Ryerson University Department of Electrical and Computer Engineering COE328 – Digital Systems

Course Number	COE 328
Course Title	Digital Systems
Semester/Year	Fall 2023
Instructor	Arghavan Asad
TA Name	Seham Al Abdul Wahid

Lab/Tutorial Report No.	6

Report Title	Design of a Simple General-Purpose Processor

Section No.	9
Group No.	7
Submission Date	Dec 3, 2023
Due Date	Dec 4, 2023

Table Of Contents:

Introduction:

The purpose of the lab is to design and implement a ALU to create a General Purpose Processor. The four key components in creating this device are procuring the input data, a storage unit, a control unit, and the ALU core.The General Purpose Processor consists of , 2 latches, 3 seven segment displays a 4 to 16 decoder, and an FSM (Mealy in this case). The first step is to input two 8 bit binary values, A and B (from the last two numbers of the student ID) into the 2 latches which are a part of the storage unit. The Control unit consists of a 4 to 16 decoder and the mealy machine counter which cycles through the 9 states while the decoder decodes the current state from the FSM to send to the ALU. The ALU takes the 16 bit input from the decoder and does an operation depending on the value from the decoder and returns a 8 bit value. The ALU has 9 different operations which then goes to the SSEGs to display the numbers in hexadecimal.

Components:

Description of Components:

In this lab, there are 5 components that make up the General Purpose Processor. The first component is the latch which takes an 8 bit input and temporarily stores the data to be used by the ALU. The second component is the Mealy machine which counts up from 0 to 8 and outputs the current states into the decoder. The third component is the 4 to 16 decoder which takes the states and outputs the corresponding microcode to the ALU. The fourth component is the ALU which performs the operation corresponding to the microcode and outputs the data to the final component, the SSEG, which takes the output from the ALU and displays it.

Latch 1/Latch 2:

Truth Table:

$A = (53)_{16} = 01010011$ $B = (87)_{16} = 10000111$

Input	Reset		Input	Output (Reset = 0)	Output (Reset = 1)
A	0	1	01010011	00000000	01010011
B	0	1	10000111	00000000	10000111

Truth table for latch 1 and latch 2

Circuit Diagram / Block Diagram:

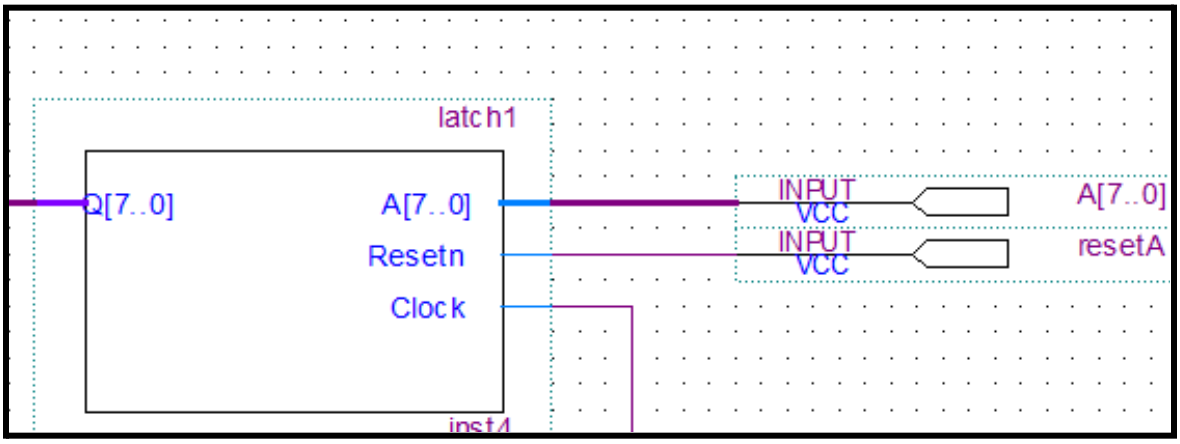
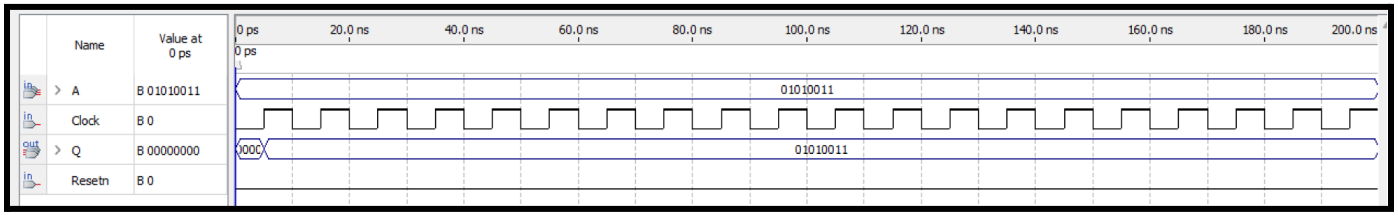


Diagram of Latch 1 (only difference between one and two is that the input is B instead of A)

Waveform:



Waveform for one of the latches

VHDL:

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

VHDL code for both latches

4:16 Decoder:

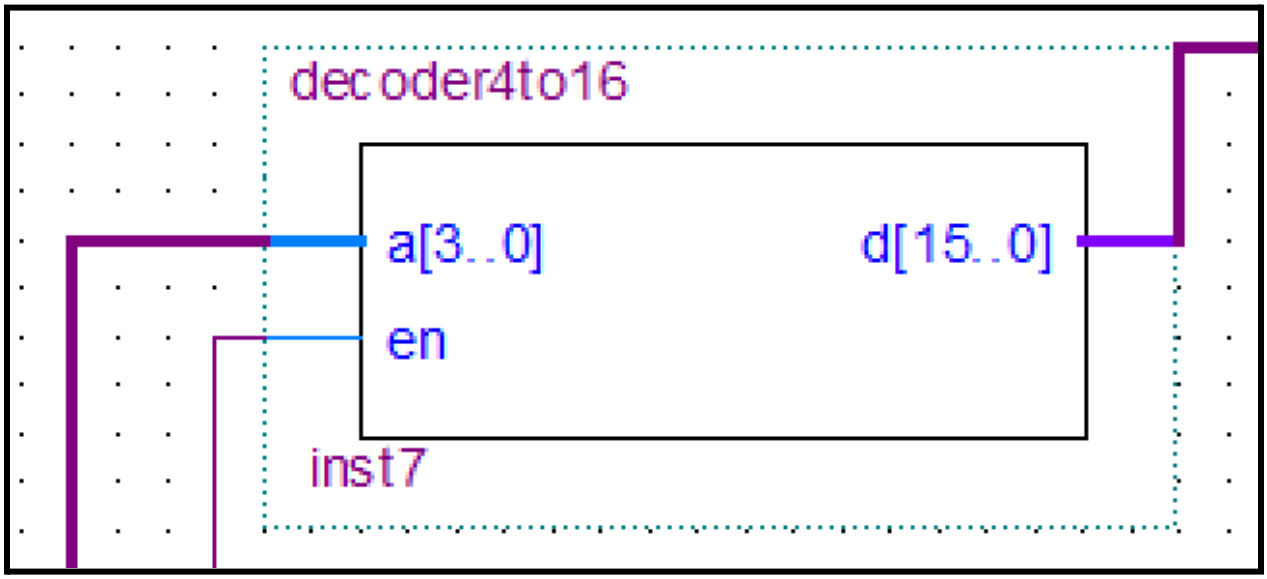
Truth Table:

4:16 Decoder Truth Table

En	Input	Output
1	0000	0000000000000001
1	0001	0000000000000010
1	0010	0000000000000100
1	0011	0000000000001000
1	0100	0000000000010000
1	0101	0000000000100000
1	0110	0000000001000000
1	0111	0000000010000000
1	1000	0000000100000000
1	1001	0000001000000000
1	1010	0000010000000000
1	1011	0000100000000000
1	1100	0001000000000000
1	1101	0010000000000000
1	1110	0100000000000000
1	1111	1000000000000000

