



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

<b>Course Title:</b>	Digital Systems
<b>Course Number:</b>	COE 328
<b>Semester/Year (e.g.F2016)</b>	F2021

<b>Instructor:</b>	Dr.Patrick Siddavattam
--------------------	------------------------

<i>Assignment/Lab Number:</i>	LAB REPORT #6
<i>Assignment/Lab Title:</i>	Design of a General-Purpose Processor Unit (GPU)

<i>Submission Date</i> :	12/08/2021
<i>Due Date:</i>	12/09/2021

<b>Student LAST Name</b>	<b>Student FIRST Name</b>	<b>Student Number</b>	<b>Section</b>	<b>Signature*</b>
Patel	Astha	501040209	14	A.P.

## TABLE OF CONTENTS

## Page #

1. A brief introduction of components involved in a GPU design.....	3
2. Components	
2.1 Latch #1.....	3
2.1.1 Block symbol A.....	3
2.1.2 Waveform A.....	3
2.1.3 VHDL code A.....	4
2.2 Latch #2.....	4
2.2.1 Block symbol B.....	4
2.2.2 Waveform B.....	4
2.2.3 VHDL code B.....	5
2.3 Finite-State Machine (FSM).....	5
2.3.1 Block symbol C.....	5
2.3.2 VHDL code C.....	6
2.3.3 Waveform C.....	9
2.4 4-to-16 decoder.....	9
2.4.1 Block symbol D.....	9
2.4.2 Waveform D.....	9
2.4.3 VHDL code D.....	10
3. ALU designs	
3.1 ALU_1.....	10
3.1.1 Problem 1.....	10
3.1.2 Block symbol 1.1.....	10
3.1.3 Waveform 1.1.....	12
3.1.4 VHDL code ALU 1.....	12
3.1.5 VHDL code Seven-segment display 2.....	14
3.1.6 Block diagram 1.....	15
3.1.7 Waveform 1.2.....	15
3.2 ALU_2.....	15
3.2.1 Problem 2.....	16
3.2.2 Block symbol 2.....	16
3.2.3 Waveform 2.1.....	17
3.2.4 VHDL code ALU 2.1.....	17
3.2.5 VHDL code Seven-Segment display 2.2.....	18
3.2.6 Block diagram 2.1.....	19
3.2.7 Waveform 2.2.....	20
3.3 ALU_3.....	20
3.3.1 Problem 3.....	21
3.3.2 Block symbol 3.....	21
3.3.3 Waveform 3.1.....	22
3.3.4 VHDL code ALU 3.1.....	23
3.3.5 VHDL code Seven-Segment display 3.2.....	24
3.3.6 Block diagram 3.1.....	24
3.3.7 Waveform 3.2.....	24
4. Conclusion.....	25

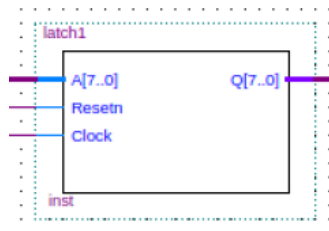
## **A BRIEF DESCRIPTION OF THE COMPONENTS**

The objective of this lab experiment is to design a simple General-purpose Processing Unit (GPU). Multiple components will be utilized to build the GPU system, these components include Latches; a Finite-State Machine (FSM); a 4-to-16 Decoder; Arithmetic & Logic Unit (ALU); Seven-Segment Display (SSEG). A latch is a level sensitive storage element that temporarily stores the input. Next, an FSM works as a counter that could be useful when a certain order of outputs is required. Furthermore, a 4-to-16 decoder utilizes multiple inputs to execute a particular operation. An ALU machine combines information from different components to execute the desired output. Finally, the seven-segment displays a binary or a hexadecimal value from the final output. For this lab experiment will utilize all the components described above to design a final GPU. Within a computer system, a GPU executes mathematical operations for the Central Processing Unit (CPU). The goal of this lab is to experiment with different mathematical operations using the GPU design.

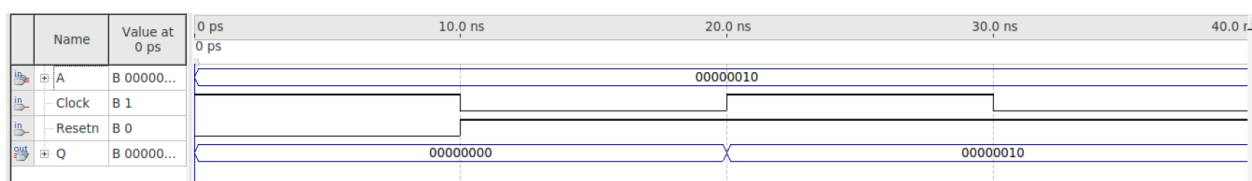
## **COMPONENTS:**

### **2.1 Latch #1:**

Latches are level sensitive storage elements; they execute operations when the machine is clocked. A latch has pins Clock; Reset; Input; output. Latch machines are level sensitive; they change or stay in the same state depending on the clock signal. The Clock pin allows the latch to discard the present data and store the new incoming data. The Reset pin resets the values to a user-customized value; the latch stays at the reset state until clocked again. The latch will store an 8-bit binary number and send its information to the ALU. The first 8-bit binary representation is the 6<sup>th</sup> and 7<sup>th</sup> digit of my student ID (0000 and 0010, respectively).



2.2.1: Block symbol A : A symbol for latch #1 machine.



### 2.1.2: Waveform A: the waveform of a latch machine.

```

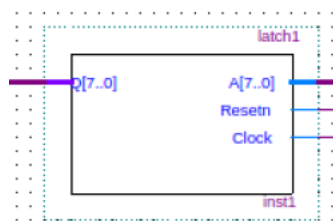
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY latch1 IS
5  PORT (A          : IN STD_LOGIC_VECTOR(7 DOWNTO 0); --8 bit input
6        Resetn, Clock : IN STD_LOGIC; --1 bit clock input and 1 bit reset input bit
7        Q           : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)); --8 bit output
8  END latch1;
9
10 ARCHITECTURE Behaviour OF latch1 IS
11 BEGIN
12   PROCESS (Resetn,Clock)--Process takes reset and clock as inputs
13   BEGIN
14     IF Resetn = '0' THEN --when reset inout is '0' the latches does not operate
15       Q <= "00000000";
16     ELSIF Clock'EVENT AND Clock = '1' THEN --level sensitive based on clock
17       Q <= A;
18     END IF;
19   END PROCESS;
20 END Behaviour;

```

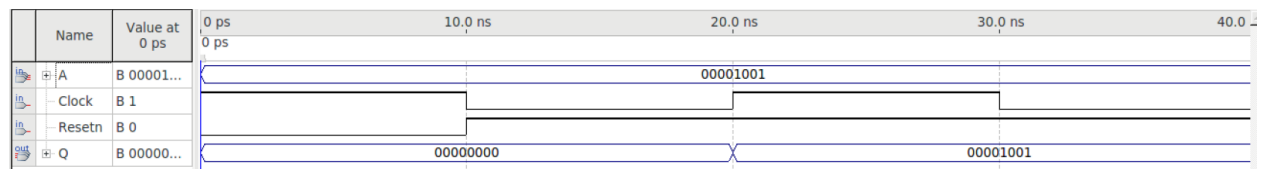
### 2.1.3: VHDL code A : A VHDL code for a Latch machine.

