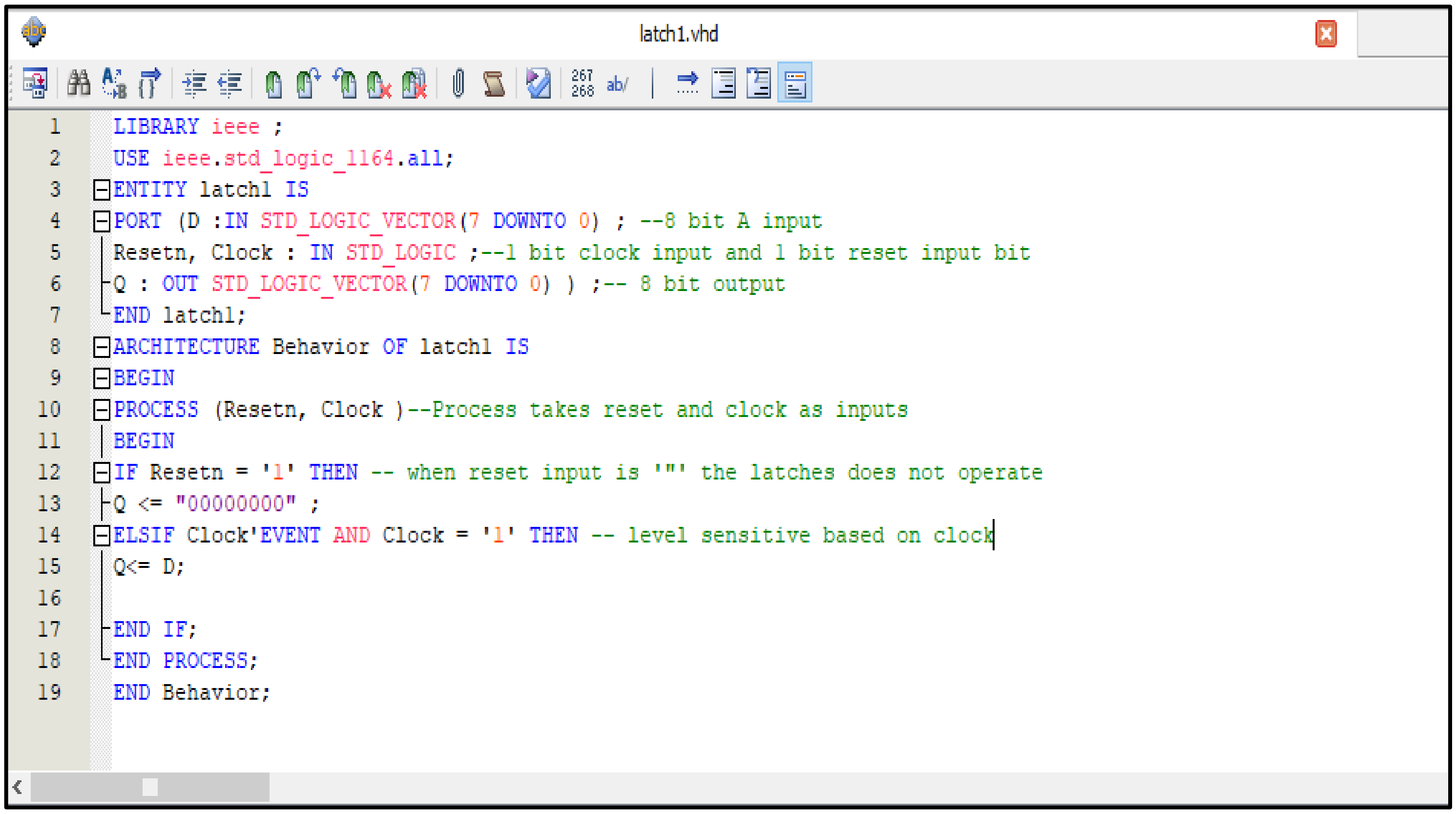
Clock	**Reset**	**Set**	**Q_n**	**Q_{n+1} (Output)**
0	X	X	X	No Change
1	0	0	X	No Change
1	0	1	X	1
1	1	0	X	0
1	1	1	X	X

Table 1: Truth Table for Gated SR Latch.