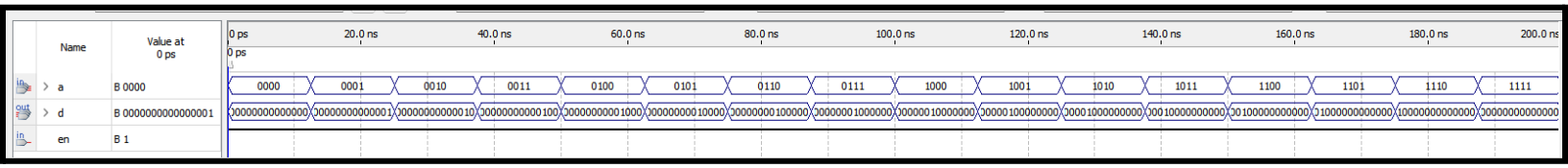
The decoder only works when enable is 1

Circuit Diagram / Block Diagram:



4:16 Decoder block diagram

Waveform:



4:16 Decoder waveform

VHDL:

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity decoder4to16 is
4   port (a : in std_logic_vector(3 downto 0);
5           en : in std_logic;
6           d : out std_logic_vector(15 downto 0));
7   end decoder4to16;
8
9   architecture decoder of decoder4to16 is
10  begin
11      process (a)
12          begin
13              if (en ='0') then
14                  d <= "0000000000000000";
15              else
16                  case a is
17                      when "0000" => d <= "0000000000000001";
18                      when "0001" => d <= "0000000000000010";
19                      when "0010" => d <= "0000000000000100";
20                      when "0011" => d <= "0000000000001000";
21                      when "0100" => d <= "0000000000010000";
22                      when "0101" => d <= "0000000000100000";
23                      when "0110" => d <= "0000000001000000";
24                      when "0111" => d <= "0000000010000000";
25                      when "1000" => d <= "0000000100000000";
26                      when "1001" => d <= "0000001000000000";
27                      when "1010" => d <= "0000010000000000";
28                      when "1011" => d <= "0000100000000000";
29                      when "1100" => d <= "0001000000000000";
30                      when "1101" => d <= "0010000000000000";
31                      when "1110" => d <= "0100000000000000";
32                      when "1111" => d <= "1000000000000000";
33                      when others => d <= "0000000000000000";
34              end case;
35          end if;
36      end process;
37  end decoder;
```

4:16 Decoder VHDL

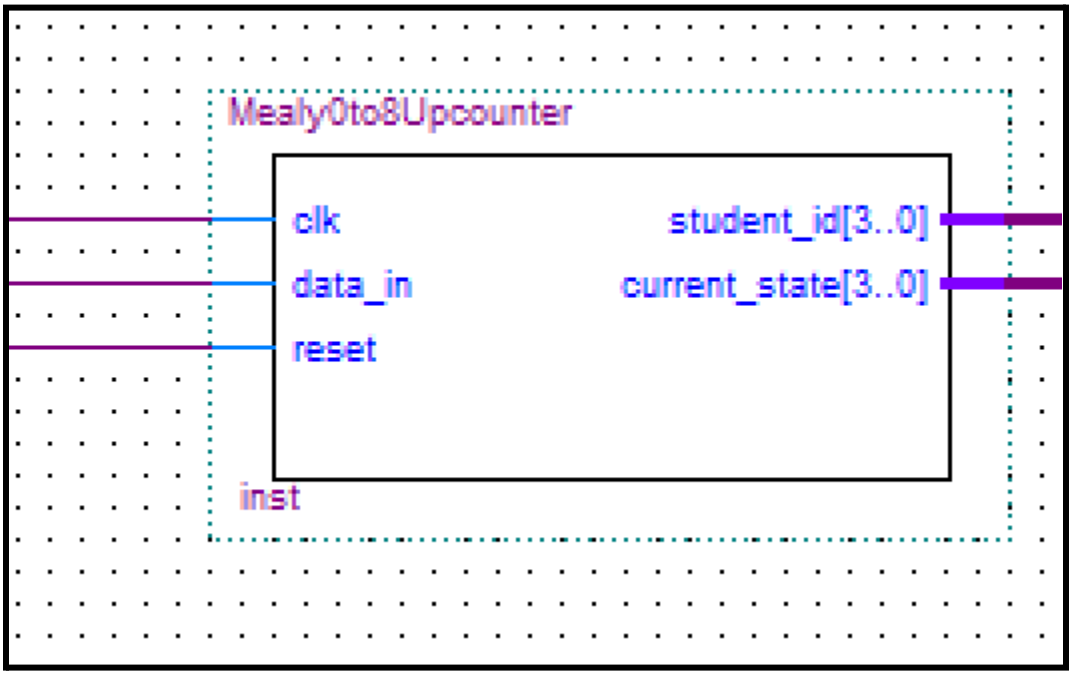
Mealy FSM:

Truth Table:

Current State (Decimal)	Current State (Binary)	Student Number (Decimal)	Student Number (Binary)	Next State (Binary)		Student Number Output (Binary)	
				Data_in = 0	Data_in = 1	Data_in =0	Data_in =1
0	0000	5	0101	0000	0001	0101	0111
1	0001	0	0000	0001	0010	0000	0101
2	0010	1	0001	0010	0011	0001	0000
3	0011	1	0001	0011	0100	0001	0001
4	0100	7	0111	0100	0101	0111	0001
5	0101	5	0101	0101	0110	0101	0111
6	0110	3	0011	0110	0111	0011	0101
7	0111	8	1000	0111	1000	1000	0011
8	1000	7	0111	1000	0000	0111	1000

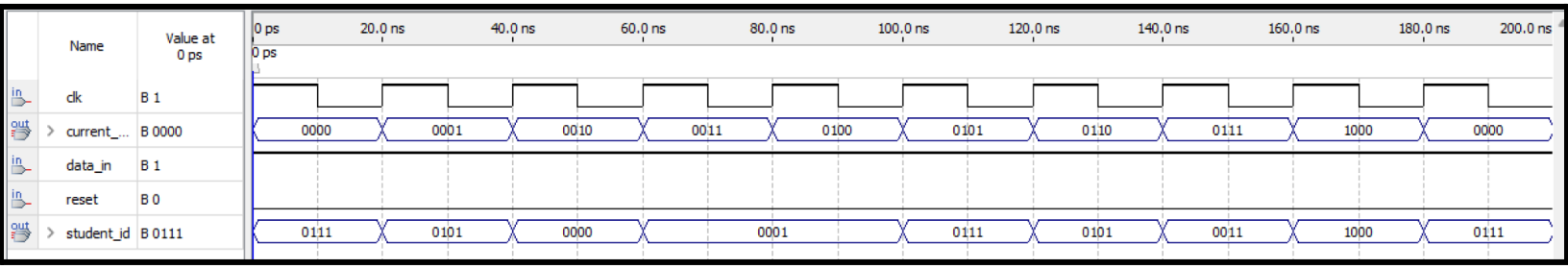
Truth table for Mealy counter. When data in is 0, the state stays the same and the output of the student number is in the same order. When data in is 1, the state moves to the next state and the student number changes accordingly.

Circuit Diagram / Block Diagram:



Block diagram for Mealy counter

Waveform:



Waveform for Mealy counter when data_in equals 1

VHDL:

```
61              elsif (data_in = '0') then
62                 yfsm <= s4;
63                 end if;
64
65           when s5 =>
66              if (data_in = '1') then
67              yfsm <= s6;
68              elsif (data_in = '0') then
69                 yfsm <= s5;
70                 end if;
71
72           when s6 =>
73              if (data_in = '1') then
74              yfsm <= s7;
75              elsif (data_in = '0') then
76                 yfsm <= s6;
77                 end if;
78
79           when s7 =>
80              if (data_in = '1') then
81              yfsm <= s8;
82              elsif (data_in = '0') then
83                 yfsm <= s7;
84                 end if;
85
86           when s8 =>
87              if (data_in = '1') then
88              yfsm <= s0;
89              elsif (data_in = '0') then
90                 yfsm <= s8;
91                 end if;
92
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity Mealy0to8Upcounter 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 Mealy0to8Upcounter 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 --how FSM moves (upcounter, cyling through states 0 to 8 )
            when s0 =>
                if (data_in = '1') then
                yfsm <= s1;
                elsif (data_in = '0') then
                yfsm <= s0;
                end if;
```