## 2.2 Latch #2:

This latch machine is identical to the previous machine, this time, this machine will store the 8<sup>th</sup> and the 9<sup>th</sup> digit (0000 and 1001, respectively) of my student ID number. As described above, latches are level sensitive storage elements; they execute operations when the machine is clocked. A latch has pins Clock; Reset; Input; output. The Clock pin allows the latch to discard the present data and store the new incoming data. The Reset pin resets the values to a user-customized value; the latch stays at the reset state until clocked again. The latch will store an 8-bit binary number and send its information to the ALU.



### 2.2.1: Block symbol B: A block diagram of Latch #2.



### 2.2.2: Waveform B: A waveform representation of the second latch symbol.

```

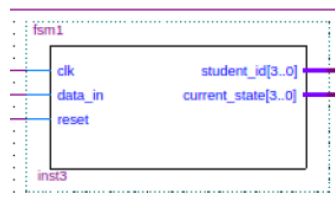
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY latch1 IS
5  PORT (A          : IN STD_LOGIC_VECTOR(7 DOWNTO 0); --8 bit input
6        Resetn, Clock : IN STD_LOGIC; --1 bit clock input and 1 bit reset input bit
7        Q           : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));--8 bit output
8  END latch1;
9
10 ARCHITECTURE Behaviour OF latch1 IS
11 BEGIN
12   PROCESS (Resetn,Clock)--Process takes reset and clock as inputs
13   BEGIN
14     IF Resetn = '0' THEN --when reset inout is '0' the latches does not operate
15       Q <= "00000000";
16     ELSIF Clock'EVENT AND Clock = '1' THEN --level sensitive based on clock
17       Q <= A;
18     END IF;
19   END PROCESS;
20 END Behaviour;

```

2.2.3: VHDL code B: A VHDL code for a Latch machine.

### 2.3 Finite-State Machine (FSM):

An FSM is a machine that could represent a particular state at any given time, from a finite set of possible states. This machine will represent one state at a time, each time it is clocked. A Mealy FSM is utilized for this experiment. A Mealy machine is dependent on the primary inputs, the present state, and the primary outputs. Depending on the primary inputs, the machine will represent a state when clocked. The Mealy machine in this lab experiment represents the student ID as the states change from 0 to 8.



2.3.1: Block symbol C: A block symbol of the FSM machine.

```

1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.all;
3
4  ENTITY fsm1 IS
5  PORT
6  (
7      clk          : IN STD_LOGIC;
8      data_in      : IN STD_LOGIC;
9      reset        : IN STD_LOGIC;
10     student_id   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
11     current_state : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
12 );
13 END fsm1;
14 ARCHITECTURE fsm OF fsm1 IS
15 --Build an enumerated type with 9 states for the state machine (9 states
16 --for parsing 9 digits of student ID)
17 TYPE state_type IS (s0,s1,s2,s3,s4,s5,s6,s7,s8);
18 --register to hold the current state
19 SIGNAL yfsm : state_type;
20
21 BEGIN
22     PROCESS (clk,reset)
23     BEGIN
24         IF reset = '1' THEN
25             yfsm <= s0;

```

```

26         ELSE IF (clk 'EVENT AND clk = '1') THEN
27             --Determine the next state synchronously, based on
28             --the current state and input.
29             CASE yfsm IS
30
31                 WHEN s0 =>
32                     IF data_in = '0' THEN
33                         current_state <= "0000";
34                         yfsm <= s0;
35                     ELSE
36                         current_state <= "0001";
37                         yfsm <= s1;
38                     END IF;
39                 WHEN s1 =>
40                     IF data_in = '0' THEN
41                         current_state <= "0001";
42                         yfsm <= s1;
43                     ELSE
44                         current_state <= "0010";
45                         yfsm <= s2;
46                     END IF;
47                 WHEN s2 =>
48                     IF data_in = '0' THEN
49                         current_state <= "0010";
50                         yfsm <= s2;
51                     ELSE
52                         current_state <= "0011";

```

```

56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82

```

```

    yfsm <= s3;
  END IF;
  WHEN s3 =>
    IF data_in = '0' THEN
      current_state <= "0011";
      yfsm <= s3;
    ELSE
      current_state <= "0100";
      yfsm <= s4;
    END IF;
  WHEN s4 =>
    IF data_in = '0' THEN
      current_state <= "0100";
      yfsm <= s4;
    ELSE
      current_state <= "0101";
      yfsm <= s5;
    END IF;
  WHEN s5 =>
    IF data_in = '0' THEN
      current_state <= "0101";
      yfsm <= s5;
    ELSE
      current_state <= "0110";
      yfsm <= s6;
    END IF;
  WHEN s6 =>

```

```

83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109

```

```

    IF data_in = '0' THEN
      current_state <= "0110";
      yfsm <= s6;
    ELSE
      current_state <= "0111";
      yfsm <= s7;
    END IF;
  WHEN s7 =>
    IF data_in = '0' THEN
      current_state <= "0111";
      yfsm <= s7;
    ELSE
      current_state <= "1000";
      yfsm <= s8;
    END IF;
  WHEN s8 =>
    IF data_in = '0' THEN
      current_state <= "1000";
      yfsm <= s1;
    ELSE
      current_state <= "0000";
      yfsm <= s0;
    END IF;
  END CASE;
END IF;
END IF;

```

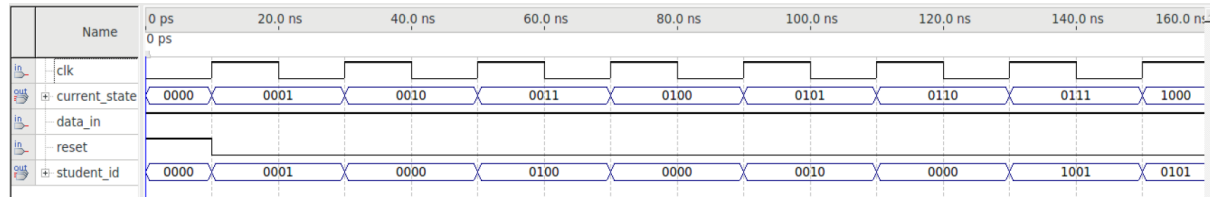
```

110      END PROCESS;
111
112      --implement the moore or mealy logic here
113
114      PROCESS (yfsm,data_in)--IF REQUIRED
115      BEGIN
116
117          CASE yfsm IS
118
119              WHEN s0 =>
120                  IF data_in = '0' THEN
121                      student_id <= "0101";
122                  ELSE
123                      student_id <= "0000";
124                  END IF;
125              WHEN s1 =>
126                  IF data_in = '0' THEN
127                      student_id <= "0000";
128                  ELSE
129                      student_id <= "0001";
130                  END IF;
131              WHEN s2 =>
132                  IF data_in = '0' THEN
133                      student_id <= "0001";
134                  ELSE
135                      student_id <= "0000";
136                  END IF;
137
138              WHEN s3 =>
139                  IF data_in = '0' THEN
140                      student_id <= "0000";
141                  ELSE
142                      student_id <= "0100";
143                  END IF;
144              WHEN s4 =>
145                  IF data_in = '0' THEN
146                      student_id <= "0100";
147                  ELSE
148                      student_id <= "0000";
149                  END IF;
150              WHEN s5 =>
151                  IF data_in = '0' THEN
152                      student_id <= "0000";
153                  ELSE
154                      student_id <= "0010";
155                  END IF;
156              WHEN s6 =>
157                  IF data_in = '0' THEN
158                      student_id <= "0010";
159                  ELSE
160                      student_id <= "0000";
161                  END IF;
162              WHEN s7 =>
163                  IF data_in = '0' THEN
164                      student_id <= "0000";
165                  ELSE
166                      student_id <= "1001";
167                  END IF;
168              WHEN s8 =>
169                  IF data_in = '0' THEN
170                      student_id <= "1001";
171                  ELSE
172                      student_id <= "0101";
173                  END IF;
174          END CASE;
175      END PROCESS;
176  END fsm;

```

2.3.2: VHDL code C: The VHDL code for the required FSM machine that is designed using the Mealy FSM. The states change from 0 to 8.