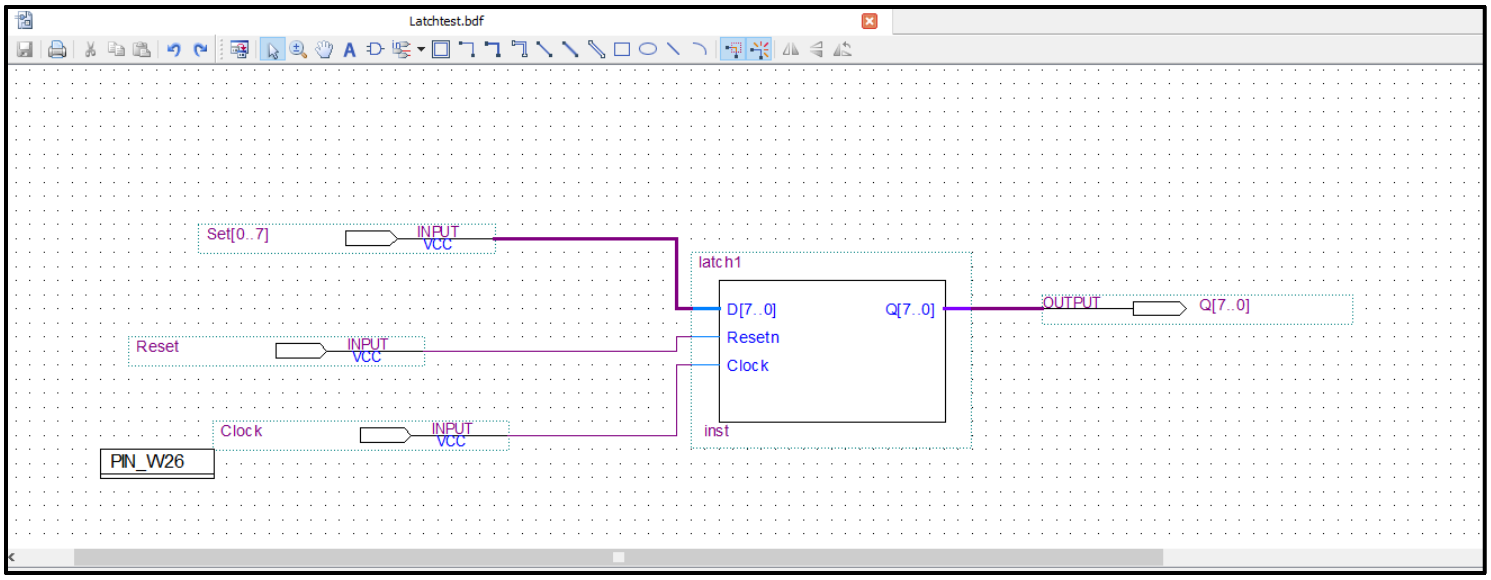


Gated SR-Latch Logic circuit Diagram

VHDL Code for Latch device



```
LIBRARY ieee ;
USE ieee.std_logic_1164.all;
ENTITY latch1 IS
PORT (D :IN STD_LOGIC_VECTOR(7 DOWNTO 0) ; --8 bit A input
Resetn, Clock : IN STD_LOGIC ;--1 bit clock input and 1 bit reset input bit
Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;-- 8 bit output
END latch1;
ARCHITECTURE Behavior OF latch1 IS
BEGIN
PROCESS (Resetn, Clock )--Process takes reset and clock as inputs
BEGIN
IF Resetn = '1' THEN -- when reset input is '"' the latches does not operate
Q <= "00000000" ;
ELSIF Clock'EVENT AND Clock = '1' THEN -- level sensitive based on clock
Q<= D;


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



Block diagram for Latch Pictured above

Waveform for Latch

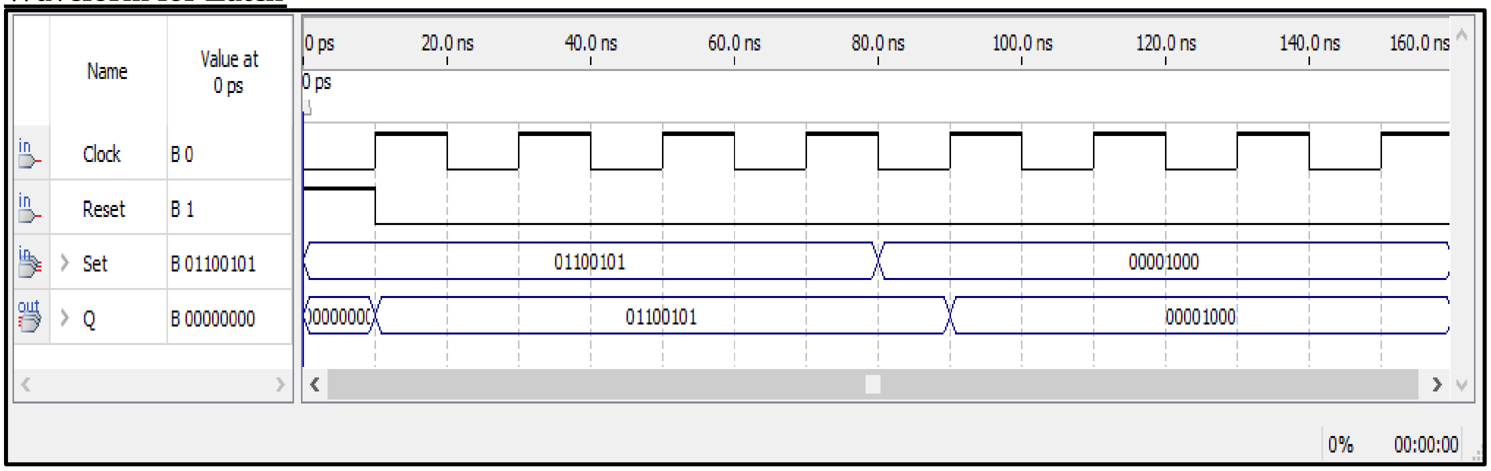


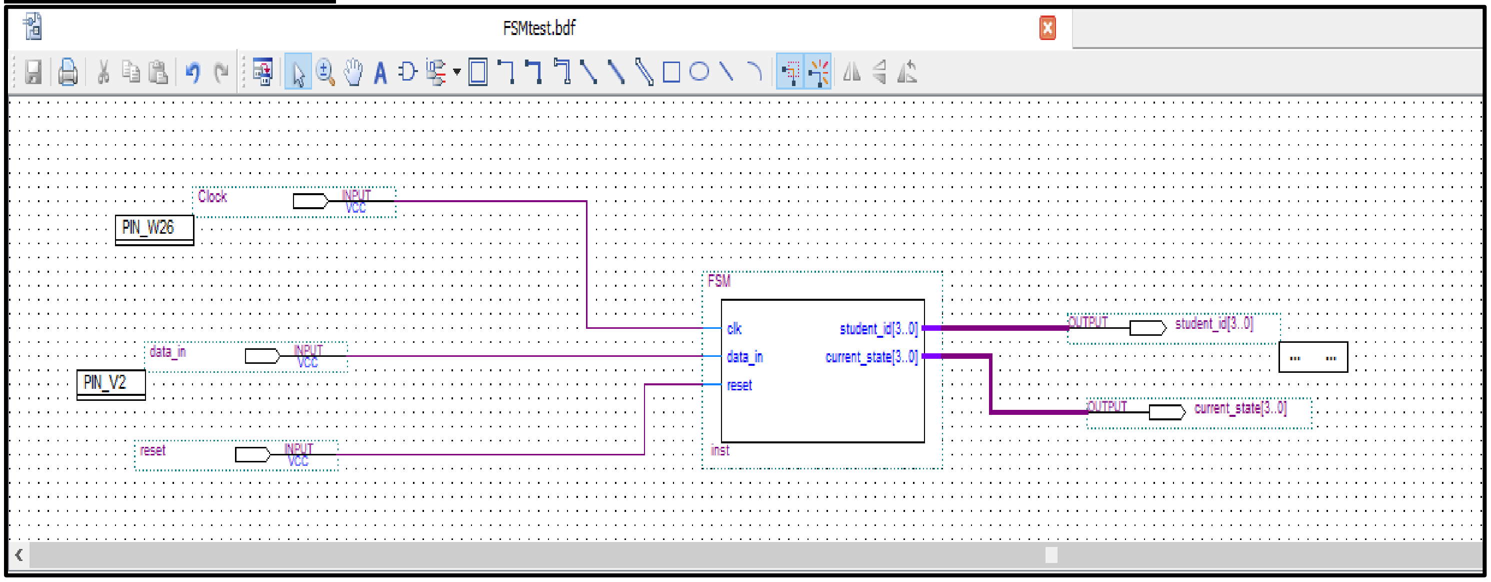
Figure 4: Waveform of “8-Input” Latch.

FSM

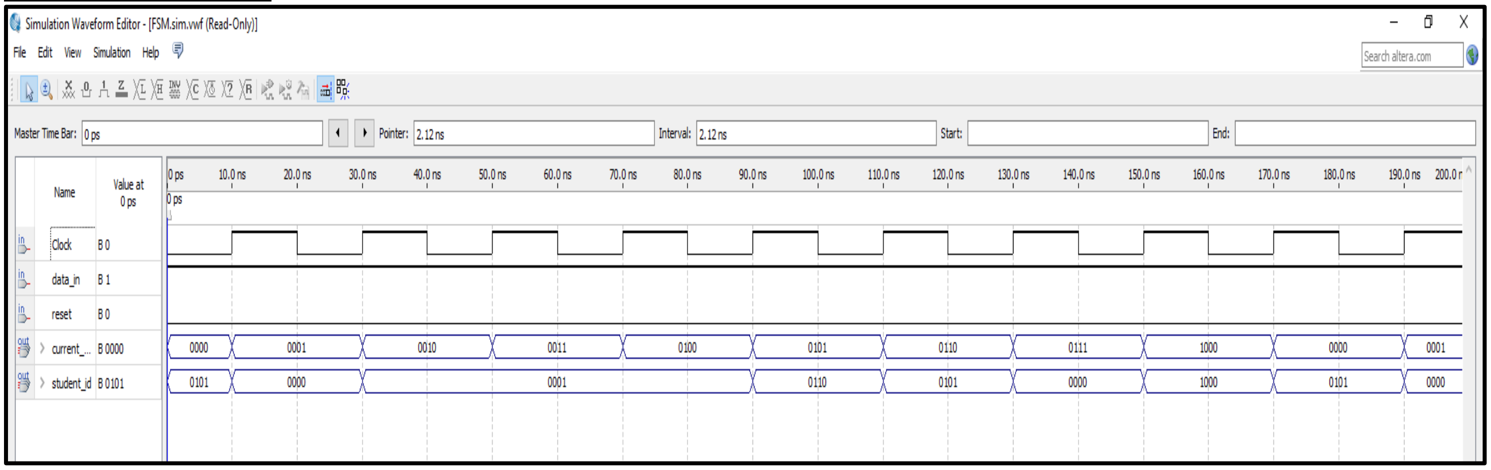
State Table for FSM

Present		Next							
		Data_in = 0				Data_in = 1			
State	Current_state	State	Current_state	Student_id		State	Current_state	Student_id	
S0	0000	S0	0000	5	0101	S1	0001	0	0000
S1	0001	S1	0001	0	0000	S2	0010	1	0001
S2	0010	S2	0010	1	0001	S3	0011	1	0001
S3	0011	S3	0011	1	0001	S4	0100	1	0001
S4	0100	S4	0100	1	0001	S5	0101	6	0110
S5	0101	S5	0101	6	0110	S6	0110	5	0101
S6	0110	S6	0110	5	0101	S7	0111	0	0000
S7	0111	S7	0111	0	0000	S8	1000	8	1000
S8	1000	S8	1000	8	1000	S0	0000	5	0101

Block Diagram for FSM



Waveform for FSM

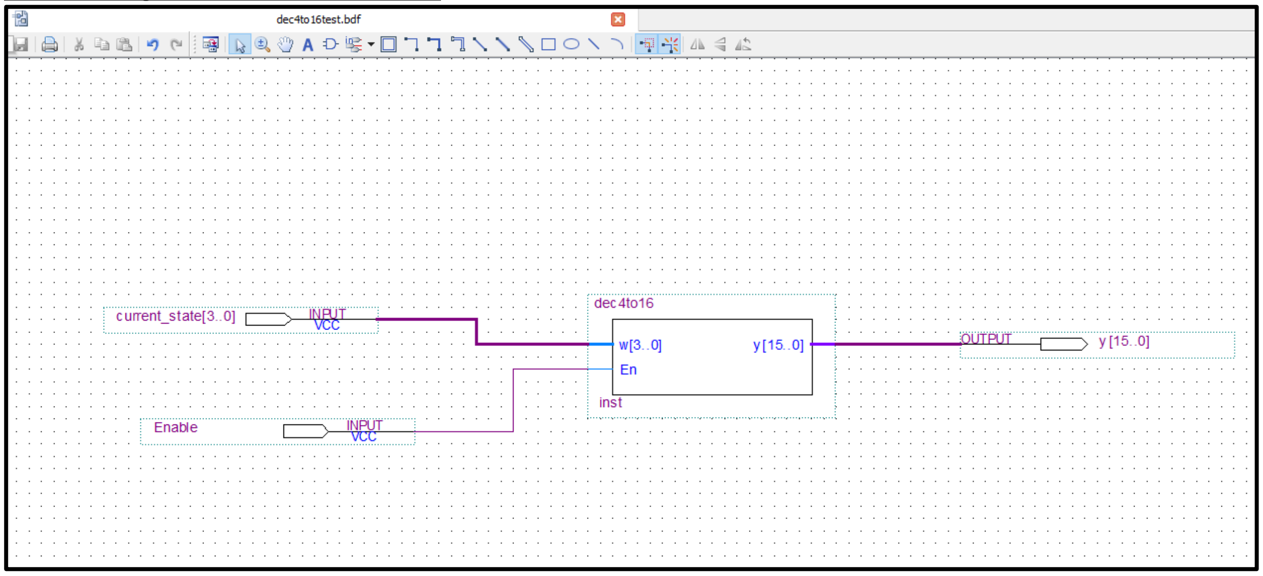


4:16 Decoder

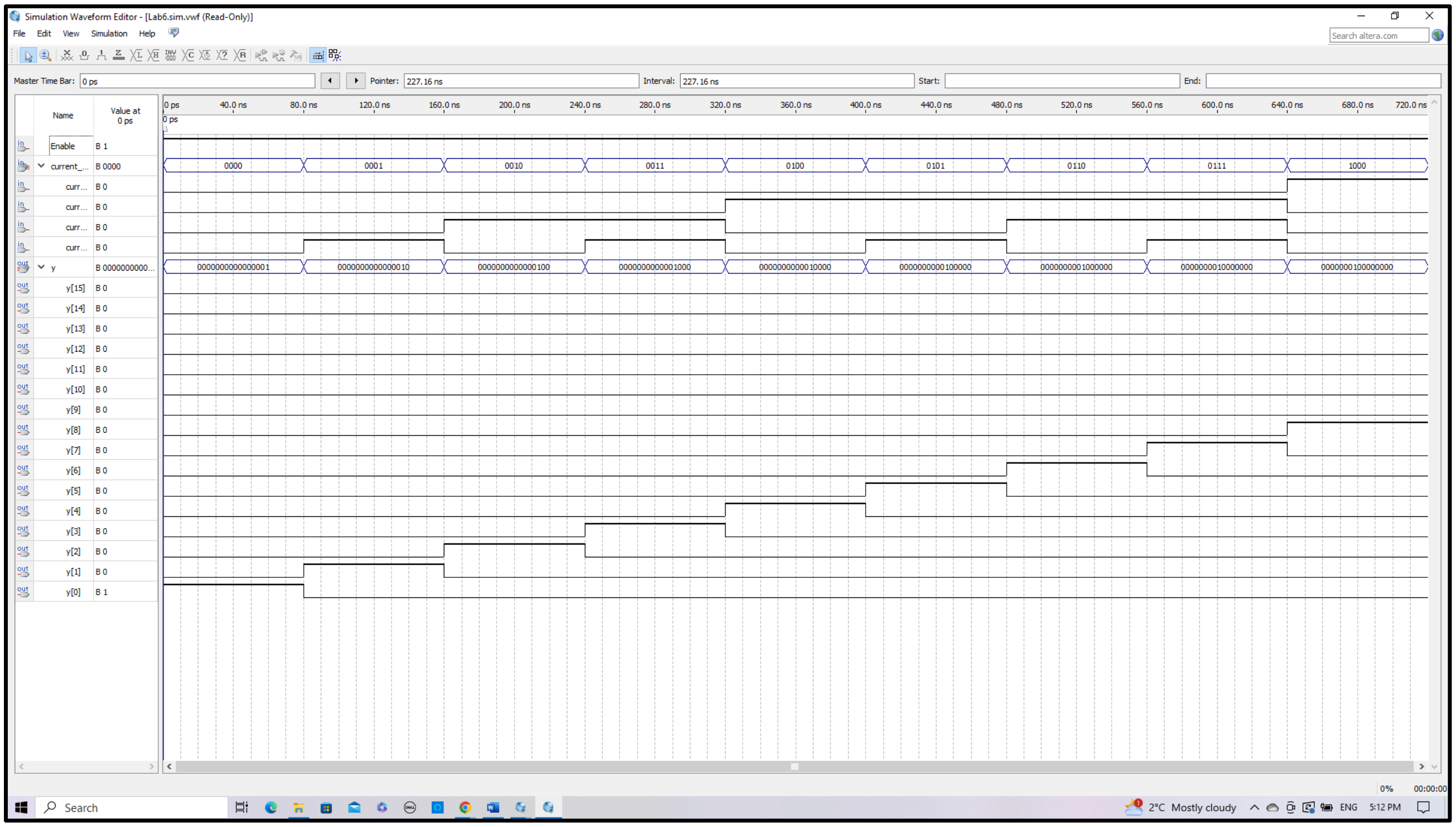
En	W3	W2	W1	W0	Cout														
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Truth table for 4:16 Decoder.

Block Diagram for 4:16 decoder



Waveform for 4:16 Decoder


Simulation Waveform Editor - [Lab6.sim.vwf (Read-Only)]
Master Time Bar: 0 ps
Pointer: 227.16 ns
Interval: 227.16 ns
Enable B 1
current_ B 0000
0000 0001 0010 0011 0100 0101 0110 0111 1000
y B 0000000000...
0000000000000001 0000000000000010 0000000000000100 0000000000001000 0000000000010000 0000000000100000 0000000001000000 0000000010000000 0000000100000000
y[15] y[14] y[13] y[12] y[11] y[10] y[9] y[8] y[7] y[6] y[5] y[4] y[3] y[2] y[1] y[0]
0 ps 40.0 ns 80.0 ns 120.0 ns 160.0 ns 200.0 ns 240.0 ns 280.0 ns 320.0 ns 360.0 ns 400.0 ns 440.0 ns 480.0 ns 520.0 ns 560.0 ns 600.0 ns 640.0 ns 680.0 ns 720.0 ns

ALU_1

The purpose of the ALU is to carry out mathematical operations on binary numbers. ALU_1 is the core arithmetic logic unit for Problem 1. Additionally, the ALU core accepts two 8-bit inputs (A and B) from the memory unit, along with one 16-bit input from the 4-16 decoder. Each time the clock signal rises, the nine distinct functions in question are performed on the inputs provided by each of the two latches (registers). However, for each of the 7-segment LED displays, the solution outputs two 4-bit buses and also outputs the negative bit (sign) if a negative value is output. Additionally, this problem reads all the digits of the student ID card in the specified order, which is not used in the first problem. The expected output and microcode for the function corresponding to the student number 501116508 are shown in the table below. Overall, the expected output is determined using inputs A and B. These inputs are determined by taking the last four digits and converting them to two hexadecimal numbers. Here, A is (65)16 and B is (47)16.