```
            yfsm <= s0;
            end if;

         when s1 =>
            if (data_in = '1') then
            yfsm <= s2;
            elsif (data_in = '0') then
            yfsm <= s1;
            end if;

         when s2 =>
            if (data_in = '1') then
            yfsm <= s3;

            elsif (data_in = '0')
            then
            yfsm <= s2;
            end if;

         when s3 =>
            if (data_in = '1') then
            yfsm <= s4;

            elsif (data_in = '0') then
            yfsm <= s3;
            end if;

         when s4 =>
            if (data_in = '1') then
            yfsm <= s5;
            elsif (data_in = '0') then
            yfsm <= s4;
```

```
      end case;
      end if;
   end process;

   process (yfsm, data_in)
   begin
      case yfsm is

         when s0 => --at s0
            current_state <= "0000"; --default current state
            if (data_in = '1') then
            student_id <= "0111"; --7
            elsif (data_in = '0') then
            student_id <= "0101"; --5
            end if;

         when s1 => --at s1
            current_state <= "0001"; --default current state
            if (data_in = '1') then
            student_id <= "0101"; --5
            elsif (data_in = '0') then
            student_id <= "0000"; --0
            end if;

         when s2 => --at s2
            current_state <= "0010"; --default current state
            if (data_in = '1') then
            student_id <= "0000"; --0
            elsif (data_in = '0') then
            student_id <= "0001"; --1
            end if;
```

```
124             end if;
125
126          when s3 => --at s3
127             current_state <= "0011"; --default current state
128             if (data_in = '1') then
129             student_id <= "0001"; --1
130             elsif (data_in = '0') then
131             student_id <= "0001"; --1
132             end if;
133
134          when s4 => --at s4
135             current_state <= "0100"; --default current state
136             if (data_in = '1') then
137             student_id <= "0001"; --1
138             elsif (data_in = '0') then
139             student_id <= "0111"; --7
140             end if;
141
142          when s5 => --at s5
143             current_state <= "0101"; --default current state
144             if (data_in = '1') then
145             student_id <= "0111"; --3
146             elsif (data_in = '0') then
147             student_id <= "0101"; --5
148             end if;
149
150          when s6 => --at s6
151             current_state <= "0110"; --default current state
152             if (data_in = '1') then
153             student_id <= "0101"; --5
154             elsif (data_in = '0') then
155             student id <= "0011"; --3
```

```
148             end if;
149
150          when s6 => --at s6
151             current_state <= "0110"; --default current state
152             if (data_in = '1') then
153             student_id <= "0101"; --5
154             elsif (data_in = '0') then
155             student_id <= "0011"; --3
156             end if;
157
158          when s7 => --at s7
159             current_state <= "0111"; --default current state
160             if (data_in = '1')then
161             student_id <= "0011"; --3
162             elsif (data_in = '0') then
163             student_id <= "1000"; --8
164             end if;
165
166          when s8 => --at s8
167             current_state <= "1000"; --default current state
168             if (data_in = '1') then
169             student_id <= "1000"; --5
170             elsif (data_in = '0') then
171             student_id <= "0111"; --7
172             end if;
173       end case;
174
175    end process;
176
177 end fsm;
178
```

VHDL code for Mealy counter

SSEG:

Truth Table:

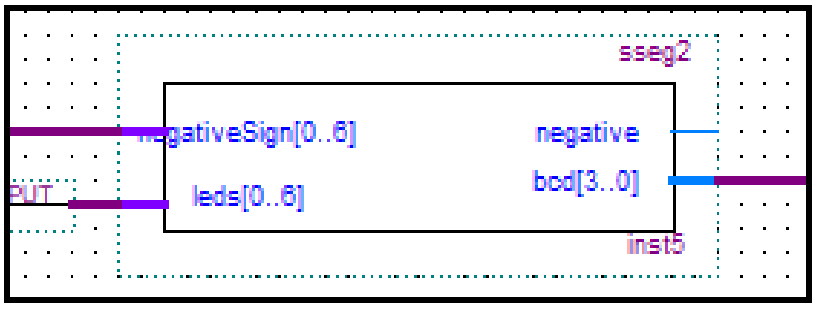
Input	Output (abcdefg)	Hexadecimal
0000	0000000	0
0001	0110000	1
0010	1101101	2
0011	1111001	3
0100	0110011	4
0101	1011011	5
0110	1011111	6
0111	1110000	7
1000	1111111	8
1001	1110011	9
1010	1110111	A
1011	0011111	B
1100	0001101	C
1101	0111101	D
1110	1001111	E
1111	1000111	F

Truth table for the seven segment display for part 1 and 2

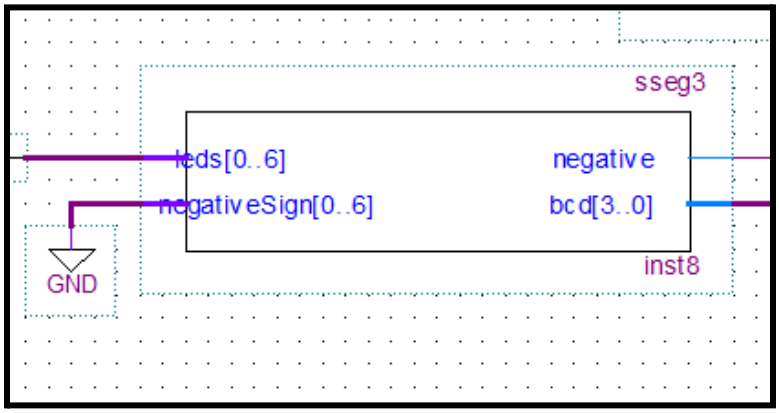
Input	Output (abcdefg)	Y/N
0000	1110110	N
0001	0110011	Y

Truth table for the seven segment display for part 3

Circuit Diagram / Block Diagram:

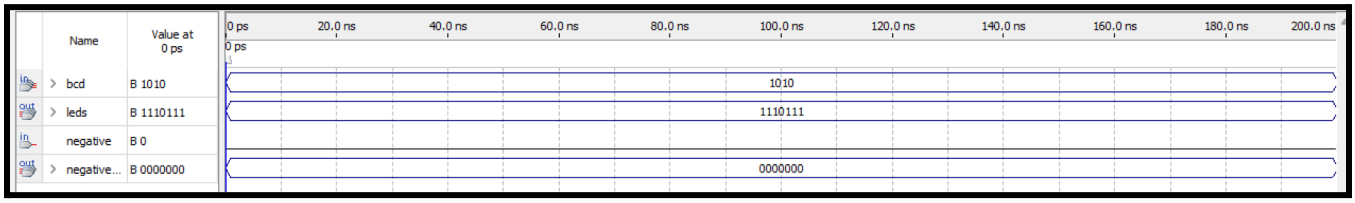


Block diagram for the seven segment display for part 1 and 2

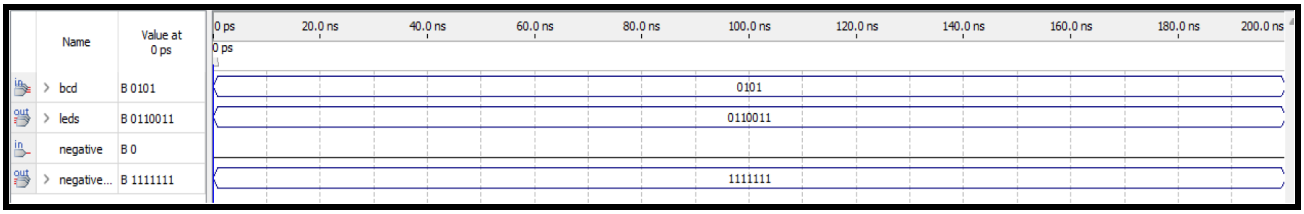


Block diagram for the seven segment display for part 3

Waveform:



Waveform for SSEG when 1010 or 10 is inputted for part 1 and 2



Waveform for SSEG when 0101 or 5 is inputted for part 3

VHDL:

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

ENTITY sseg2 IS
    PORT (negative : IN STD_LOGIC;
            bcd : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            leds, negativeSign: OUT STD_LOGIC_VECTOR(0 TO 6));
END sseg2;

ARCHITECTURE Behavior OF sseg2 IS
BEGIN
    PROCESS (bcd)
    BEGIN
        if negative = '1' then
            negativeSign <= "0000001";
         else
            negativeSign <="0000000";

             END IF;
        CASE bcd IS -- abcdefg
            WHEN "0000" => leds <= NOT"0000001";
            WHEN "0001" => leds <= NOT"1001111";
            WHEN "0010" => leds <= NOT"0010010";
            WHEN "0011" => leds <= NOT"0000110";
            WHEN "0100" => leds <= NOT"1001100";
            WHEN "0101" => leds <= NOT"0100100";
            WHEN "0110" => leds <= NOT"0100000";
            WHEN "0111" => leds <= NOT"0001111";
            WHEN "1000" => leds <= NOT"0000000";
            WHEN "1001" => leds <= NOT"0001100";
            WHEN "1010" => leds <= NOT"0001000";
            WHEN "1011" => leds <= NOT"1100000";
            WHEN "1100" => leds <= NOT"1110010";
            WHEN "1101" => leds <= NOT"1000010";
            WHEN "1110" => leds <= NOT"0110000";
            WHEN "1111" => leds <= NOT"0111000";
            WHEN OTHERS => leds <= "-------";
        END CASE;
    END PROCESS;
END Behavior;
```