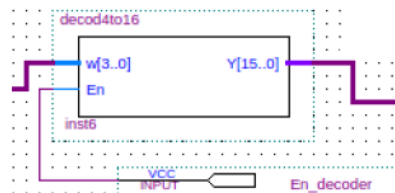




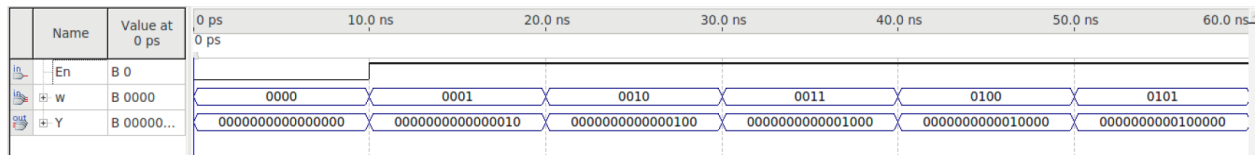
2.3.3: Waveform C : A waveform of the Mealy FSM. An example of the above mealy FSM design. The data\_in = 1, the mealy machine refers to the next state, which is s1 and the student digit is s1 = 0 when data\_in = 1.

#### 2.4 4-to-16 Decoder:

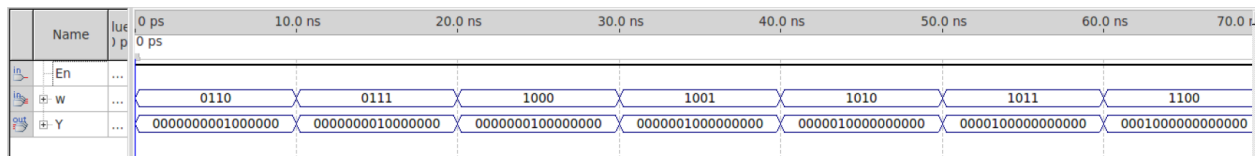
A 4-to-16 decoder receives a 4-bit binary number that is utilized to output one piece of information, a 16-bit binary number for this case. There are sixteen possible outputs that are executed one at a time depending on the 4-bit input. For this experiment, these inputs represent an output which is then reported to the ALU that executes the operations. The 4-to-16 decoder will receive a state that the FSM is representing at a particular time, the decoder will then output a 16-bit binary number that is then sent to the ALU to manage. An Enable pin enables the decoder to be turned on and off. The decoder will only output information if the Enable is on.



2.4.1: Block symbol D: A block symbol of a 4-to-16 decoder.



2.4.2: Waveform D part 1: A waveform of a 4-to-16 decoder for the first 6 macrocodes.



2.4.2: Waveform D part 2: A waveform of a 4-to-16 decoder for the next 7 macrocodes.

	Name	Value at 0 ps	0 ps	10.0 ns	20.0 ns	30.0 ns	40.0 ns
in	En	B 1					
in	W	B 1100	1100	1101	1110	1111	
out	Y	B 00010...	0001000000000000	0010000000000000	0100000000000000	1000000000000000	

2.4.2: Waveform D part 3: A waveform of a 4-to-15 decoder for the last 4 macrocodes.

```

1  LIBRARY ieee ;
2  USE ieee.std_logic_1164.all ;
3
4  ENTITY decod4to16 IS
5  PORT (w : IN  STD_LOGIC_VECTOR(3 DOWNTO 0) ;
6        En : IN  STD_LOGIC ;
7        Y : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)) ;
8
9  END decod4to16 ;
10
11 ARCHITECTURE Behavior OF decod4to16 IS
12     SIGNAL Enw : STD_LOGIC_VECTOR(4 DOWNTO 0) ;
13 BEGIN
14     Enw <= En & w ;
15     WITH Enw SELECT
16     Y <= "0000000000000001" WHEN "10000",
17         "0000000000000010" WHEN "10001",
18         "0000000000000100" WHEN "10010",
19         "0000000000001000" WHEN "10011",
20         "0000000000010000" WHEN "10100",
21         "0000000000100000" WHEN "10101",
22         "0000000001000000" WHEN "10110",
23         "0000000010000000" WHEN "11000",
24         "0000000100000000" WHEN "11001",
25         "0000001000000000" WHEN "11010",
26         "0000010000000000" WHEN "11011",
27         "0000100000000000" WHEN "11100",
28         "0001000000000000" WHEN "11101",
29
30         "0010000000000000" WHEN "11110",
31         "0100000000000000" WHEN "11111",
32         "0000000000000000" WHEN OTHERS;
33
34
35 END Behavior ;

```

2.4.3: VHDL code D : a VHDL code for a 4-to-16 decoder.

## ALU 1

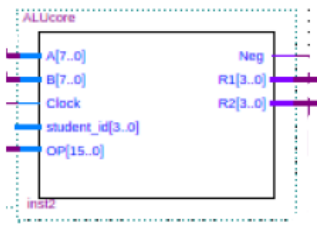
An ALU is a processing unit that handles mathematical calculations. For this lab experiment, the ALU component receives two types of information, first, two 8-bit numbers from the two latches, pin A[7..0] and pin B[7..0], and second, a 16-bit microcode from the 4-to-16 decoder. The decoder will receive a value from the FSM and send the 16-bit microcode command associated with the current state to the ALU. Inside the ALU code prompt, there are operations defined under each microcode; the operation is executed each time a microcode is received. Furthermore, once the ALU receives a microcode, the operation is applied to the two 8-bit numbers from the latches. The result, also an 8-bit number, stores the number inside two arrays, R1[3..0] and R2[3..0]; R1 stores the lower 4-

bits and R2 stores the higher 4-bits of the result. Regarding the pins on the block diagram: the input pins A[7..0] and B[7..0] are the two 8-bit numbers received from the two latches, pin OP[15..0] is the 16-bit number sent from the decoder, and the Clock pin allows ALU to move onto the next cycle of the microcode, and the output pins, Neg will be utilized to display a negative sign if the result is a negative number, and finally, pins R1[3..0] and R2[3..0] will store the 8-bit result executed from the ALU operation. This result is then sent to the seven-segment display that will display the binary or hexadecimal number on the display, observed on the waveform.

Function #	Microcode	Boolean Operation / Function
1	0000000000000001	$\text{sum}(A, B)$
2	0000000000000010	$\text{diff}(A, B)$
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	$\overline{A \oplus B}$

Table 1. ALU Core Operations for Problem 1

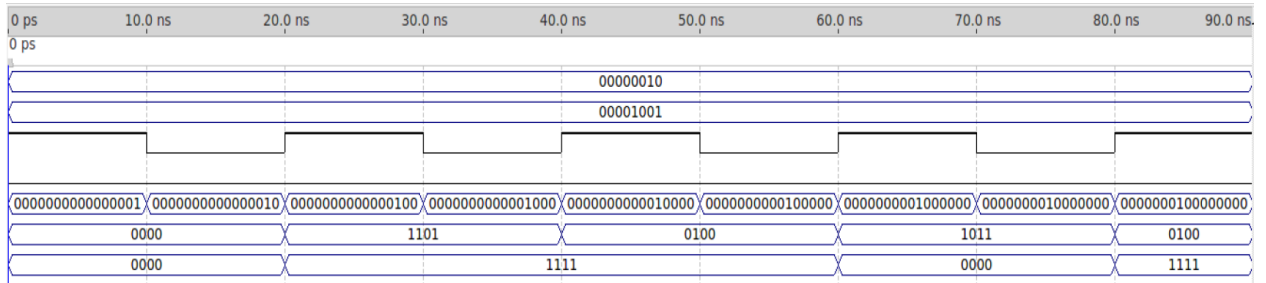
### 3.1.1: Problem 1



3.1.2: Block symbol 1.1 : a block symbol of the ALU component with operations integrated concerning Table 1.