Purpose of Inputs and Outputs of this component

Inputs

• Clock: Signal transmitted on the rising edge of clock cycles, the clock signal causes the ALU to change or maintain its output signal in accordance with the input signal values during transition.

• A[7..0]: Denotes the 8-bit input A that was read from the storage unit; A is the binary representation of the student ID's sixth and seventh digits.

• B[7..0]: This points to the 8-bit input B that was taken from the storage unit. The binary representation of B is the student ID's final two digits.

• OP[15..0]: The control unit serves as an operation selector by sending OP to the ALU. This input is used by the ALU to choose a particular operation from the microcode provided by the decoder.

Outputs

• Neg: Indicates and highlights any potential system overflows.

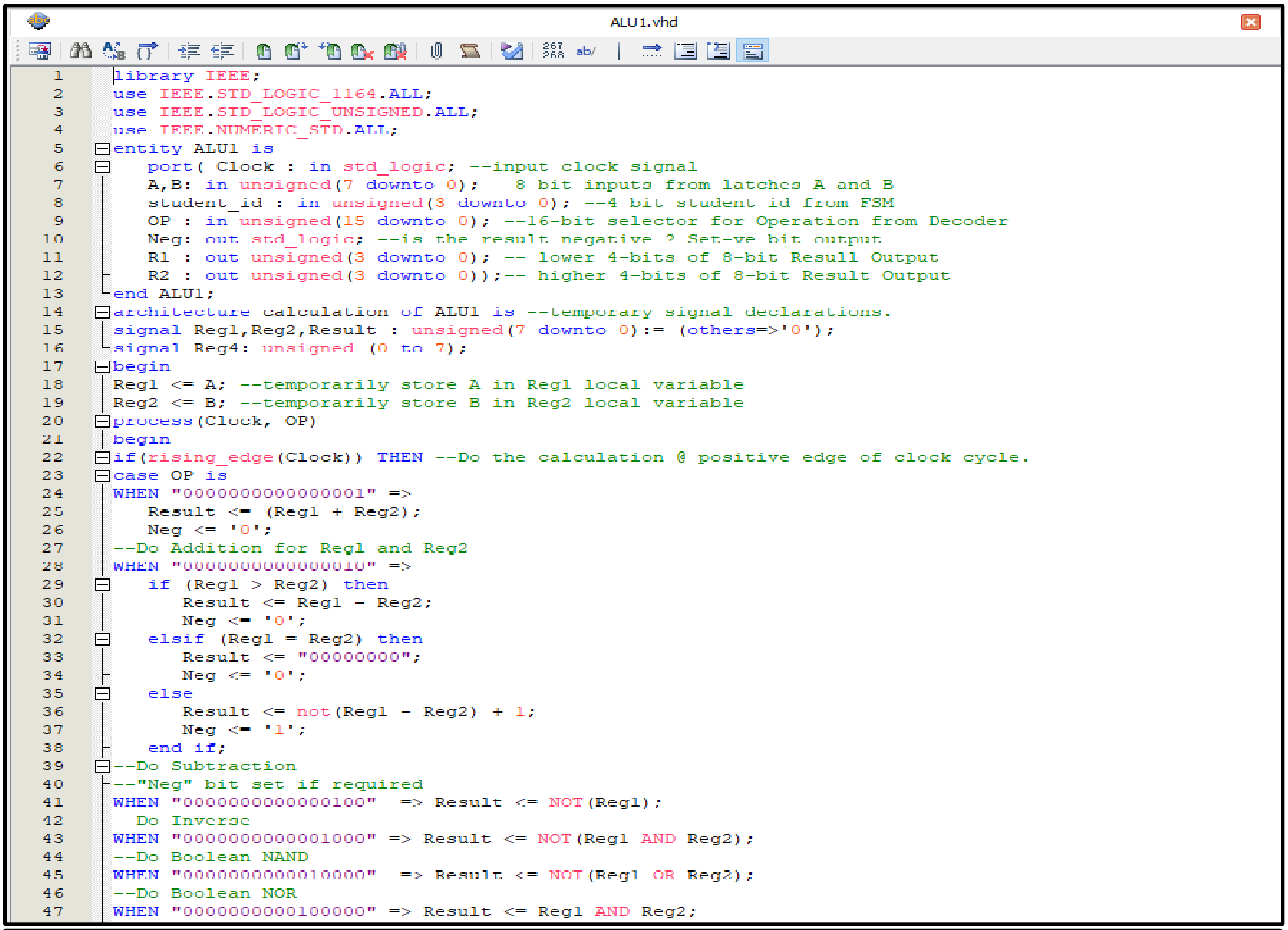
• R1[3..0] Display: It sends the initial four bits of the eight-bit result to the seven-segment display from the Boolean operations of A and B.

• R2[3..0] Display: It sends to the 7-segment display the last 4 bits of the 8-bit result acquired from the Boolean operations of A and B.

Function	**F#**	**Microcode**	sign	Output (8- bit binary)	Output(Hex)	**Hex2 (disp)**	**Hex1 (R_last_four) (disp)**	**Hex0 (R_First_four) (disp)**
Sum(A, B)	**1**	**0000000000000001**	+	0110 1101	6 D	**0000000**	**1011111**	**0111101**
Diff(A, B)	**2**	**0000000000000010**	+	0101 1101	5 D	**0000000**	**1011011**	**0111101**
NOT(A)	**3**	**0000000000000100**	+	1001 1010	9 A	**0000000**	**1111011**	**1110111**
(A NAND B)	**4**	**0000000000001000**	+	1111 1111	F F	**0000000**	**1000111**	**1000111**
(A NOR B)	**5**	**0000000000010000**	+	1001 0010	9 2	**0000000**	**1111011**	**1101101**
A AND B	**6**	**0000000000100000**	+	0000 0000	0 0	**0000000**	**1111110**	**1111110**
A XOR B	**7**	**0000000001000000**	+	0110 1101	6 D	**0000000**	**1011111**	**0111101**
A OR B	**8**	**0000000010000000**	+	0110 1101	6 D	**0000000**	**1011111**	**0111101**
(A XNOR B)	**9**	**0000000100000000**	+	1001 0010	9 2	**0000000**	**1111011**	**1101101**

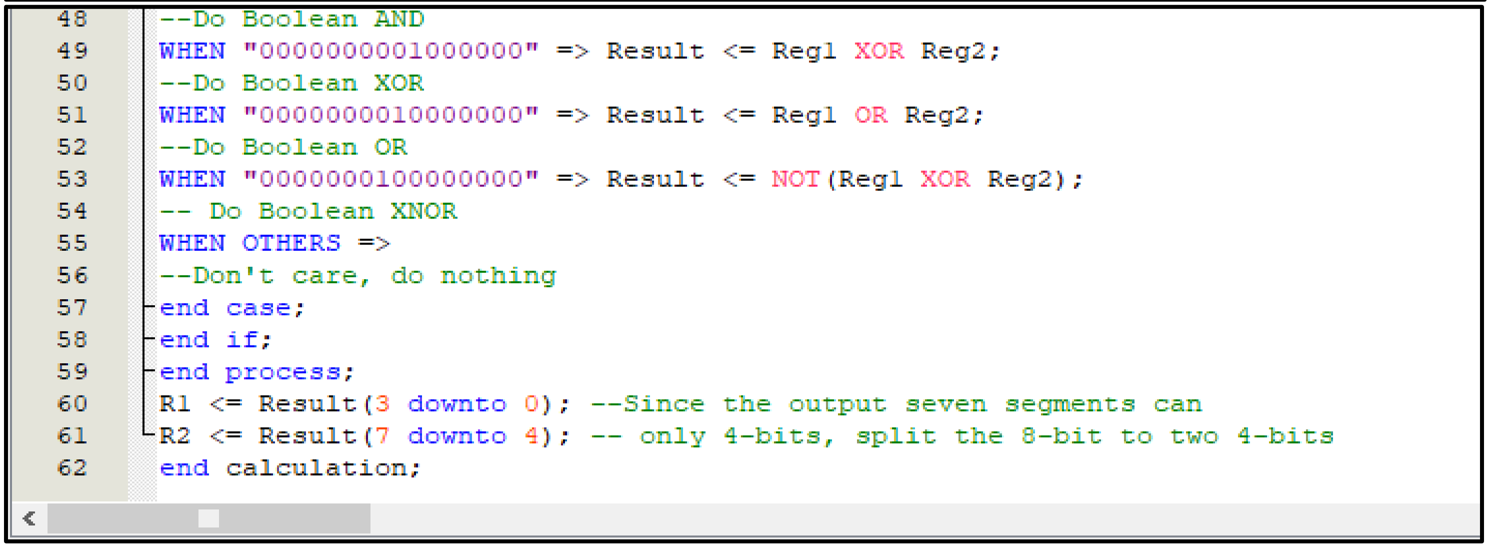
Table of microcodes (generated by the decoder), along with expected output and Functions for ALU_1 (for problem set 1) pictured above

VHDL code for ALU_1

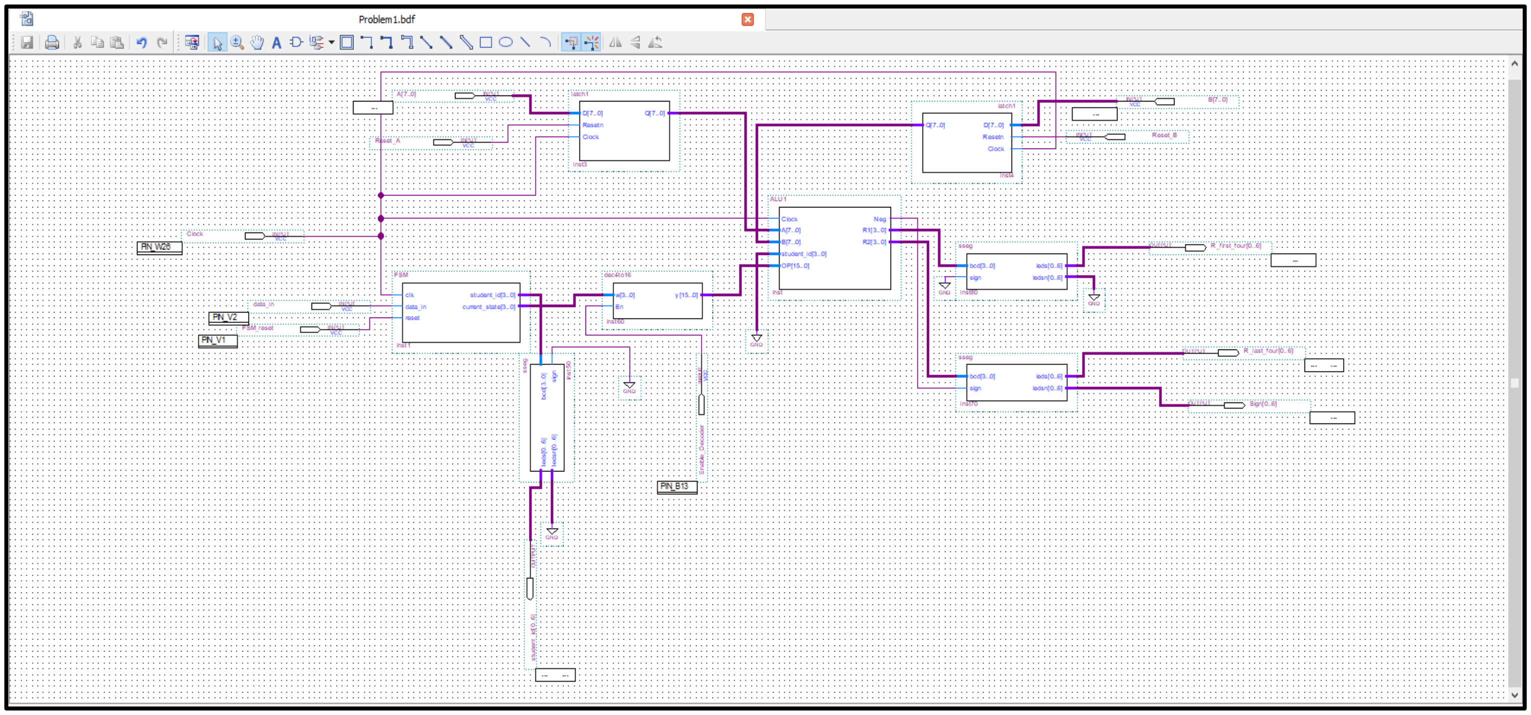


ALU1.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
entity ALU1 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 Resull Output
   R2 : out unsigned(3 downto 0));-- higher 4-bits of 8-bit Result Output
end ALU1;
architecture calculation of ALU1 is --temporary signal declarations.
signal Reg1,Reg2,Result : unsigned(7 downto 0):= (others=>'0');
signal Reg4: unsigned (0 to 7);
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);
   Neg <= '0';
--Do Addition for Reg1 and Reg2
WHEN "0000000000000010" =>
   if (Reg1 > Reg2) then
      Result <= Reg1 - Reg2;
      Neg <= '0';
   elsif (Reg1 = Reg2) then
      Result <= "00000000";
      Neg <= '0';
   else
      Result <= not(Reg1 - Reg2) + 1;
      Neg <= '1';
   end if;
--Do Subtraction
--"Neg" bit set if required
WHEN "0000000000000100"  => Result <= NOT(Reg1);
--Do Inverse
WHEN "0000000000001000" => Result <= NOT(Reg1 AND Reg2);
--Do Boolean NAND
WHEN "0000000000010000"  => Result <= NOT(Reg1 OR Reg2);
--Do Boolean NOR
WHEN "0000000000100000" => Result <= Reg1 AND Reg2;
```