VHDL code for SSEG part 1 and 2

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

ENTITY sseg3 IS
    PORT (negative : IN STD_LOGIC;
            bcd : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
            leds, negativeSign: OUT STD_LOGIC_VECTOR(0 TO 6));
END sseg3;

ARCHITECTURE Behavior OF sseg3 IS
BEGIN
    PROCESS (bcd)
    BEGIN
        if negative = '1' then
            negativeSign <= "1111110";
        else
            negativeSign<= "1111111";

        END if;

        CASE bcd IS -- abcdefg
            WHEN "0000" => leds <= "1110110";  -- n
            WHEN "0001" => leds <= "0110011";  -- y
            WHEN OTHERS => leds <= "-------";
        END CASE;
    END PROCESS;
END Behavior;
```

VHDL code for SSEG part 3

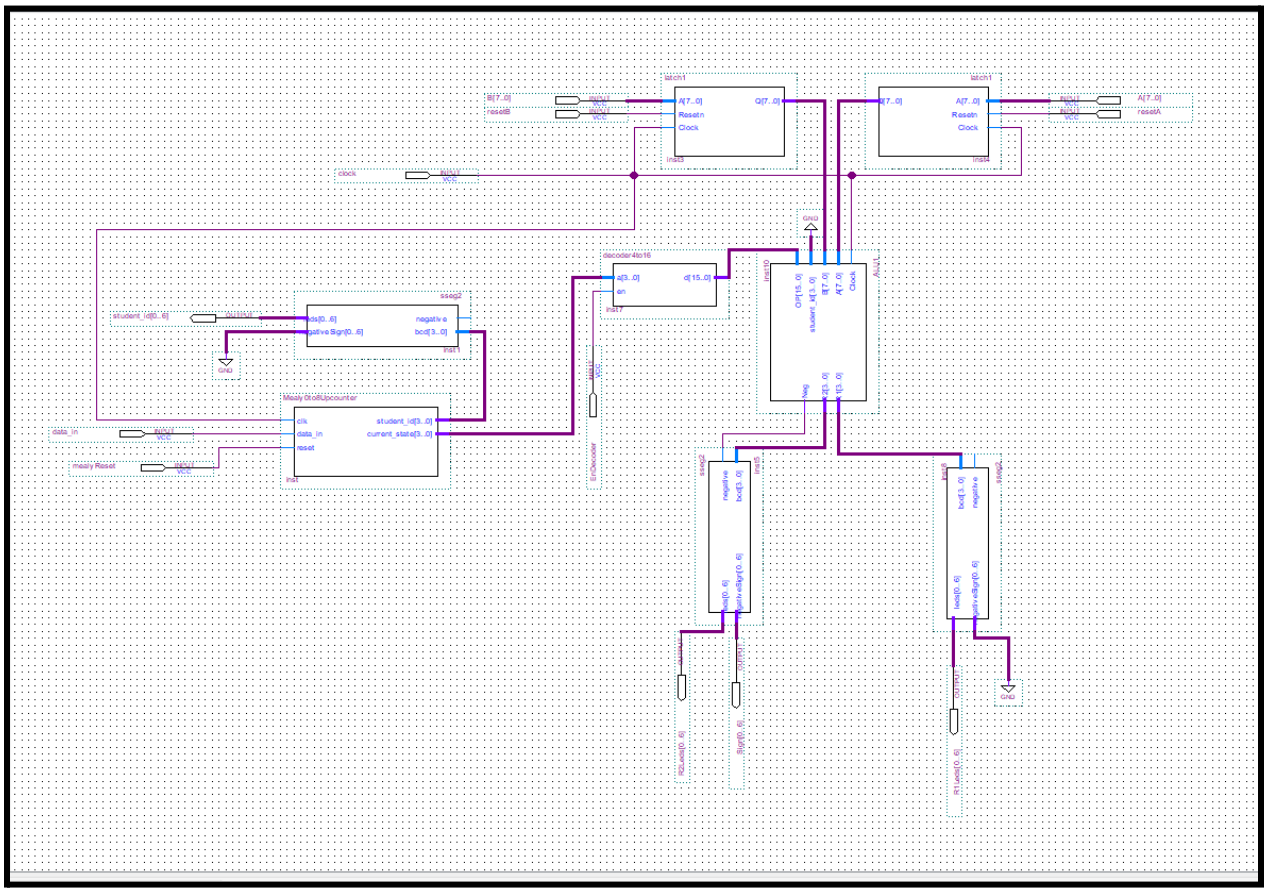
ALU_1:

Description:

The General Purpose Processor (GPP) consists of storage units, a ALU, a control unit and a seven segment display unit. The most important component is the ALU as it does the computation of the imputed values. The up-counting from the FSM passes through the decoder which interprets the microcode and influences the type of operation the ALU performs to the two inputs (A and B) from the two latches. After the operation is complete, the outcome is passed to the two SSEGs which, using LEDs, display the hexadecimal equivalent of the output. The GPP has many inputs and outputs. The first input comes from the user who inputs the last four digits of their student number in binary from hexadecimal. The first two of the four digits are stored into A which holds 8 bits while the last two are stored in B. The two inputs are sent to the latches

which output A and B to the ALU. The Mealy machine cycles through 9 states each corresponding to the student number. The states of the machine are imputed into the decoder which turns the states into microcodes that are passed to the ALU. The ALU reads the microcodes and performs the specific operation to the A and B inputs before outputting to the SSEGs which displays the hexadecimal equivalent of the ALU output. The first SSEG displays the first hexadecimal equivalent, the second displays the last hexadecimal equivalent and the last displays the negative sign if the number is negative. The purpose of the inputs A and B are to be inputs for the arithmetic operations done by the ALU. The control unit outputs the operation that will be performed on the inputs A and B. ALU outputs the final answer which will be inputted to the SSEG to be displayed in hexadecimal format.

Block Diagram:



Block diagram of GPP1 and ALU1

Table of Microcode:

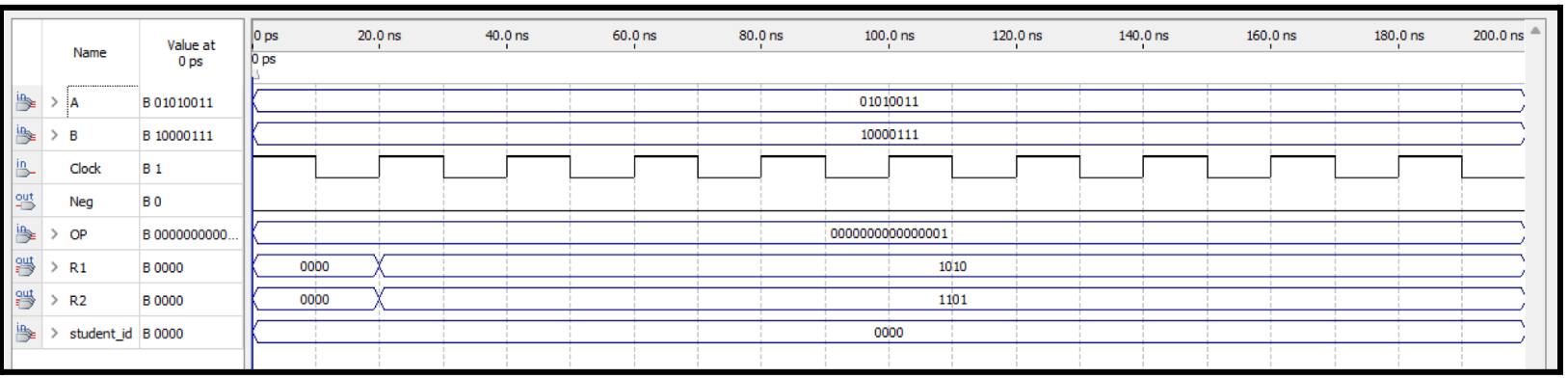
Function	Microcode (From Decoder)	Operation
1	0000000000000001	Sum(A,B)
2	0000000000000010	Diff(A,B)
3	0000000000000100	$\overline{A}$
4	0000000000001000	$\overline{A \bullet B}$
5	0000000000010000	$\overline{A + B}$
6	0000000000100000	$A \bullet B$
7	0000000001000000	$A \oplus B$
8	0000000010000000	$A + B$
9	0000000100000000	$\overline{A \oplus B}$

Table of microcodes for GPP 1 for part 1

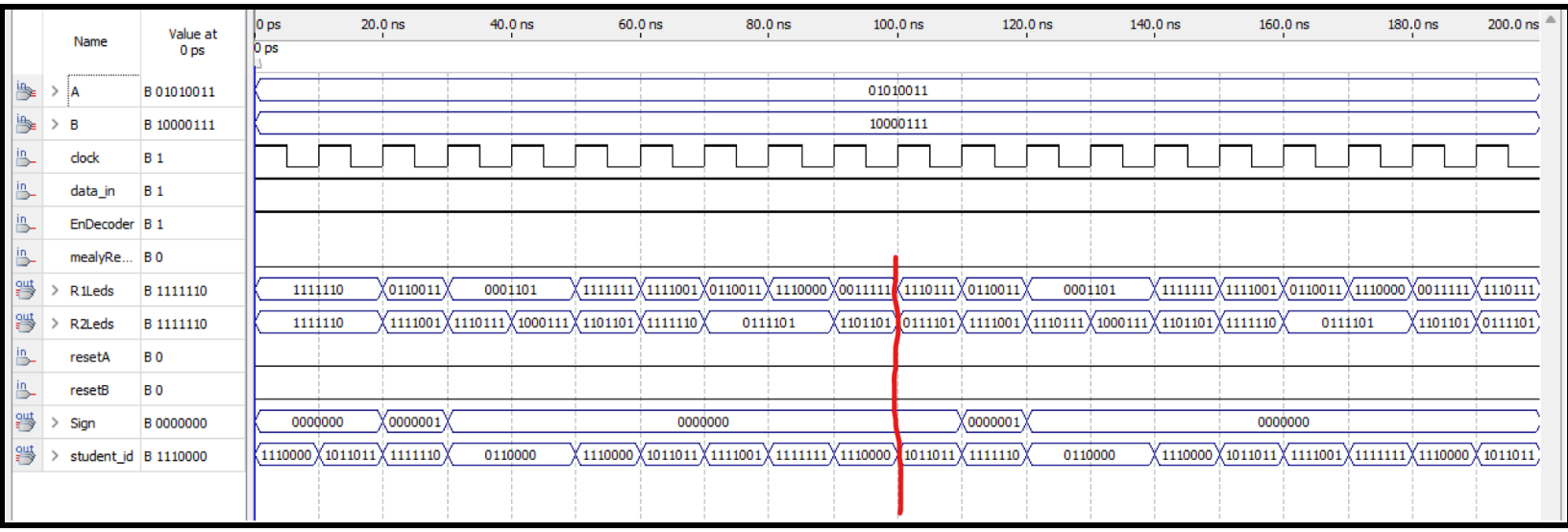
Truth Table for Problem Set 1						
Input of Latch1 (A)				**Input of Latch2 (B)**		
Hexadecimal		53		Hexadecimal		87
Binary		01010011		Binary		10000111
ASU1		Result in Binary		abcdefg		
Function #	Operation	Result 2	Result 1	SSEG2	SSEG1	Sign SSEG
1	Sum(A,B)	1101	1010	D: 0111101	A:1110111	0000000
2	Diff(A,B)	0011	0100	3: 1111001	4: 0110011	0000001
3	$\overline{A}$	1010	1100	A: 1110111	C:0001101	0000000
4	$\overline{A \bullet B}$	1111	1100	F: 1000111	C:0001101	0000000
5	$\overline{A + B}$	0010	1000	2: 1101101	8:1111111	0000000
6	$A \bullet B$	0000	0011	0: 1111110	3: 1111001	0000000
7	$A \oplus B$	1101	0100	D: 0111101	4: 0110011	0000000
8	$A + B$	1101	0111	D: 0111101	7: 1110000	0000000
9	$\overline{A \oplus B}$	0010	1011	2:1101101	B: 0011111	0000000

Truth table for GPP 1

Waveform:



ALU1 Waveform running the sum function



GPP1 Waveform (after the red line the numbers are correct)

VHDL:

```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 Result 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"=>
      -- Do Addition for Reg1 and Reg2
      Result <= Reg1 + Reg2;
      Neg <= '0';

      when "0000000000000010" =>
```

```vhdl
      when "0000000000000010" =>
      if Reg1<Reg2 then
      Result <= Reg2 - Reg1;
      neg <= '1';
      else
      Result <= Reg1 - Reg2;
      neg <= '0';
      end if;

      WHEN "0000000000000100"=>
      --Do Inverse
      Result <= NOT Reg1;
      Neg <= '0';

      WHEN "0000000000001000"=>
      -- Do Boolean NAND
      Result <= NOT(Reg1 AND Reg2);
      Neg <= '0';

      WHEN "0000000000010000"=>
      -- Do Boolean NOR
      Result <= NOT(Reg1 OR Reg2);
      Neg <= '0';

      WHEN "0000000000100000"=>
      -- Do Boolean AND
      Result <= Reg1 AND Reg2;
      Neg <= '0';

      WHEN "0000000001000000"=>
      -- Do Boolean XOR
```

```
54          Neg <= '0';
55
56          WHEN "0000000000100000"=>
57          -- Do Boolean AND
58          Result <= Reg1 AND Reg2;
59          Neg <= '0';
60
61          WHEN "0000000001000000"=>
62          -- Do Boolean XOR
63          Result <= Reg1 XOR Reg2;
64          Neg <= '0';
65
66          WHEN "0000000010000000"=>
67          -- Do Boolean OR
68          Result <= Reg1 OR Reg2;
69          Neg <= '0';
70
71          WHEN "0000000100000000"=>
72          -- Do Boolean XNOR
73          Result <= Reg1 XNOR Reg2;
74          Neg <= '0';
75
76          WHEN OTHERS =>
77          -- Don't care, do nothing
78      end case;
79    end if;
80  end process;
81  R1 <= Result(3 downto 0); -- Since the output seven segments can
82  R2 <= Result(7 downto 4); -- only 4-bits, split the 8-bit to two 4-bits.
83  end calculation;
84
```

VHDL Code for ALU 1

ALU_2:

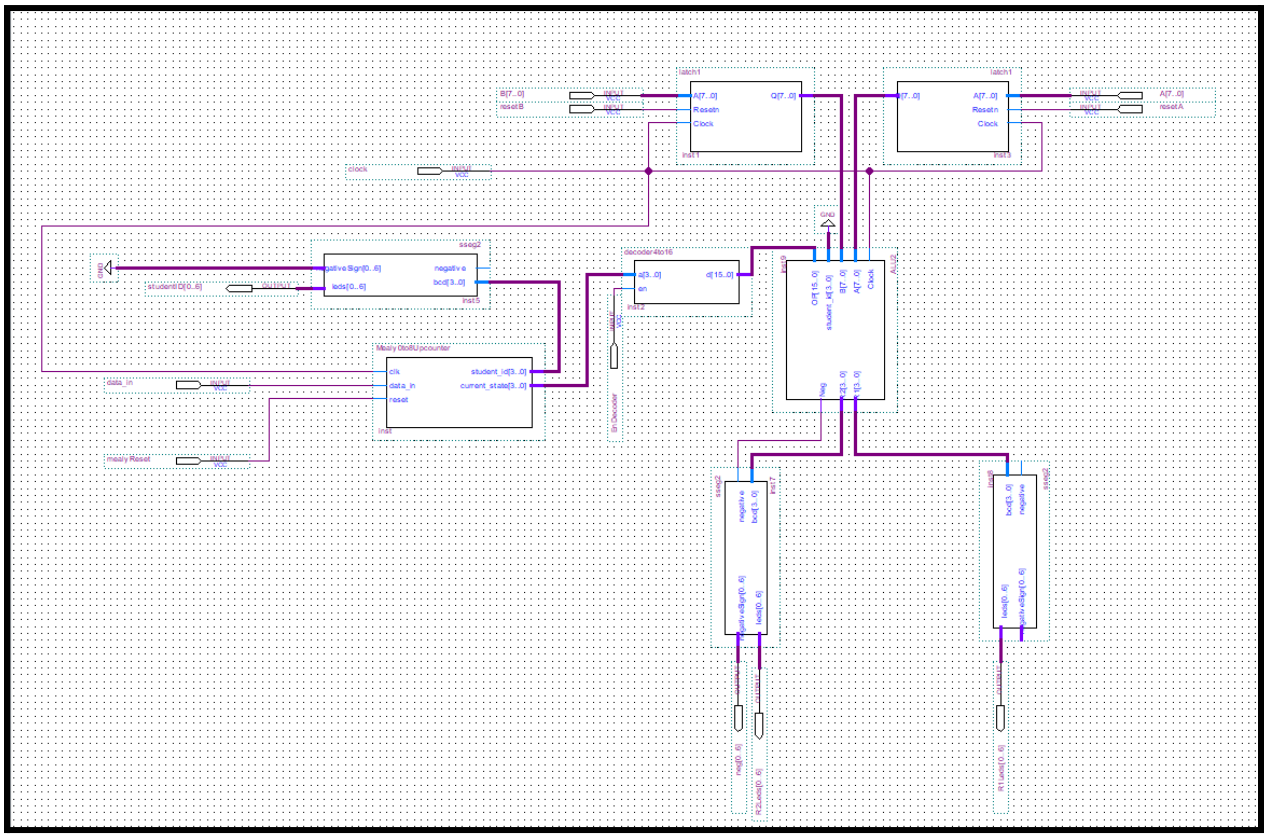
(Function g)

Description:

The purpose of the ALU2 is to perform operations that are different to ALU1 based on the function given. The ALU2 is similar to ALU1 as they both perform an operation based on the microcode sent from the decoder. After every rising edge, the Mealy machine outputs a state from 0 to 8 which the decoder reads and sends the equivalent microcode to ALU2 so that it can perform the operation. The only difference between ALU1 and ALU2 is that the operations which are based on the table g, are different for each microcode. ALU2 takes in 2 inputs, A and B from the latches as well as the decoder input to perform the proper operation. The output of the ALU2 is split into two 4 bit representations to be sent to two SSEGs so that they can display the hexadecimal equivalent. The purpose of the inputs A and B are to be inputs for the arithmetic operations done by the ALU. The control unit outputs the operation that will be performed on the