	Name
in	A
in	B
in	Clock
out	Neg
in	OP
out	R1
out	R2

Figure 1: a table of the pins utilized on the waveform in the next figure.



3.1.3: Waveform 1.1: A waveform of the ALU component only.

```

1  LIBRARY ieee;
2  USE ieee.STD_LOGIC_1164.all;
3  USE ieee.STD_LOGIC_UNSIGNED.all;
4  USE ieee.NUMERIC_STD.all;
5
6  ENTITY Problem2d IS
7  PORT (A,B      : IN UNSIGNED(7 DOWNTO 0); -- 8 bit inputs from latches A and B
8        Clock    : IN STD_LOGIC; -- input clock signal
9        student_id : IN UNSIGNED(3 DOWNTO 0); --4 bit student ID from FSM
10       OP       : IN UNSIGNED(15 DOWNTO 0); --16-bit selector for Operation from Decoder
11       Neg      : OUT STD_LOGIC; -- is the result negative? set -ve output
12       R1       : OUT UNSIGNED(3 DOWNTO 0); --lower 4-bits of 8-bit Result Outout
13       R2       : OUT UNSIGNED(3 DOWNTO 0); --higher 4-bits of 8-bits Result Output
14
15  END Problem2d;
16
17  ARCHITECTURE calculation OF Problem2d IS -- temporary signal declarations
18
19  SIGNAL Reg1,Reg2,Result : UNSIGNED(7 DOWNTO 0) := (OTHERS => '0');
20  SIGNAL Reg4 :UNSIGNED(0 TO 7);
21
22  BEGIN
23
24  Reg1 <= A; --temporary store A in Reg1 local variable
25  Reg2 <= B; --temporary store B in Reg2 local variable
26
27  PROCESS (Clock,OP)
28  BEGIN

```

```

29 IF(rising_edge(Clock)) THEN --Do calculation @ positive edge of clock cycle
30 CASE OP IS
31
32     WHEN "0000000000000001" =>
33         --Shift A to right by two bits, input bit =1 (SHR)
34         Result <= Reg1 srl 2;
35
36     WHEN "0000000000000010" =>
37         --Produce the difference of A and B and then increment by 4
38         Result <= (Reg1 - Reg2) + 4;
39         IF (Result < 0) THEN
40             Neg <= '1';
41         ELSE Neg <= '0';
42         END IF;
43
44     WHEN "0000000000000100" =>
45         --Find the greater value of A and B and produce the results (Max(A,B))
46         IF(Reg1 > Reg2) THEN
47             Result <= Reg1;
48         ELSE Result <= Reg2;
49         END IF;
50
51     WHEN "0000000000001000" =>
52         --Swap the upper 4 bits of A by the lower 4 bits of B
53         Result(7 DOWNTO 4) <= Reg2(3 DOWNTO 0);
54         Result(3 DOWNTO 0) <= Reg1(3 DOWNTO 0);
55

```

```

56         --Do boolean NOR
57         Result <= Reg1 NOR Reg2;
58
59     WHEN "0000000000100000" =>
60         --Do boolean AND
61         Result <= Reg1 AND Reg2;
62
63     WHEN "0000000001000000" =>
64         --Do boolean XOR
65         Result <= Reg1 XOR Reg2;
66
67     WHEN "0000000010000000" =>
68         --Do boolean OR
69         Result <= Reg1 OR Reg2;
70
71     WHEN "0000000100000000" =>
72         --Do boolean XNOR
73         Result <= Reg1 XNOR Reg2;
74
75     WHEN OTHERS =>
76         --Don't care ,do nothing
77
78     END CASE;
79     END IF;
80     END PROCESS;
81
82     R1 <= Result(3 DOWNTO 0); --Since the output seven segment can
83
84     R2 <= Result(7 DOWNTO 4); -- Only 4-bits, split the 8-bit two 4-bits
85     END calculation;

```

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY sseg1 IS
5  PORT ( bcd :IN STD_LOGIC_VECTOR(3 DOWNTO 0);
6        leds :OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
7        neg  : IN STD_LOGIC;
8        neg_leds : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
9        );
10
11  end sseg1;
12
13  ARCHITECTURE Behavior OF sseg1 IS
14  BEGIN
15  PROCESS (bcd,neg)
16  BEGIN
17  IF (neg = '0') THEN
18      neg_leds <= "0000000";
19  ELSE
20      CASE bcd IS
21          --abcdefg
22          WHEN "0000" => leds <= "1111110";--0
23          WHEN "0001" => leds <= "0110000";--1
24          WHEN "0010" => leds <= "1101101";--2
25          WHEN "0011" => leds <= "1111001";--3
26          WHEN "0100" => leds <= "0110011";--4
27          WHEN "0101" => leds <= "1011011";--5
28          WHEN "0110" => leds <= "1011111";--6
29          WHEN "0111" => leds <= "1110000";--7
30          WHEN "1000" => leds <= "1111111";--8

```

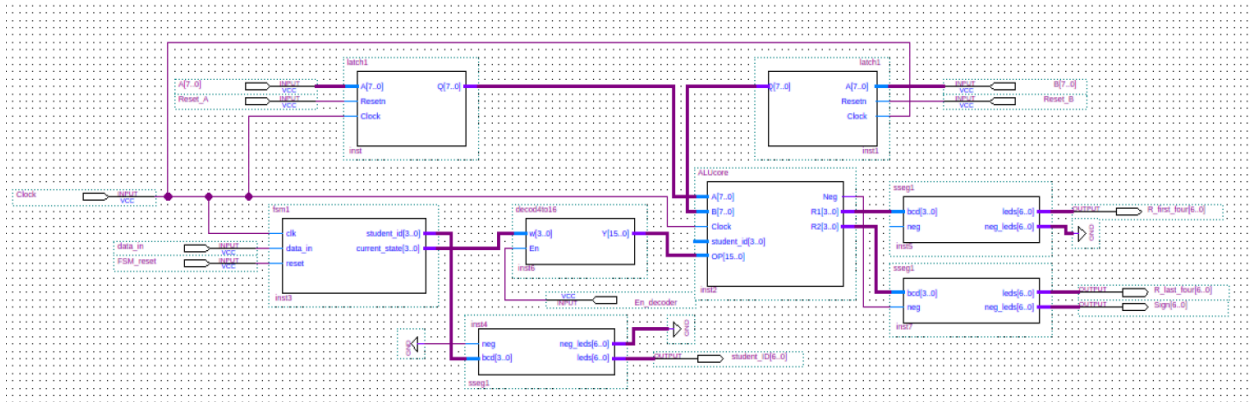
2.4.4: VHDL code 1.1: A VHDL code for the ALU design.