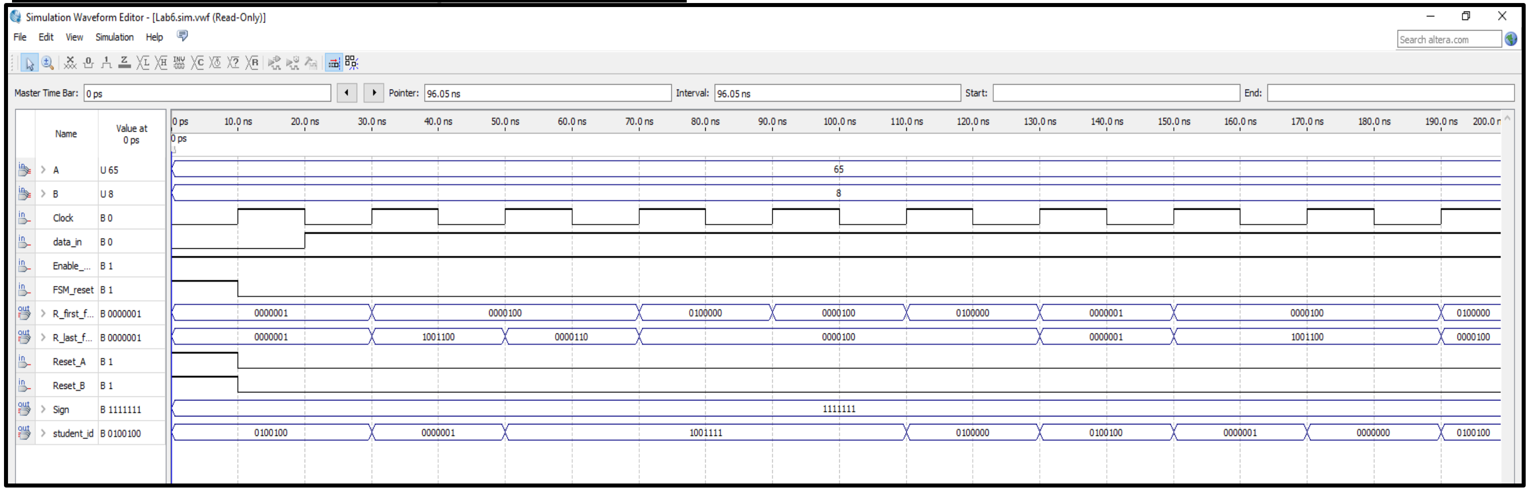


```vhdl
--Do Boolean AND
WHEN "0000000001000000" => Result <= Reg1 XOR Reg2;
--Do Boolean XOR
WHEN "0000000010000000" => Result <= Reg1 OR Reg2;
--Do Boolean OR
WHEN "0000000100000000" => Result <= NOT(Reg1 XOR Reg2);
-- Do Boolean XNOR
WHEN OTHERS =>
--Don't care, do nothing
end case;
end if;
end process;
R1 <= Result(3 downto 0); --Since the output seven segments can
R2 <= Result(7 downto 4); -- only 4-bits, split the 8-bit to two 4-bits
end calculation;
```