inputs A and B. ALU outputs the final answer which will be inputted to the SSEG to be displayed in hexadecimal format.

Block Diagram:



Block diagram of GPP1 and ALU1

Table of Microcode:

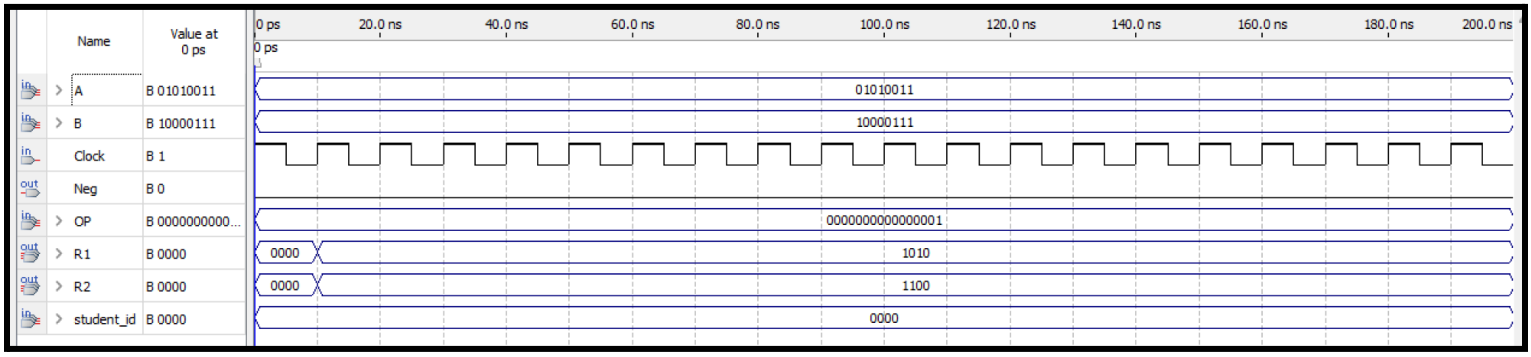
Function	Microcode (From Decoder)	Operation
1	0000000000000001	Invert Sig-bit A
2	0000000000000010	SHL A by 4, all 1 after
3	0000000000000100	Invert Upper 4-bit B
4	0000000000001000	Min(A, B)
5	0000000000010000	Sum(A,B) add 4
6	0000000000100000	A add 3
7	0000000001000000	Even-bits A = Even-bits B
8	0000000010000000	$\overline{A \oplus B}$
9	0000000100000000	ROR B by 3 bits

Table of microcodes for GPP 1 for part 1

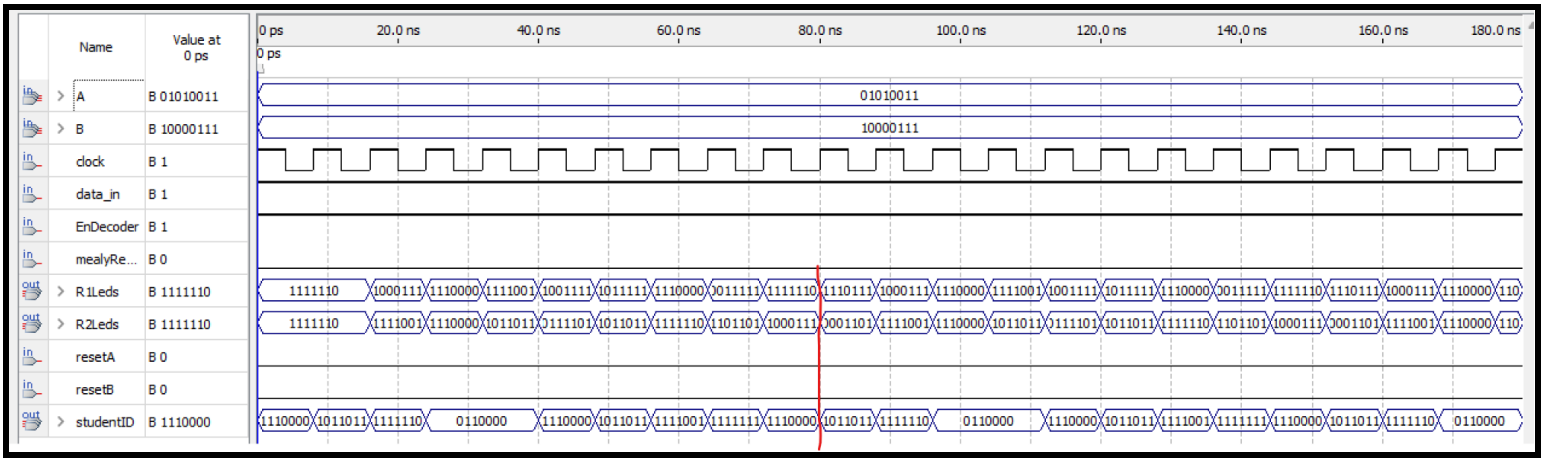
Truth Table for Problem Set 2					
Input of Latch1 (A)			Input of Latch2 (B)		
Hexadecimal	01		Hexadecimal	38	
Binary	00000001		Binary	00111000	
ASU2		Result in Binary		abcdefg	
Function #	Operation	Result 2	Result 1	SSEG2	SSEG1
1	Invert Sig-bit A	1100	1010	C:0001101	A:1110111
2	SHL A by 4, all 1 after	0011	1111	3:1111001	F:1000111
3	Invert Upper 4-bit B	0111	0111	7:1110000	7:1110000
4	Min(A, B)	0101	0011	5:1011011	3:1111001
5	Sum(A,B) add 4	1101	1110	D:0111101	E:1001111
6	A add 3	0101	0110	5:1011011	6:1011111
7	Even-bits A = Even-bits B	0000	0111	0:1111110	7:1110000
8	$\overline{A \oplus B}$	0010	1011	2:1101101	B:0011111
9	ROR B by 3 bits	1111	0000	F:1000111	0:0000000

Truth table for GPP2

Waveform:



Waveform for ALU2 inverting the bit significance of A (operation 1)



Waveform for GPP 2 (after the red line the numbers are correct)

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 Result 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 (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(0)<= Reg1(7);
                  Result(1)<= Reg1(6);
                  Result(2)<= Reg1(5);
                  Result(3)<= Reg1(4);
                  Result(4)<= Reg1(3);
                  Result(5)<= Reg1(2);
                  Result(6)<= Reg1(1);
                  Result(7)<= Reg1(0);
                  --Neg<='0';


            WHEN "0000000000000010" =>

                   Result<= Reg1 sll 4;
                   Result(0)<='1';
                   Result(1)<='1';
                   Result(2)<='1';
                   Result(3)<='1';

            WHEN "0000000000000100" =>
                  Result(7)<= Not Reg2(7);
                  Result(6)<= Not Reg2(6);
                  Result(5)<= Not Reg2(5);
                  Result(4)<= Not Reg2(4);
                  Result(3)<=  Reg2(3);
```