```

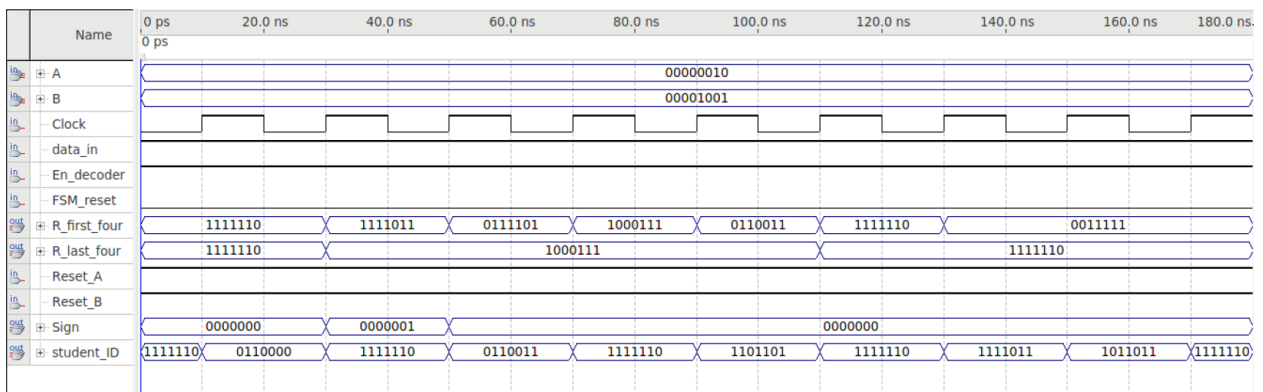
29      WHEN "1001" => leds <= "1111011";--9
30      WHEN "1010" => leds <= "1110111";--A
31      WHEN "1011" => leds <= "0011111";--B
32      WHEN "1100" => leds <= "1001110";--C
33      WHEN "1101" => leds <= "0111101";--D
34      WHEN "1110" => leds <= "1001111";--E
35      WHEN "1111" => leds <= "1000111";--F
36      WHEN OTHERS => leds <= "-----";
37  END CASE;
38
39  ELSE
40      neg_leds <= "0000001";
41  CASE bcd IS
42      WHEN "0000" => leds <= "1111110";--0
43      WHEN "0001" => leds <= "0110000";--1
44      WHEN "0010" => leds <= "1101101";--2
45      WHEN "0011" => leds <= "1111001";--3
46      WHEN "0100" => leds <= "0110011";--4
47      WHEN "0101" => leds <= "1001011";--5
48      WHEN "0110" => leds <= "1011111";--6
49      WHEN "0111" => leds <= "1110000";--7
50      WHEN "1000" => leds <= "1111111";--8
51      WHEN "1001" => leds <= "1111011";--9
52      WHEN "1010" => leds <= "1110111";--A
53      WHEN "1011" => leds <= "0011111";--B
54      WHEN "1100" => leds <= "1001110";--C
55      WHEN "1101" => leds <= "0111101";--D
56
57      WHEN "1110" => leds <= "1001111";--E
58      WHEN "1111" => leds <= "1000111";--F
59      WHEN OTHERS => leds <= "-----";
60  END CASE;
61  END IF;
62  END PROCESS;
63  END Behavior;

```

3.1.5: VHDL code 1.2: A VHDL code for the seven-segment display.



3.1.6: Block Diagram 1: The final block diagram of the GPU design.



3.1.7: Waveform 1.2: A waveform of the final GPU design.

## ALU 2

The second ALU design executes mathematic operations based on a 16-bit input as well. This ALU2 design will receive two 8-bit inputs from the latches as well and a 16-bit microcode from the 4-to-16 decoder. The decoder receives a current state from the FSM and the decoder associates that current state binary value with the 16-bit microcode inside the decoder code. The decoder will send this microcode to the ALU to handle the rest of the execution process. Once the ALU receives the microcode, it will run the mathematical operation linked to that microcode. The operation will be applied to the two 8-bit binary numbers that are temporarily stores in the latches. The result will be an 8-bit number that will be split into two arrays. The pins containing the ALU block symbol are: the input pins A[7..0] and B[7..0] are the two 8-bit numbers obtained from the latches, pin OP[15..0] will send the microcode from the decoder, and the Clock symbol will allow the ALU to move onto the next cycle, and the output pins, R1[3..0] and R2[3..0] will contain the result that is split into two arrays. R1[3..0] contains the lower 4-bits of the result and R2[3..0] contain the

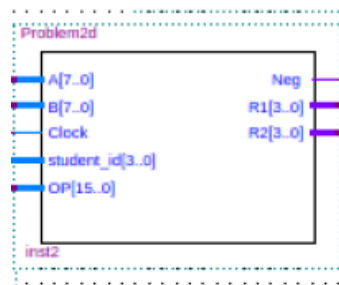


higher 4-bits. This result will be displayed on the seven-segment display as either a binary or a hexadecimal representation of the student ID number, observed on the waveform.

d)

Function #	Operation / Function
1	Shift <b>A</b> to right by two bits, input bit = 1 (SHR)
2	Produce the difference of <b>A</b> and <b>B</b> and then increment by 4
3	Find the greater value of <b>A</b> and <b>B</b> and produce the results ( $\text{Max}(\mathbf{A}, \mathbf{B})$ )
4	Swap the upper 4 bits of <b>A</b> by the lower 4 bits of <b>B</b>
5	Increment <b>A</b> by 1
6	Produce the result of ANDing <b>A</b> and <b>B</b>
7	Invert the upper four bits of <b>A</b>
8	Rotate <b>B</b> to left by 3 bits (ROL)
9	Show null on the output

3.2.1: Table 2: a table of operations integrated for this ALU design.

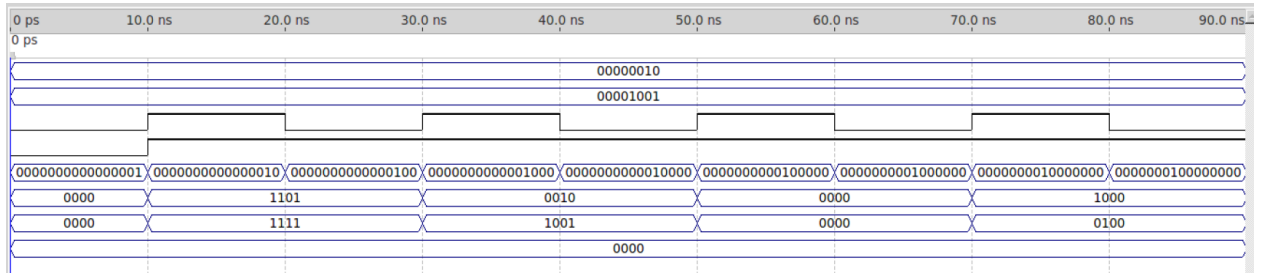


3.2.2: Block symbol 2.1: A block symbol of the ALU component for problem 2.

	Name
in	A
in	B
in	Clock
out	Neg
in	OP
out	R1
out	R2
in	stude...

Figure 2.1: A table of the pins utilized on the waveform of ALU.





3.2.3: Waveform 2.1: A waveform of the ALU design concerning Table 2.