Block Diagram for Problem 1 (with alu_1 block) pictured above

Waveform for Problem 1 (using FSM student ID)



ALU_2

Purpose and Design of the component

The purpose of the ALU is to carry out mathematical operations on binary numbers. ALU_1 is the core arithmetic logic unit for Problem 1. Additionally, the ALU core accepts two 8-bit inputs (A and B) from the memory unit, along with one 16-bit input from the 4-16 decoder. Each time the clock signal rises, the nine distinct functions in question are performed on the inputs provided by each of the two latches (registers). A set of operations (stated in the table below) was added to the ALU code and design used in the previous problem. The ALU core produces an overall 8-bit output, which is displayed on a seven-segment display.

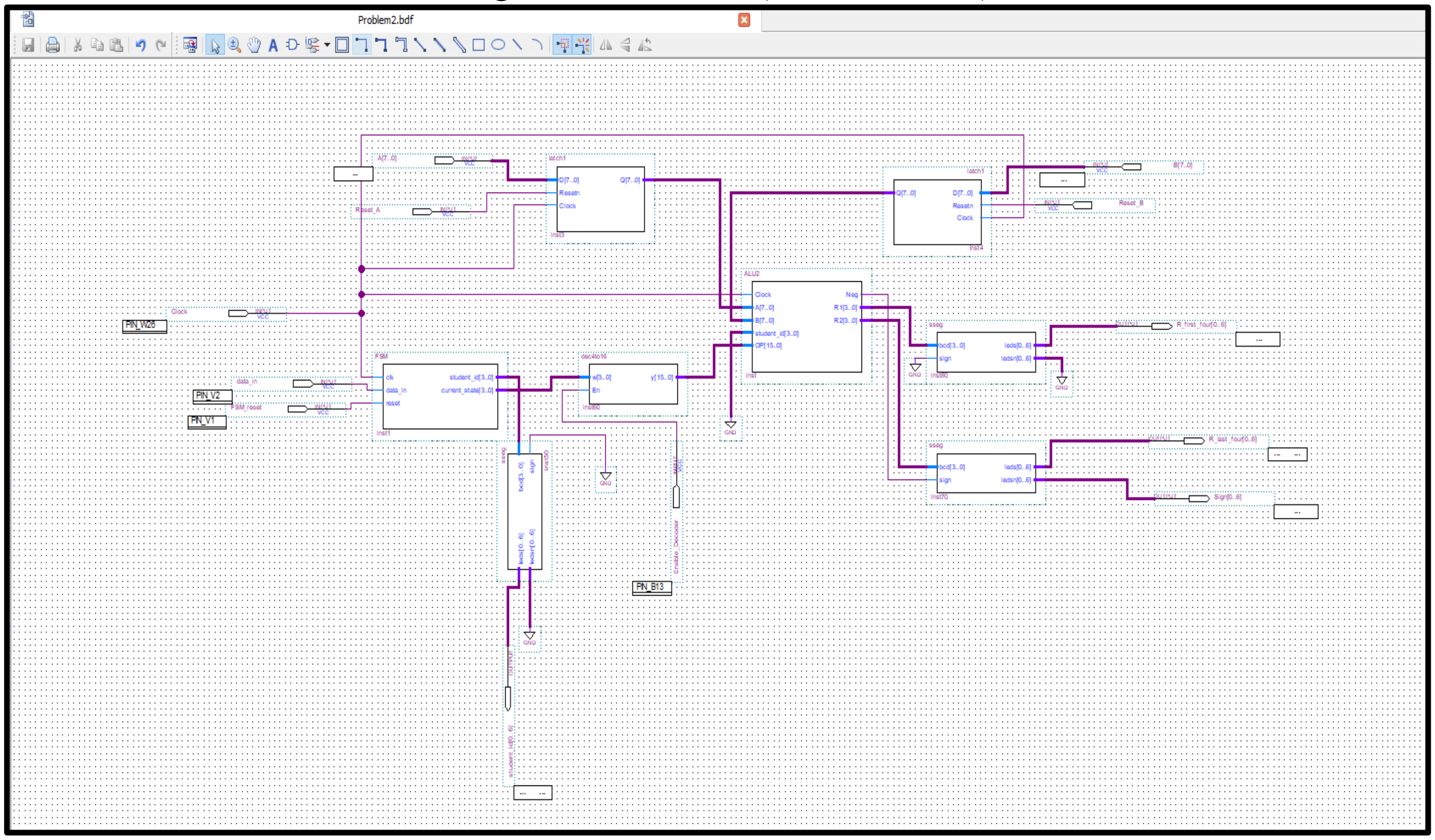
Purpose of Inputs and Outputs of this component

The inputs and outputs for Alu_2 is the same as the ones used in Alu_1. See the purpose of the the inputs and outputs of the component section in ALU 1 for a description on these inputs and outputs

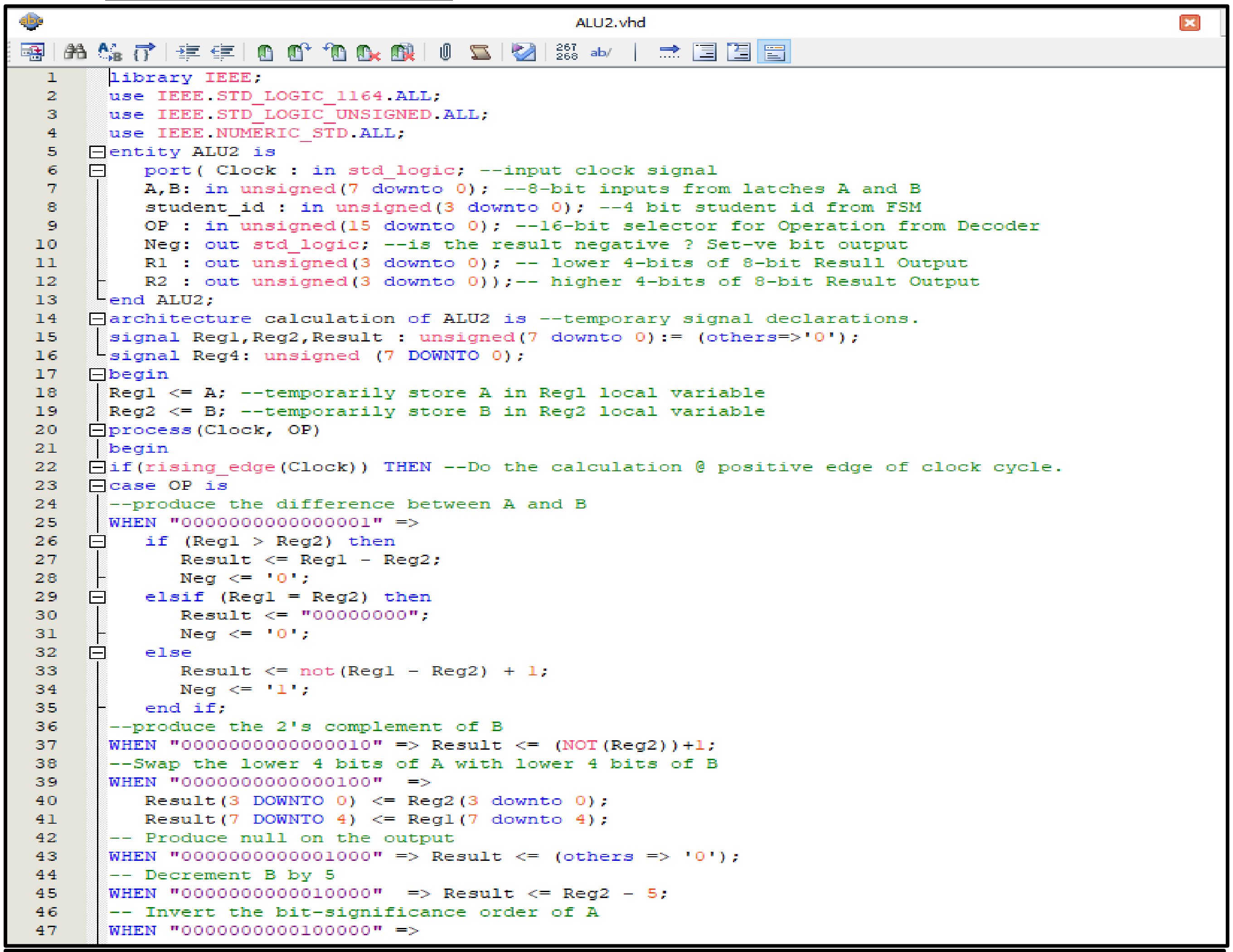
Table of microcodes (generated by the decoder), along with expected output and Functions for ALU_2