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


            WHEN "0000000000001000" =>
                    if(Reg1<=Reg2) then
                        Result<= Reg1;
                    else
                        Result<= Reg2;
                    end iF;

            WHEN "0000000000010000" => Result <= Reg1 + Reg2+ "00000100"; --note before was 4

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

            WHEN "0000000001000000" =>

                  Result(0) <= Reg2(0);
                  Result(1) <= Reg1(1);
                  Result(2) <= Reg2(2);
                  Result(3) <= Reg1(3);
                  Result(4) <= Reg2(4);
                  Result(5) <= Reg1(5);
                  Result(6) <= Reg2(6);
                  Result(7) <= Reg1(7);


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

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

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


            end case;
      end if;
   end process;
R1 <= Result(3 downto 0); -- Since the output seven segments can
R2 <= Result(7 downto 4); -- only 4-bits, split the 8-bit to two 4-bits.
end calculation;
```

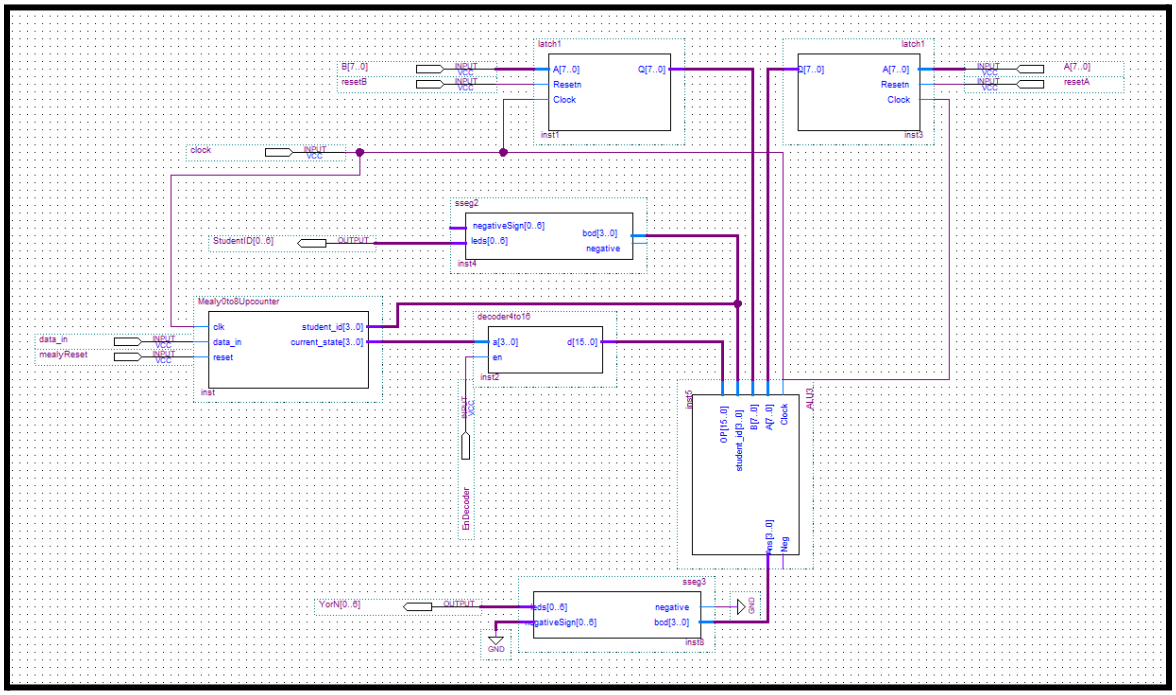
VHDL code for ALU 2

ALU_3:

Description:

The purpose of ALU3 is to compare one of the 2 digits of A, in this case $(53)_{16}$, with each number of the student ID. If the student ID number matches either 5 or 3 in binary, the SSEG will display a “y” otherwise it will display a “n”. The rising edge of the clock allows the Mealy machine to cycle through all the states of the student number and ALU3 compares the two numbers. Once the ALU3 is finished comparing the numbers, it will output either “00000000” or “00000001” to the SSEG which interprets “00000000” as “n” and “00000001” as “y”. The SSEG then displays this output. The purpose of the input A is that is allows the ALU3 to compare it with the student number. The control unit outputs the operation that will be performed on the input A. The ALU3 output is the result of the operation done by the ALU3 which is imputed to the SSEG. The purpose of the SSEG is to display “y” or “n” according to if the numbers match.

Block Diagram:



Block diagram for GPP3 and ALU3

Table of Microcode:

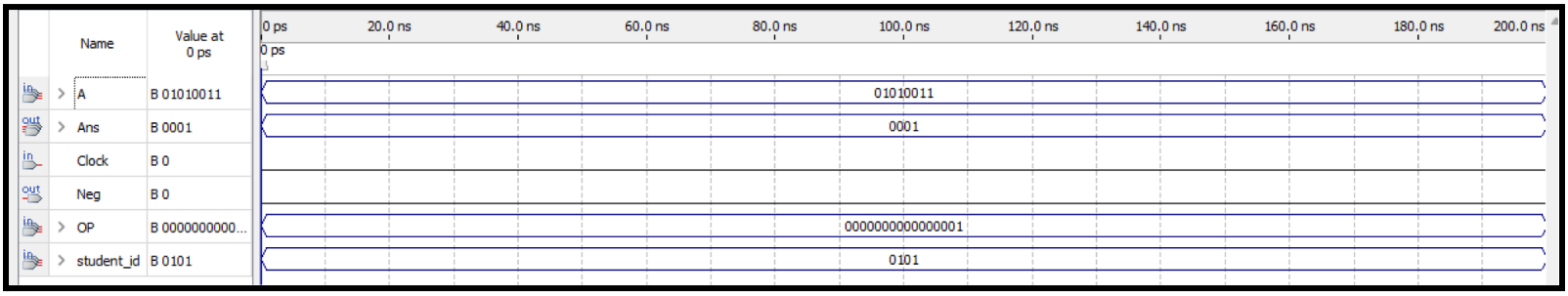
Function	Microcode (From Decoder)	Operation
1	0000000000000001	Compare the first number of student number with both bits of A in hexadecimal
2	0000000000000010	Compare the second number of student number with both bits of A in hexadecimal
3	0000000000000100	Compare the third number of student number with both bits of A in hexadecimal
4	0000000000001000	Compare the fourth number of student number with both bits of A in hexadecimal
5	0000000000010000	Compare the fifth number of student number with both bits of A in hexadecimal
6	0000000000100000	Compare the sixth number of student number with both bits of A in hexadecimal
7	0000000001000000	Compare the seventh number of student number with both bits of A in hexadecimal
8	0000000010000000	Compare the eighth number of student number with both bits of A in hexadecimal
9	0000000100000000	Compare the ninth number of student number with both bits of A in hexadecimal

Table of microcode for GPP3

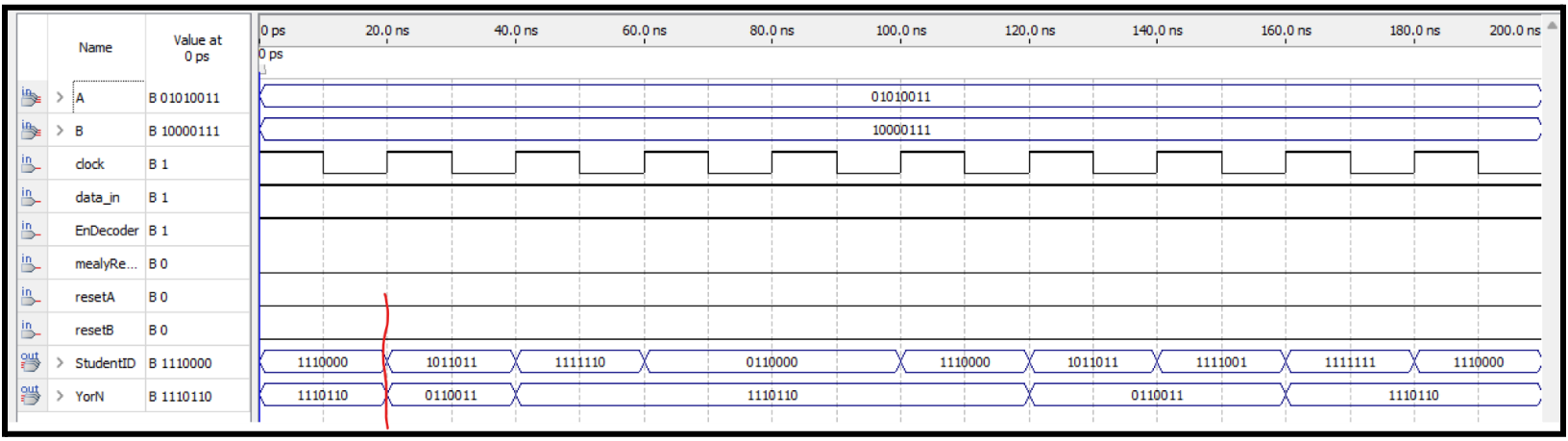
Truth Table for Problem Set 3						
Input of Latch1 (A)						
Hexadecimal		53				
Binary		01010011				
ASU3		A in Binary		Output in Binary	SSEG	
Function #	Student ID	First 4-bits	Last 4-bits		abcdefg	Y or N
1	5: 0101	0101	0011	0001	0110011	Y
2	0: 0000	0101	0011	0000	1110110	N
3	1: 0001	0101	0011	0000	1110110	N
4	1: 0001	0101	0011	0000	1110110	N
5	7: 0111	0101	0011	0000	1110110	N
6	5: 0101	0101	0011	0001	0110011	Y
7	3: 0011	0101	0011	0001	0110011	Y
8	8: 1000	0101	0011	0000	1110110	N
9	7: 0111	0101	0011	0000	1110110	N

Truth table for GPP3

Waveform:



Waveform for ALU3 testing if 5 (0101) is equal to one of the two bits of A (01010011)



Waveform for GPP3 (after the red line the numbers are correct)

VHDL:

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.all;
3   USE IEEE.STD_LOGIC_UNSIGNED.all;
4   USE IEEE.NUMERIC_STD.all;
5   ENTITY ALU3 IS
6   PORT (Clock: IN STD_LOGIC;
7         A: IN UNSIGNED(7 DOWNTO 0);
8         student_id: IN UNSIGNED(3 DOWNTO 0);
9         OP: IN UNSIGNED(15 DOWNTO 0);
10        Neg: OUT STD_LOGIC;
11        Ans: OUT UNSIGNED(3 DOWNTO 0));
12
13  END ALU3;
14
15  ARCHITECTURE calculation of ALU3 IS