```

1  LIBRARY ieee;
2  USE ieee.STD_LOGIC_1164.all;
3  USE ieee.STD_LOGIC_UNSIGNED.all;
4  USE ieee.NUMERIC_STD.all;
5
6  ENTITY Problem2d IS
7  PORT (A,B          : IN UNSIGNED(7 DOWNTO 0); -- 8 bit inputs from latches A and B
8        Clock        : IN STD_LOGIC; -- input clock signal
9        student_id    : IN UNSIGNED(3 DOWNTO 0); --4 bit student ID from FSM
10       OP            : IN UNSIGNED(15 DOWNTO 0); --16-bit selector for Operation from Decoder
11       Neg           : OUT STD_LOGIC; -- is the result negative? set -ve output
12       R1            : OUT UNSIGNED(3 DOWNTO 0); --lower 4-bits of 8-bit Result Outout
13       R2            : OUT UNSIGNED(3 DOWNTO 0); --higher 4-bits of 8-bits Result Output
14
15  END Problem2d;
16
17  ARCHITECTURE calculation OF Problem2d IS -- temporary signal declarations
18
19  SIGNAL Reg1,Reg2,Result : UNSIGNED(7 DOWNTO 0) := (OTHERS => '0');
20  SIGNAL Reg4 : UNSIGNED(0 TO 7);
21
22  BEGIN
23
24  Reg1 <= A; --temporary store A in Reg1 local variable
25  Reg2 <= B; --temporary store B in Reg2 local variable
26
27  PROCESS (Clock,OP)
28  BEGIN
29
30      IF(rising_edge(Clock)) THEN --Do calculation @ positive edge of clock cycle
31          CASE OP IS
32
33              WHEN "0000000000000001" =>
34                  --Shift A to right by two bits, input bit =1 (SHR)
35                  Result <= Reg1 srl 2;
36
37              WHEN "0000000000000010" =>
38                  --Produce the difference of A and B and then increment by 4
39                  IF(Reg1 < Reg2) THEN
40                      Result <= (Reg1 - Reg2) + 4;
41                      Neg <= '1';
42                  ELSE
43                      Result <= (Reg1 + Reg2) + 4;
44                      Neg <= '0';
45                  END IF;
46
47              WHEN "0000000000000100" =>
48                  --Find the greater value of A and B and produce the results (Max(A,B))
49                  IF Reg1 > Reg2 THEN
50                      Result <= Reg1;
51                  ELSE Result <= Reg2;
52                  END IF;
53                  Neg <= '0';
54
55              WHEN "0000000000001000" =>
56                  --Swap the upper 4 bits of A by the lower 4 bits of B

```

```

56         WHEN "000000000010000" =>
57             --Increment A by 1
58             Result <= Reg1 + 1;
59
60         WHEN "000000000100000" =>
61             --Produce the results of ANDing A and B
62             Result <= Reg1 AND Reg2;
63
64         WHEN "000000001000000" =>
65             --Invert the upper four bits of A
66             Result(0) <= Reg1(0);
67             Result(1) <= Reg1(1);
68             Result(2) <= Reg1(2);
69             Result(3) <= Reg1(3);
70             Result(4) <= NOT Reg1(4);
71             Result(5) <= NOT Reg2(5);
72             Result(6) <= NOT Reg2(6);
73             Result(7) <= NOT Reg2(7);
74
75         WHEN "000000001000000" =>
76             --Rotate B to left by 3 bits (ROL)
77             Result <= Reg2 ROL 3;
78
79         WHEN "000000010000000" =>
80             --Show null on the output
81             Result <= NULL;
82
83         WHEN OTHERS =>
84             --Don't care ,do nothing
85
86     END CASE;
87 END IF;
88 END PROCESS;
89
90 R1 <= Result(3 DOWNTO 0); --Since the output seven segment can
91 R2 <= Result(7 DOWNTO 4); -- Only 4-bits, split the 8-bit two 4-bits
92
93 END calculation;

```

### 3.2.4: VHDL code 2.1: A VHDL code for the ALU design concerning problem 2.

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY sseg1 IS
5  PORT ( bcd :IN STD_LOGIC_VECTOR(3 DOWNTO 0);
6        leds :OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
7        neg  : IN STD_LOGIC;
8        neg_leds : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
9        );
10
11  end sseg1;
12
13  ARCHITECTURE Behavior OF sseg1 IS
14  BEGIN
15      PROCESS (bcd,neg)
16      BEGIN
17          IF (neg = '0') THEN
18              neg_leds <= "0000000";
19              CASE bcd IS
20                  WHEN "0000" => leds <= "1111110";--0
21                  WHEN "0001" => leds <= "0110000";--1
22                  WHEN "0010" => leds <= "1101101";--2
23                  WHEN "0011" => leds <= "1111001";--3
24                  WHEN "0100" => leds <= "0110011";--4
25                  WHEN "0101" => leds <= "1011011";--5
26                  WHEN "0110" => leds <= "1011111";--6
27                  WHEN "0111" => leds <= "1110000";--7
28                  WHEN "1000" => leds <= "1111111";--8

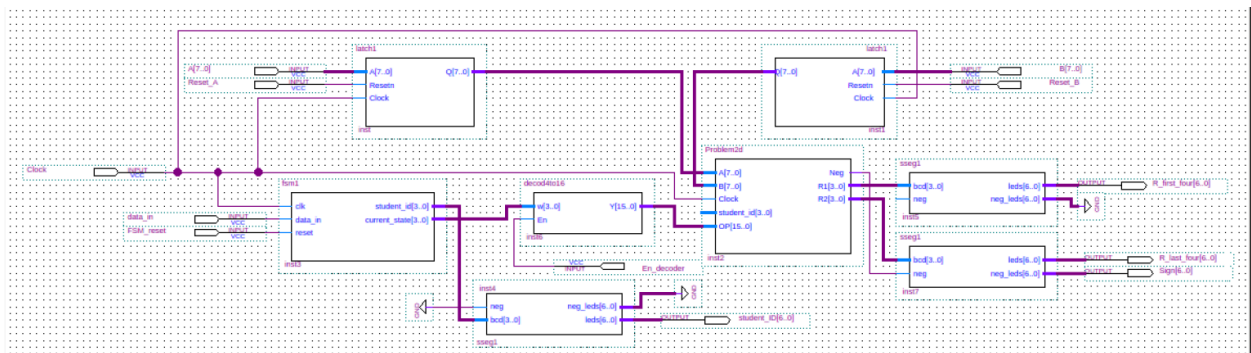
```

```













29         WHEN "1001" => leds <= "1111011";--9
30         WHEN "1010" => leds <= "1110111";--A
31         WHEN "1011" => leds <= "0011111";--B
32         WHEN "1100" => leds <= "1001110";--C
33         WHEN "1101" => leds <= "0111101";--D
34         WHEN "1110" => leds <= "1001111";--E
35         WHEN "1111" => leds <= "1000111";--F
36         WHEN OTHERS => leds <= "-----";
37     END CASE;
38
39 ELSE
40     neg_leds <= "0000001";
41     CASE bcd IS
42     WHEN "0000" => leds <= "1111110";--0
43     WHEN "0001" => leds <= "0110000";--1
44     WHEN "0010" => leds <= "1101101";--2
45     WHEN "0011" => leds <= "1111001";--3
46     WHEN "0100" => leds <= "0110011";--4
47     WHEN "0101" => leds <= "1001011";--5
48     WHEN "0110" => leds <= "1011111";--6
49     WHEN "0111" => leds <= "1110000";--7
50     WHEN "1000" => leds <= "1111111";--8
51     WHEN "1001" => leds <= "1111011";--9
52     WHEN "1010" => leds <= "1110111";--A
53     WHEN "1011" => leds <= "0011111";--B
54     WHEN "1100" => leds <= "1001110";--C
55     WHEN "1101" => leds <= "0111101";--D
56     WHEN "1110" => leds <= "1001111";--E
57     WHEN "1111" => leds <= "1000111";--F
58     WHEN OTHERS => leds <= "-----";
59     END CASE;
60 END IF;
61 END PROCESS;
62 END Behavior;