Function	#	Microcode	sign	Output (8-bit binary)	Out (Hex)	Hex2 (disp)	Hex1 (R_last_four) (disp)	Hex0 (R_First_four) (disp)
Produce difference between A and B (A-B)	1	0000000000000001	+	0101 1101	5 D	0000000	1011011	0111101
Produce 2's complement of B	2	0000000000000010	+	1111 1000	F 8	0000000	1000111	1111111
Swap the lower 4-bits of A with lower 4 bits of B	3	0000000000000100	+	0110 1001	6 9	0000000	1011111	1111011
Produce null on the output	4	0000000000001000	+	0000 0000	0 0	0000000	1111110	1111110
Decrement B by 5	5	0000000000010000	+	0000 0011	0 3	0000000	1111110	1111001
Invert the bit-significance order of A	6	0000000000100000	+	1010 0110	A 6	0000000	1110111	1011111
Shift B to left by 3 bits, input bit = 1 (SHL)	7	0000000001000000	+	0100 0000	4 0	0000000	0110011	1111110
Increment A by 3	8	0000000010000000	+	0110 1000	6 8	0000000	1011111	1111111
Invert all bits of B	9	0000000100000000	+	1111 0111	F 7	0000000	1000111	1110000

Block Diagram for Problem 2 (with alu_2 block)

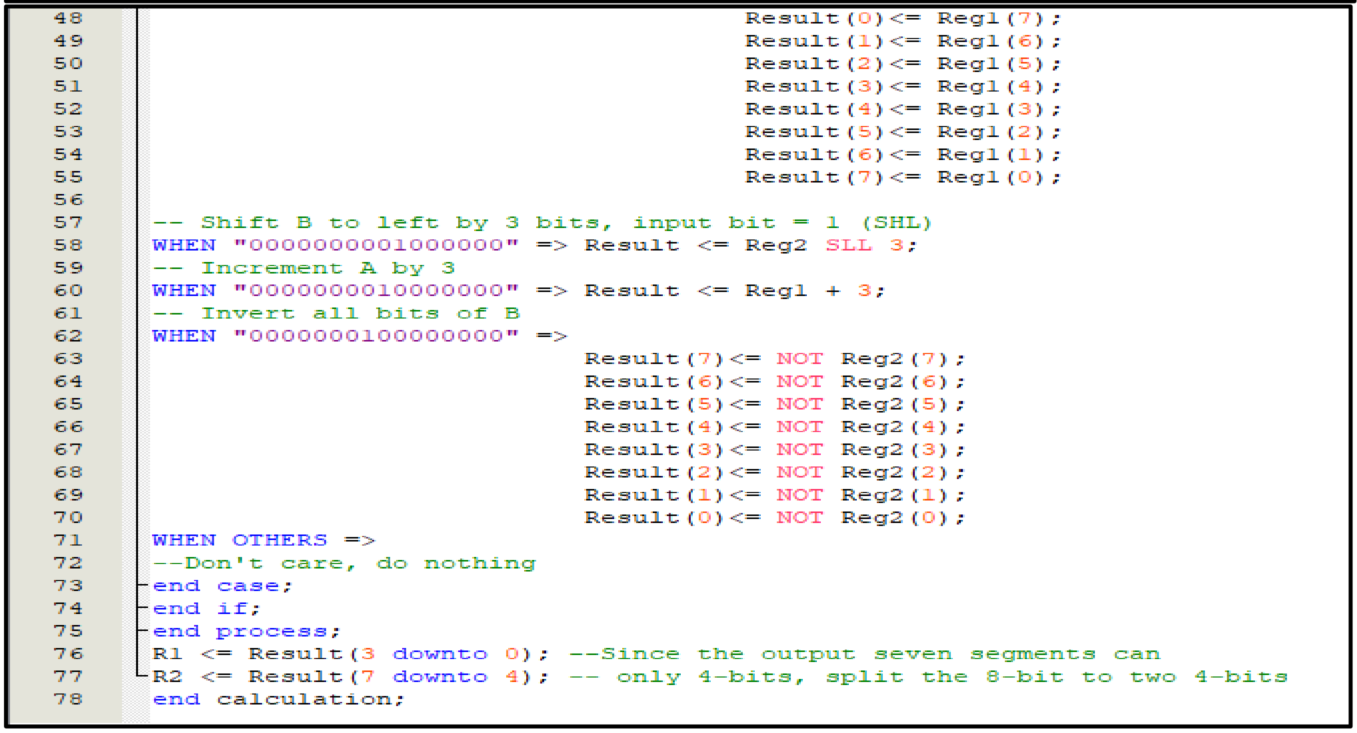


VHDL code for ALU_2 core

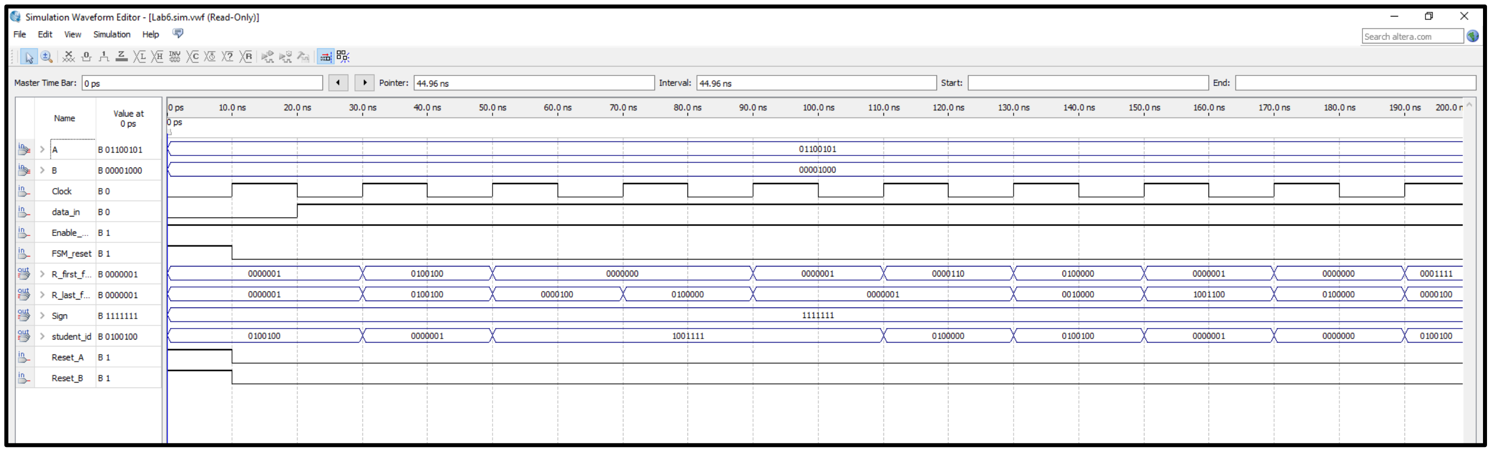


ALU2.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
entity ALU2 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 Resull Output
    R2 : out unsigned(3 downto 0));-- higher 4-bits of 8-bit Result Output
end ALU2;
architecture calculation of ALU2 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
--produce the difference between A and B
WHEN "0000000000000001" =>
    if (Reg1 > Reg2) then
        Result <= Reg1 - Reg2;
        Neg <= '0';
    elsif (Reg1 = Reg2) then
        Result <= "00000000";
        Neg <= '0';
    else
        Result <= not(Reg1 - Reg2) + 1;
        Neg <= '1';
    end if;
--produce the 2's complement of B
WHEN "0000000000000010" => Result <= (NOT(Reg2))+1;
--Swap the lower 4 bits of A with lower 4 bits of B
WHEN "0000000000000100"  =>
    Result(3 DOWNTO 0) <= Reg2(3 downto 0);
    Result(7 DOWNTO 4) <= Reg1(7 downto 4);
-- Produce null on the output
WHEN "0000000000001000" => Result <= (others => '0');
-- Decrement B by 5
WHEN "0000000000010000"  => Result <= Reg2 - 5;
-- Invert the bit-significance order of A
WHEN "0000000000100000" =>
```



```vhdl
                                        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);