16  SIGNAL Reg1, Reg2, Result: UNSIGNED(7 DOWNTO 0):= (OTHERS =>'0');
17  SIGNAL Reg4: UNSIGNED (0 TO 7);
18  BEGIN
19
20  Reg1 <= A;
21
22  PROCESS(Clock, OP)
23  BEGIN
24  CASE OP IS
25     WHEN "0000000000000001" =>
26     if (Reg1(3 downto 0) = student_id) then
27        Result <= "00000001";
28     elsif (Reg1(7 downto 4) = student_id) then
29        Result <= "00000001";
30     else
31        Result <= "00000000";
32     end if;
33
34     WHEN "0000000000000010" =>
35     if (Reg1(3 downto 0) = student_id) then
36        Result <= "00000001";
37     elsif (Reg1(7 downto 4) = student_id) then
38        Result <= "00000001";
39     else
40        Result <= "00000000";
41     end if;
42
43
44     WHEN "0000000000000100" =>
45     if (Reg1(3 downto 0) = student_id) then
46        Result <= "00000001";
47     elsif (Reg1(7 downto 4) = student_id) then
48        Result <= "00000001";
49     else
50        Result <= "00000000";
51     end if;
52
53     WHEN "0000000000001000" =>
54     if (Reg1(3 downto 0) = student_id) then
55        Result <= "00000001";
56     elsif (Reg1(7 downto 4) = student_id) then
57        Result <= "00000001";
58     else
59        Result <= "00000000";
60     end if;
61
62     WHEN "0000000000010000" =>
63     if (Reg1(3 downto 0) = student_id) then
64        Result <= "00000001";
65     elsif (Reg1(7 downto 4) = student_id) then
66        Result <= "00000001";
67     else
68        Result <= "00000000";
69     end if;
70
71     WHEN "0000000000100000" =>
72     if (Reg1(3 downto 0) = student_id) then
73        Result <= "00000001";
74     elsif (Reg1(7 downto 4) = student_id) then
75        Result <= "00000001";
76     else
77        Result <= "00000000";
78     end if;
79
80     WHEN "0000000001000000" =>
81     if (Reg1(3 downto 0) = student_id) then
82        Result <= "00000001";
83     elsif (Reg1(7 downto 4) = student_id) then
84        Result <= "00000001";
85     else
86        Result <= "00000000";
87     end if;
88
89     WHEN "0000000010000000" =>
90     if (Reg1(3 downto 0) = student_id) then
91        Result <= "00000001";
92     elsif (Reg1(7 downto 4) = student_id) then
93        Result <= "00000001";
94     else
95        Result <= "00000000";
96     end if;
97
```

```
67      else
68         Result <= "00000000";
69      end if;
70
71      WHEN "0000000000100000" =>
72      if (Regl(3 downto 0) = student_id) then
73         Result <= "00000001";
74      elsif (Regl(7 downto 4) = student_id) then
75         Result <= "00000001";
76      else
77         Result <= "00000000";
78      end if;
79
80      WHEN "0000000001000000" =>
81      if (Regl(3 downto 0) = student_id) then
82         Result <= "00000001";
83      elsif (Regl(7 downto 4) = student_id) then
84         Result <= "00000001";
85      else
86         Result <= "00000000";
87      end if;
88
89      WHEN "0000000010000000" =>
90      if (Regl(3 downto 0) = student_id) then
91         Result <= "00000001";
92      elsif (Regl(7 downto 4) = student_id) then
93         Result <= "00000001";
94      else
95         Result <= "00000000";
96      end if;
97
98      WHEN "0000000100000000" =>
99      if (Regl(3 downto 0) = student_id) then
100        Result <= "00000001";
101     elsif (Regl(7 downto 4) = student_id) then
102        Result <= "00000001";
103     else
104        Result <= "00000000";
105     end if;
106
107     WHEN OTHERS =>
108     Result <= "--------";
109
110     END CASE;
111  END PROCESS;
112
113  Ans <= Result(3 downto 0);
114
115  END calculation;
```

Conclusion:

In conclusion, in lab 6 we successfully created multiple GPPs by implementing 3 different ALUs and different components created from all the labs throughout the semester like the mealy machine, 4 to 16 decoder, SSEG, and the latches. Through the waveforms, we were able to confirm that ALU1, ALU2, and ALU3 were working properly and that when connected to create each GPP for the different problem sets, they outputted the correct waveform. Each ALU consists of a mealy machine that would cycle through the different states which allowed the ALUs to perform different operations with the two inputs A and B. The importance of each component was emphasized in this lab as many problems we faced were due to small errors in a specific component.The completion of lab 6 demonstrates the growth in the semester from being new to quartus to creating our own GPP.