```

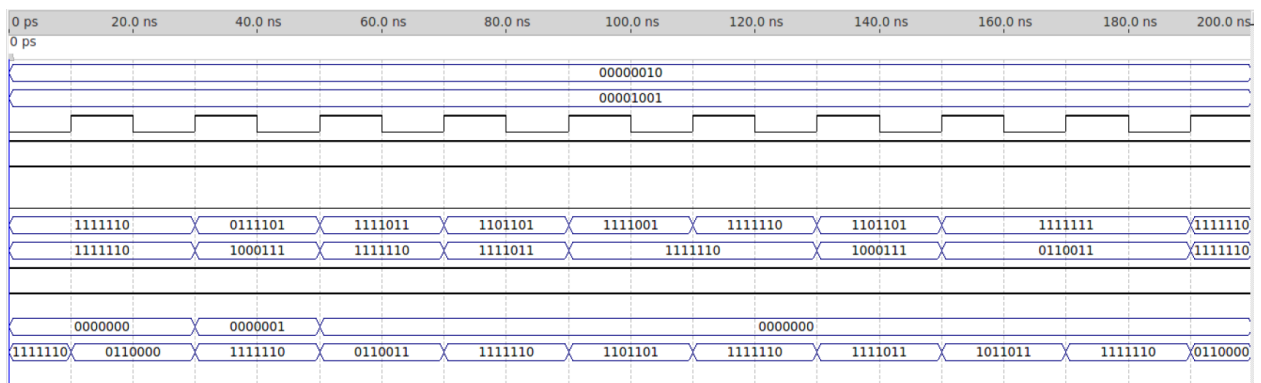
3.2.5: VHDL code 2.2: A VHDL code for the seven-segment display.



3.2.6: Block diagram 2: A block diagram of the second GPU design.

	Name
	A
	B
	Clock
	data_in
	En_decoder
	FSM_reset
	R_first_four
	R_last_four
	Reset_A
	Reset_B
	Sign
	student_ID

3.2.7 Figure 2.2: A table of the input/output pins utilized for the next waveform.



3.2.8 : Waveform 2.2: A waveform of the second GPU design.

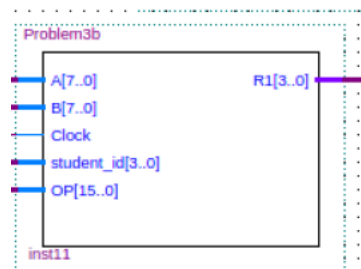
### ALU 3

The following ALU design is dependent on the input student ID number. The ALU receives a 16-bit microcode from the 4-to-16 decoder and an 8-bit student ID number; this student ID is obtained from the FSM component. The 16-bit microcode decides which operation will be executed by the ALU. For this specific case of the ALU design, each microcode will assess the same condition for each microcode received. If the student ID is odd, then the output will be 'y' and if the student ID is even then the output will be 'n'. The even or odd number is determined via a modulus operator, if the remainder of any number divided by an even number is 0 then the input is an even number and odd when remainder equals 1. The seven-segment will display this output as either 'y' or 'n'. Furthermore, ALU will receive a microcode each time the component is clocked, the decoder will receive the state that the FSM is in and send the microcode to the ALU to execute the operations. The result will be either '0000 0000' if the ID is odd or '0000 0001' if the ID is even. The result will be stored in an array of 4-bits. This is

because the higher 4-bits are the same for both results, it would not be useful to store those values, thus, only the lower 4-bits will be stored inside an array, pin R1[3..0]. This result will be assessed by the seven-segment display and the final output will be displayed on the seven-segment display as either 'y' or 'n'. The pins utilized for this ALU design ; the input pins are student\_id[3..0] which is obtained from the FSM (resides in the control unit), pin OP[15..0] is the microcode received from the 4-to-16 bit decoder(resides in the control unit), and Clock will allow the ALU to move onto the next cycle, and the only output pin utilized is R1[3..0] that contains the lower 4-bits of the result. Contrary to the previous two ALU designs that utilized two arrays to display the output, only the lower 4-bits from the result will be assessed further. The higher 4-bits could be ignored as they are identical. In fact, only the lower 2-bits are assessed in this experiment but keeping the first array (R1[3..0]) is useful to prevent any extra adjustments to the seven-segment code.

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

3.3.1: Figure 3.1: Problem 3.



3.3.2: Block Symbol 3: A block symbol of the third ALU component.







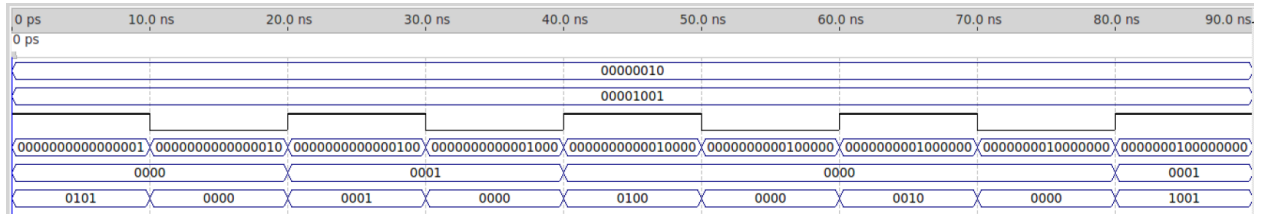
	Name
	A
	B
	Clock
	OP
	R1
	student_id

Table 3.1 : A table of pins utilized for the ALU waveform.



3.3.3: Waveform 3.1 : A waveform for the ALU design concerned with Problem 3.

```

1  LIBRARY ieee;
2  USE ieee.STD_LOGIC_1164.all;
3  USE ieee.STD_LOGIC_UNSIGNED.all;
4  USE ieee.NUMERIC_STD.all;
5
6  ENTITY Problem3b IS
7  PORT (A,B       : IN UNSIGNED(7 DOWNTO 0); -- 8 bit inputs from latches A and B
8        Clock     : IN STD_LOGIC; -- input clock signal
9        student_id : IN UNSIGNED(3 DOWNTO 0); --4 bit student ID from FSM
10       OP        : IN UNSIGNED(15 DOWNTO 0); --16-bit selector for Operation from Decoder
11       R1        : OUT UNSIGNED(3 DOWNTO 0); --lower 4-bits of 8-bit Result Outout
12
13  END Problem3b;
14
15  ARCHITECTURE calculation OF Problem3b IS -- temporary signal declarations
16
17  SIGNAL Reg1,Reg2,Result : UNSIGNED(7 DOWNTO 0) := (OTHERS => '0');
18  SIGNAL Reg4 : UNSIGNED(0 TO 7);
19
20  BEGIN
21
22  PROCESS (Clock,OP)
23  BEGIN
24      IF (rising_edge(Clock)) THEN --Do calculation @ positive edge of clock cycle
25          CASE OP IS
26
27              WHEN "0000000000000001" =>
28
29
29          IF (student_id MOD 2) = 0 THEN
30              Result <= "00000000"; --Y
31
32          ELSE
33              Result <= "00000001"; --N
34          END IF;
35
36          WHEN "0000000000000010" =>
37              IF (student_id MOD 2) = 0 THEN
38                  Result <= "00000000"; --Y
39
40              ELSE
41                  Result <= "00000001"; --N
42              END IF;
43
44          WHEN "0000000000000100" =>
45              IF (student_id MOD 2) = 0 THEN
46                  Result <= "00000000"; --Y
47
48              ELSE
49                  Result <= "00000001"; --N
50              END IF;
51
52          WHEN "0000000000001000" =>
53              IF (student_id MOD 2) = 0 THEN
54                  Result <= "00000000"; --Y
55