-- Shift B to left by 3 bits, input bit = 1 (SHL)
WHEN "0000000001000000" => Result <= Reg2 SLL 3;
-- Increment A by 3
WHEN "0000000010000000" => Result <= Reg1 + 3;
-- Invert all bits of B
WHEN "0000000100000000" =>
                            Result(7)<= NOT Reg2(7);
                            Result(6)<= NOT Reg2(6);
                            Result(5)<= NOT Reg2(5);
                            Result(4)<= NOT Reg2(4);
                            Result(3)<= NOT Reg2(3);
                            Result(2)<= NOT Reg2(2);
                            Result(1)<= NOT Reg2(1);
                            Result(0)<= NOT Reg2(0);
WHEN OTHERS =>
--Don't care, do nothing
end case;
end if;
end process;
R1 <= Result(3 downto 0); --Since the output seven segments can
R2 <= Result(7 downto 4); -- only 4-bits, split the 8-bit to two 4-bits
end calculation;
```



Master Time Bar: 0 ps | Pointer: 44.96 ns | Interval: 44.96 ns | Start: | End:

Name	Value at 0 ps
A	B 01100101
B	B 00001000
Clock	B 0
data_in	B 0
Enable_...	B 1
FSM_reset	B 1
R_first_f...	B 0000001
R_last_f...	B 0000001
Sign	B 1111111
student_id	B 0100100
Reset_A	B 1
Reset_B	B 1

Waveform for Problem 2 (using FSM) pictured above

ALU_3

Purpose and design of Component

In contrast to the previous ALUs, ALU_3 makes use of the student ID output that the FSM has implemented. With that said, the FSM, a component of the control unit, provides the Student ID output, and the ALU core receives two inputs (A and B) from the storage unit. The ALU design has been modified such that it now examines the four-bit binary value for each microcode instruction (each digit in the student ID) and determines whether it contains odd parity. If so, it displays "y" for "yes" and "n" for "no." Furthermore, since this problem does not require the other bus utilized by the other two ALUs, it is grounded as a result. The seven-segment design was also modified so that ‘y’ or ‘n’ is displayed on the seven-segment display.

Purpose of all inputs and Outputs of this component

Inputs

- Clock: Signal transmitted on the rising edge of clock cycles, the clock signal causes the ALU to change or maintain its output signal in accordance with the input signal values during transition.
- A[7..0]: Denotes the 8-bit input A that was read from the storage unit; A is the binary representation of the student ID's sixth and seventh digits.
- B[7..0]: This points to the 8-bit input B that was taken from the storage unit. The binary representation of B is the student ID's final two digits.
- OP[15..0]: The control unit serves as an operation selector by sending OP to the ALU. This input is used by the ALU to choose a particular operation from the microcode provided by the decoder.
- Student_id[3..0]: The student ID is generated by the FSM and transmitted to the ALU as input so that it can carry out the assigned task.

Outputs

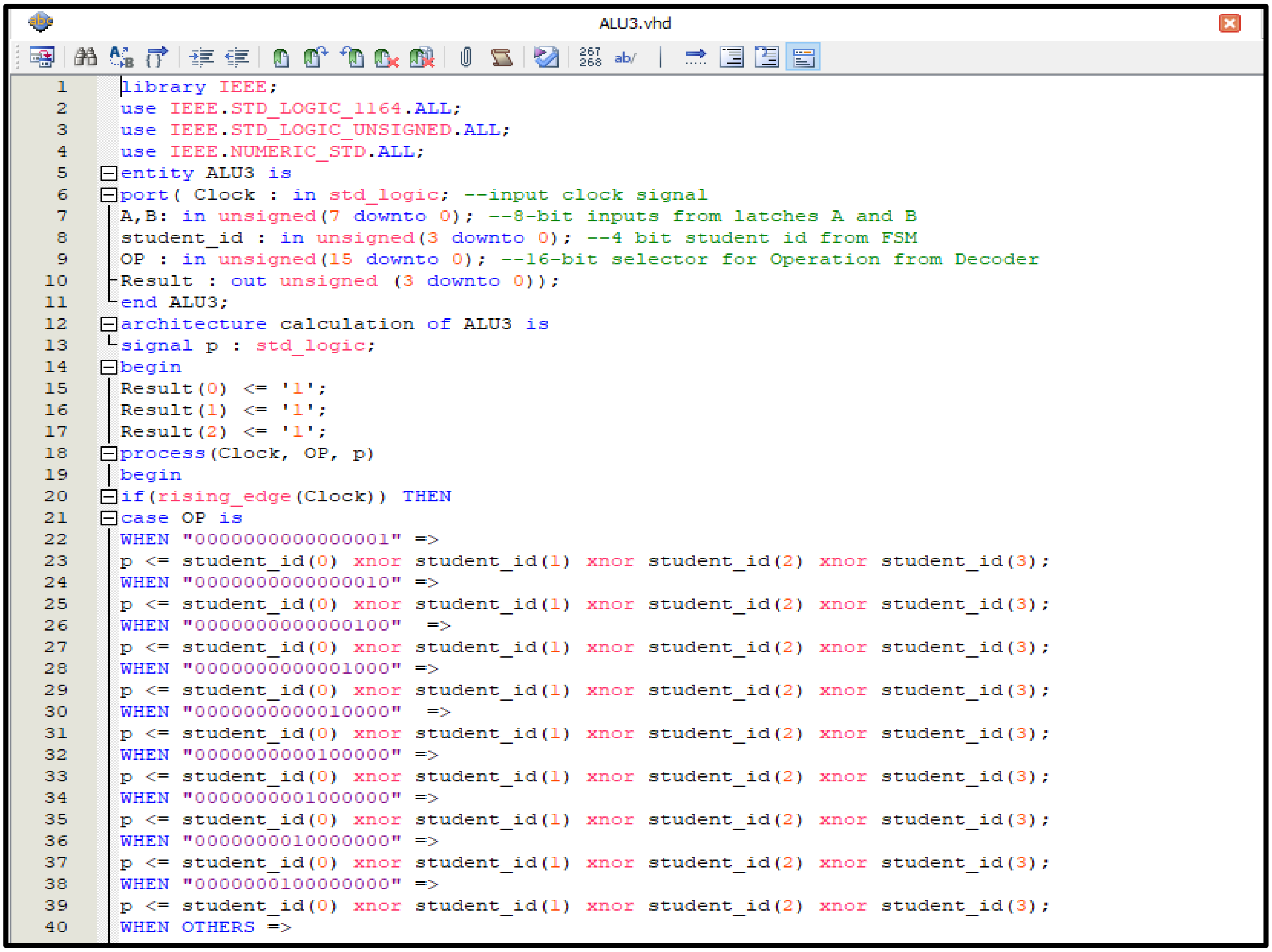
Result[3..0]: The modified 7-segment display receives a 4-bit output. This 4-bit representation indicates either 'y' or 'n,' depending on which ALU operations are performed in connection with the function that has been provided.

Table of Microcodes generated by decoder for ALU_

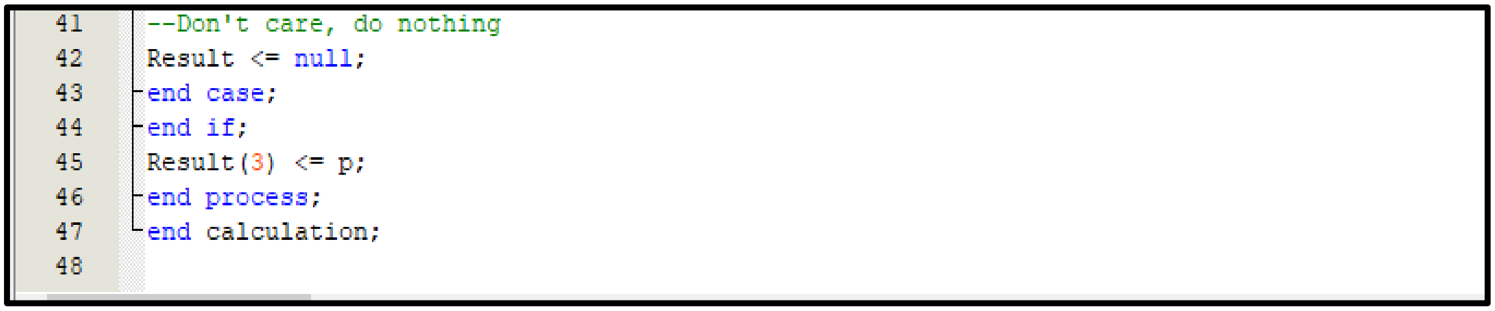
d#	Microcode	Digit	Output (bin)	Odd parity?	Hex0(disp) (R_last_four)
1	0000000000000001	5	0101	No	0010101
2	0000000000000010	0	0000	No	0010101
3	0000000000000100	1	0001	Yes	0111011
4	0000000000001000	1	0001	Yes	0111011
5	0000000000010000	1	0001	Yes	0111011
6	0000000000100000	6	0110	No	0010101
7	0000000001000000	5	0101	No	0010101
8	0000000010000000	0	0000	No	0010101
9	0000000100000000	8	1000	Yes	0111011

Both the ALU core and seven segment LED decoder have modified code for problem set 3.
Both are shown below:

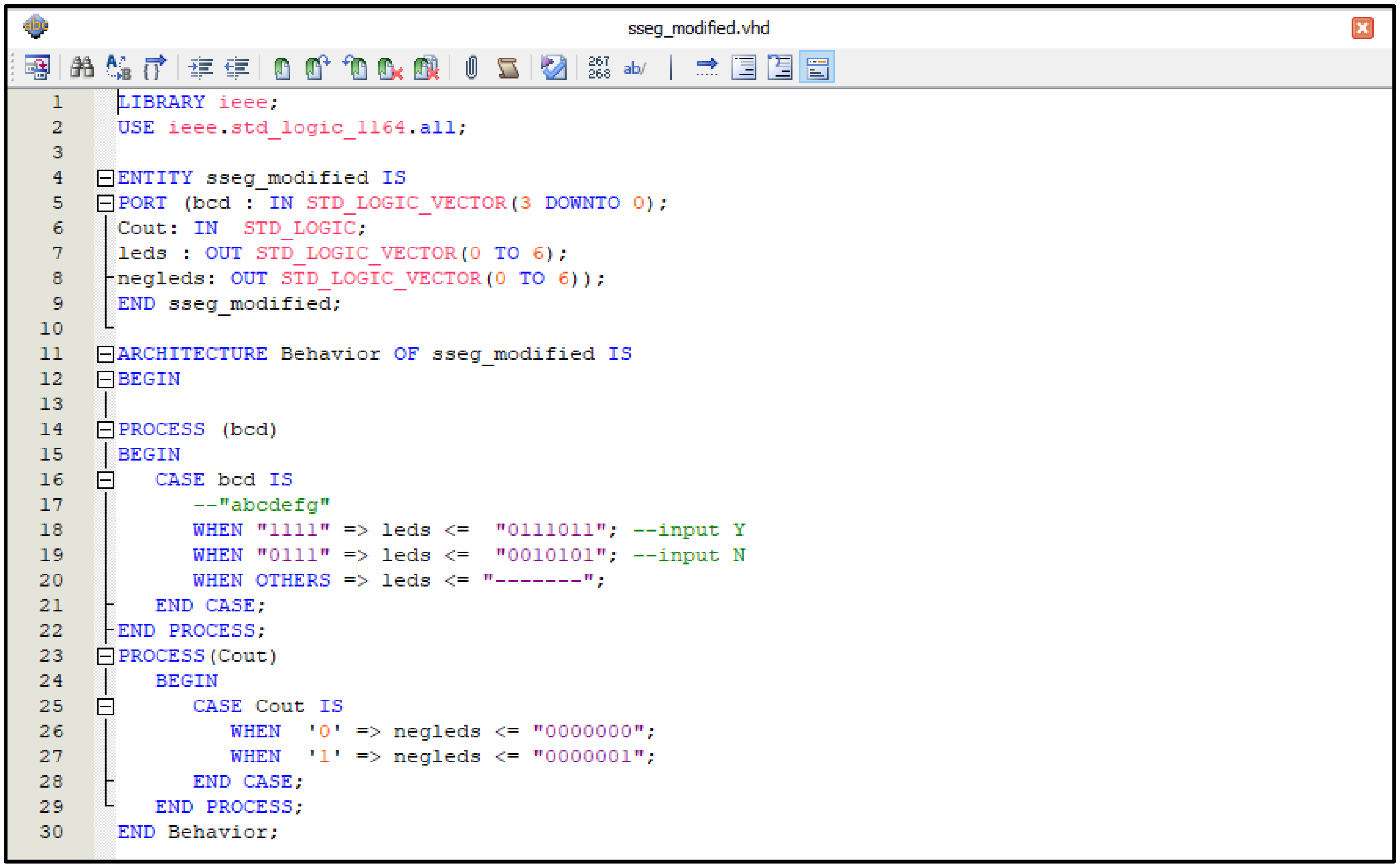
<u>VHDL code for ALU_3</u>



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
entity ALU3 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
Result : out unsigned (3 downto 0));
end ALU3;
architecture calculation of ALU3 is
signal p : std_logic;
begin
Result(0) <= '1';
Result(1) <= '1';
Result(2) <= '1';
process(Clock, OP, p)
begin
if(rising_edge(Clock)) THEN
case OP is
WHEN "0000000000000001" =>
p <= student_id(0) xnor student_id(1) xnor student_id(2) xnor student_id(3);
WHEN "0000000000000010" =>
p <= student_id(0) xnor student_id(1) xnor student_id(2) xnor student_id(3);
WHEN "0000000000000100"  =>
p <= student_id(0) xnor student_id(1) xnor student_id(2) xnor student_id(3);
WHEN "0000000000001000" =>
p <= student_id(0) xnor student_id(1) xnor student_id(2) xnor student_id(3);
WHEN "0000000000010000"  =>
p <= student_id(0) xnor student_id(1) xnor student_id(2) xnor student_id(3);
WHEN "0000000000100000" =>
p <= student_id(0) xnor student_id(1) xnor student_id(2) xnor student_id(3);
WHEN "0000000001000000" =>
p <= student_id(0) xnor student_id(1) xnor student_id(2) xnor student_id(3);
WHEN "0000000010000000" =>
p <= student_id(0) xnor student_id(1) xnor student_id(2) xnor student_id(3);
WHEN "0000000100000000" =>
p <= student_id(0) xnor student_id(1) xnor student_id(2) xnor student_id(3);
WHEN OTHERS =>
```



```vhdl
--Don't care, do nothing
Result <= null;
end case;
end if;
Result(3) <= p;
end process;
end calculation;
```

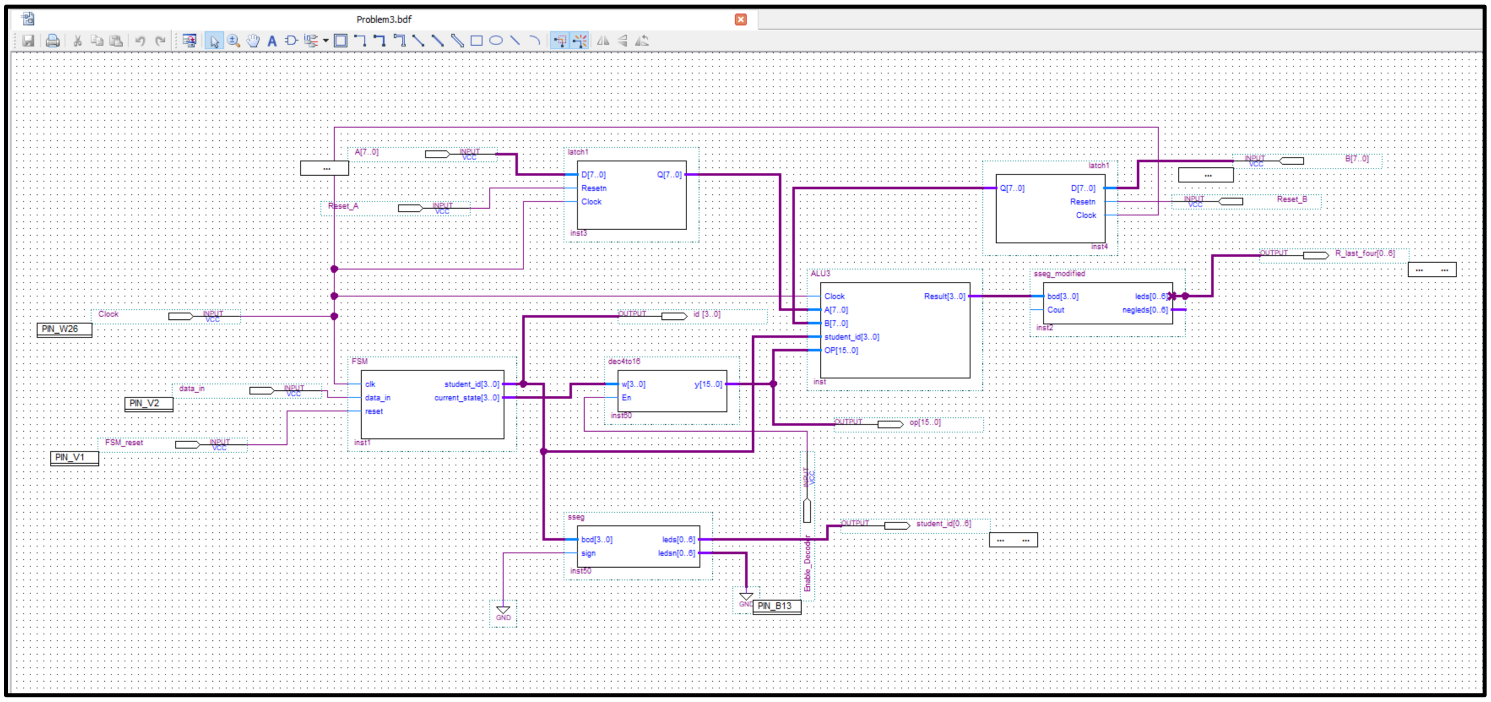


```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY sseg_modified 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 sseg_modified;

ARCHITECTURE Behavior OF sseg_modified IS
BEGIN

PROCESS (bcd)
BEGIN
   CASE bcd IS
      --"abcdefg"
      WHEN "1111" => leds <=  "0111011"; --input Y
      WHEN "0111" => leds <=  "0010101"; --input N
      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 Behavior;
```

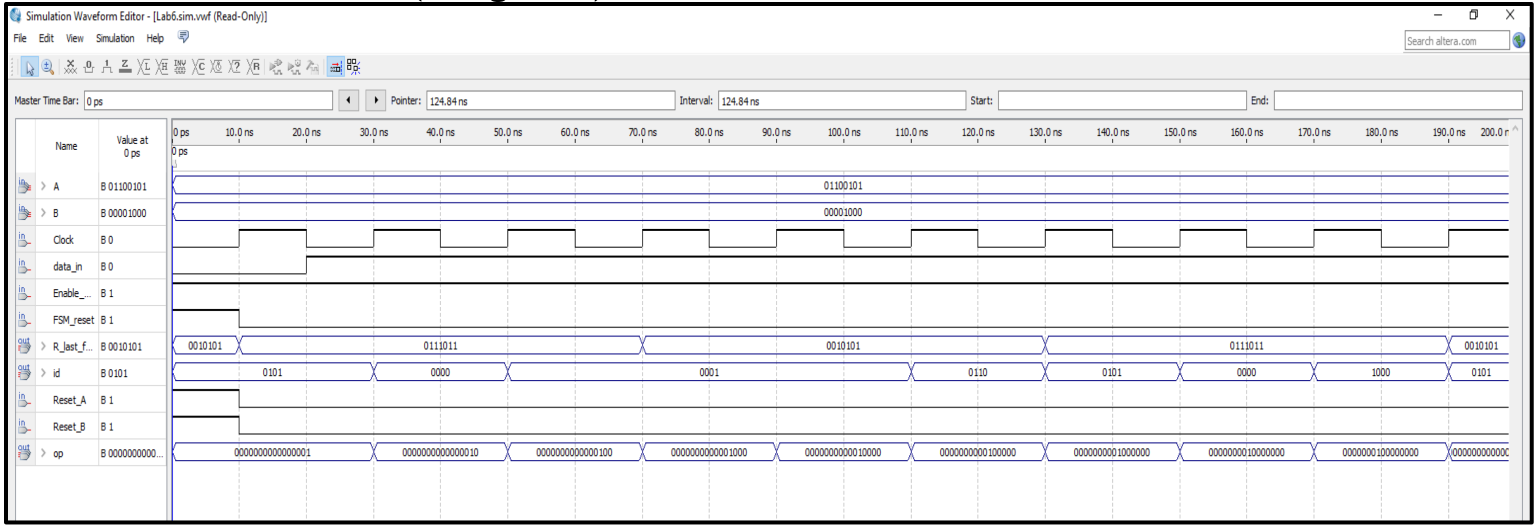
VHDL code for modified 7-segment decoder pictured above

Note: Since it differs from the one created and discussed on in previous labs, the seven-segment decoder code was shown due to the modifications made to it in accordance with the task assigned for problem 3.

Block Diagram for Problem 3 (with alu_3 block)



Waveform for Problem 3 (using FSM)



Simulation Waveform Editor - [Lab6.sim.vwf (Read-Only)]

File Edit View Simulation Help

Master Time Bar: 0 ps Pointer: 124.84 ns Interval: 124.84 ns Start: End:

```
Name        Value at 0 ps
A           B 01100101        01100101
B           B 00001000        00001000
Clock       B 0
data_in     B 0
Enable_...  B 1
FSM_reset   B 1
R_last_f... B 0010101         0010101 | 0111011 | 0010101 | 0111011 | 0010101
id          B 0101            0101 | 0000 | 0001 | 0110 | 0101 | 0000 | 1000 | 0101
Reset_A     B 1
Reset_B     B 1
op          B 0000000000...   0000000000000001 | 0000000000000010 | 0000000000000100 | 0000000000001000 | 0000000000010000 | 0000000000100000 | 0000000001000000 | 0000000010000000 | 0000000100000000 | 000000000000
```

Conclusion

In summary, this formal report on laboratory 6 summarizes the processor's efficiency over a range of problem sets. Created in Quartus with VHDL code and block diagrams, the basic general-purpose processor demonstrates its adaptability and effectiveness in a range of tasks.