```

```

56      ELSE
57          Result <= "00000001"; --N
58      END IF;
59
60      WHEN "0000000000010000" =>
61          IF (student_id MOD 2) = 0 THEN
62              Result <= "00000000"; --Y
63          ELSE
64              Result <= "00000001"; --N
65          END IF;
66
67      WHEN "0000000000100000" =>
68          IF (student_id MOD 2) = 0 THEN
69              Result <= "00000000"; --Y
70          ELSE
71              Result <= "00000001"; --N
72          END IF;
73
74      WHEN "0000000001000000" =>
75          IF (student_id MOD 2) = 0 THEN
76              Result <= "00000000"; --Y
77          ELSE
78              Result <= "00000001"; --N
79          END IF;
80
81      ELSE
82          Result <= "00000001"; --N
83      END IF;
84
85      WHEN "0000000010000000" =>
86          IF (student_id MOD 2) = 0 THEN
87              Result <= "00000000"; --Y
88          ELSE
89              Result <= "00000001"; --N
90          END IF;
91
92      WHEN "0000000100000000" =>
93          IF (student_id MOD 2) = 0 THEN
94              Result <= "00000000"; --Y
95          ELSE
96              Result <= "00000001"; --N
97          END IF;
98
99      WHEN OTHERS =>
100          --Don't care ,do nothing
101
102      END CASE;
103  END IF;
104  END PROCESS;
105
106  R1 <= Result(3 DOWNTO 0);
107
108  END calculation;
109
110

```

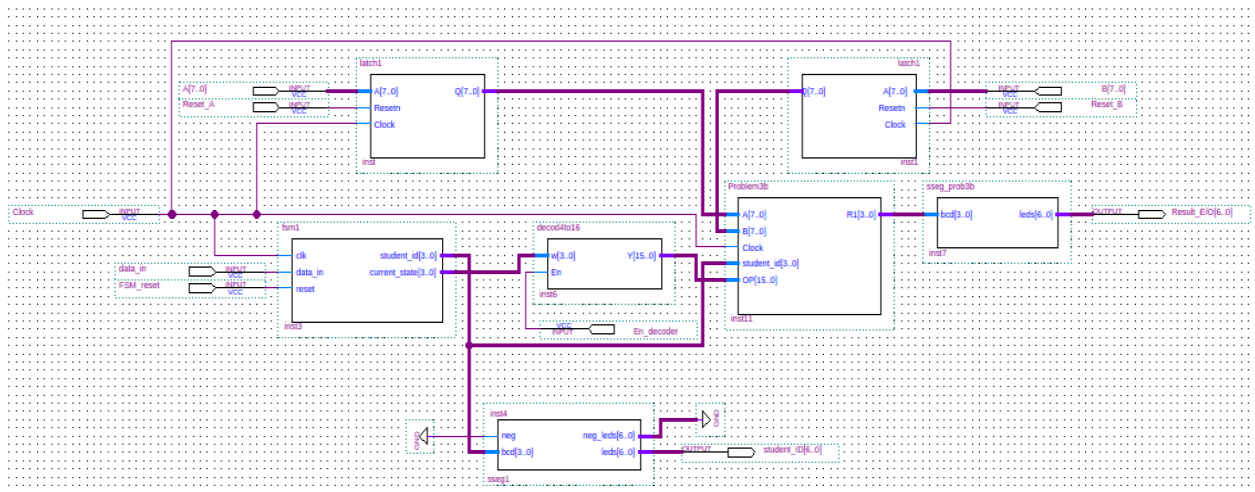
3.3.4: VHDL code 3.1: A VHDL code for the ALU component regarding problem 3.

```

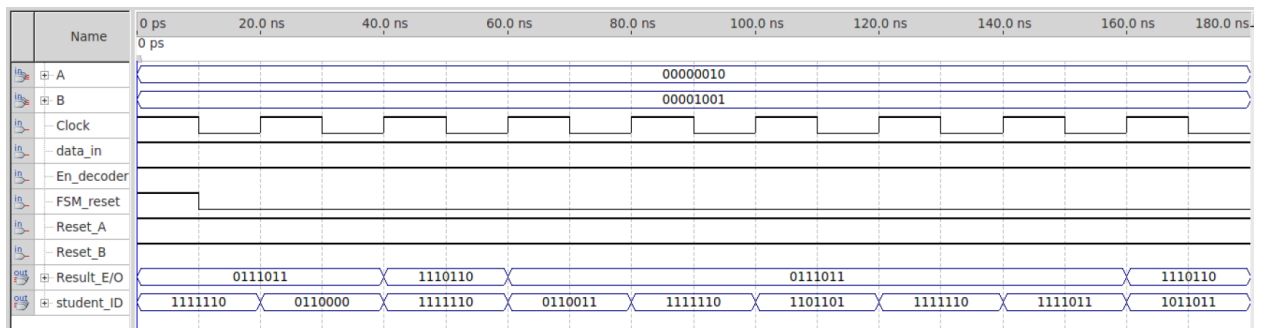
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY sseg_prob3b IS
5  PORT ( bcd :IN STD_LOGIC_VECTOR(3 DOWNTO 0);
6        leds :OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
7        );
8
9  end sseg_prob3b;
10
11 ARCHITECTURE Behavior OF sseg_prob3b IS
12 BEGIN
13   PROCESS (bcd)
14   BEGIN
15
16     CASE bcd IS
17       WHEN "0000" => leds <= "0111011"; --y
18       WHEN "0001" => leds <= "1110110"; --N
19       WHEN OTHERS => leds <= "-----"; --
20     END CASE;
21   END PROCESS;
22 END Behavior;

```

3.3.5: VHDL code 3.3: A VHDL code for the seven-segment display regarding Problem 3.



3.3.6: Block Diagram 3.1: A block diagram of the GPU design concerned with Problem 3.



3.3.7: Waveform 3.2: A waveform of the GPU design.



## **CONCLUSION**

A Central Processing Unit (CPU) is a key component within every computer system. It is known as the brain of a computer system. A CPU is a computer's control unit that receives information, chooses operations to be executed related to that information and sends the result of that operation as an output. Within the CPU, there is a General-Purpose Processing Unit (GPU) that handles all mathematical operations for the CPU. The GPU receives information which it then associates with a microcode that executes a particular operation and outputs a result. There are three different ALU designs involved in this experiment, the 1<sup>st</sup> and the 2<sup>nd</sup> ALU design involves mathematical operation being applied to two 8-bit binary numbers and the result outputs an 8-bit result, the 3<sup>rd</sup> ALU design checks whether the input 8-bit number is even or odd and the result determined using a mathematical operation as well. First the two 8-bit numbers are stored in a storage element, a latch for this experiment, which is sent to the ALU, the ALU receives a second input from the control unit, a microcode. This microcode decides which operation is to be executed by the ALU design. The result is sent to the seven-segment display, which is then associated with a binary-to-decimal or hexadecimal representation of the binary output result. The result of the performed operation is then displayed on a seven-segment display.

The goal of designing a simple GPU design was successful as the waveforms of all ALU designs display the expected output on the waveforms. All waveforms display a seven-segment display output represents the decimal or hexadecimal representation of binary representation of the result.