Department of Electrical and Computer Engineering COE328 – Digital Systems

Course Number:	COE328
Course Title:	Introduction Digital System
Semester/Year:	3/2023
Instructor:	
TA Name:	

Lab/Tutorial Report No.	

Report Title:	Design of a Simple General-Purpose Processor (GPU)

Section No.	
Group No.	
Submission Date:	12/ 3/ 23 by email
Due Date:	12/4/23

Student Name	Student ID	Signature*
Jonathan Ly	xxxx70138	

Table Of Contents:

Introduction:

The purpose of Lab 6 is to design and implement an Arithmetic and Logic Unit (ALU) using VHDL code and implement it on the FPGA board. A general processing unit (GPU) is composed of four distinct components. The Procuring Input Data, Storage Unit (Register), Control Unit and the ALU Core. The Procuring Input Data will consist of two 8-bit binary inputs called A and B which will go into the storage units. The Storage Units will be made up of two latches that will temporarily store A and B inputs that will be connected to the ALU core. The ALU core will perform the arithmetic and logical operations from the Storage Units and will output it to the Seven Segment Display Unit. The Control Unit is composed of a FSM block and 4 to 16 Decoder. The Control Unit will give instructions in the forms of Microcode, and each microcode will do a boolean operation (functions). The Control Unit will connect directly to the ALU core. All the components are connected using the same clock, so each rising edge, the components will receive the necessary inputs and instructions to obtain the outputs.

Components:

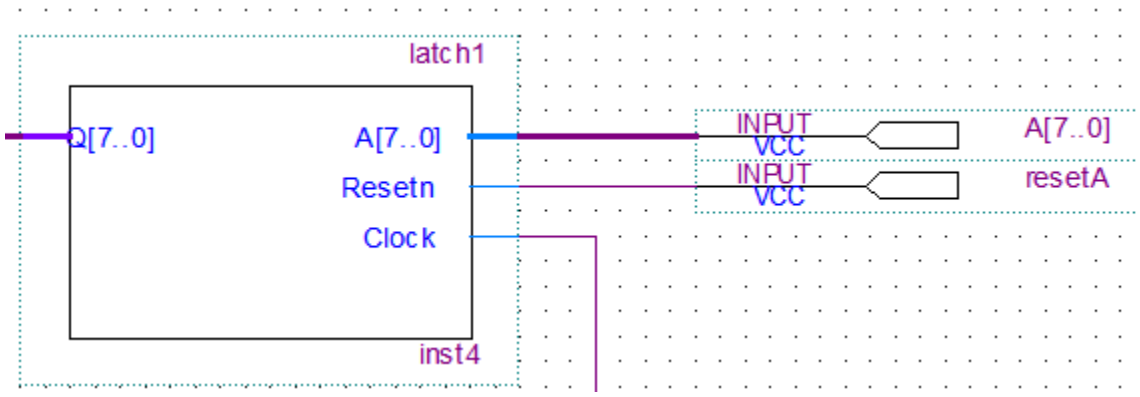
Description of Four Primary Components:

The Storage Units will consist of two Latches which temporarily store the input to be later used by the ASU core. It will receive eight bits binary input which is converted from our last four student numbers in hexadecimal. The last four digits of my student numbers are “0138”, Latch 1 will receive 8-bit binary conversion of “01” (A) in hexadecimal and Latch 2 will receive 8-bit binary conversion of “38” (B) in hexadecimal. The FSM block is an up counter sequential circuit, which consists of 9 states. When it moves on a rising edge clock, the FSM will output the current state in 4-bit binary to the 4 to 16 Decoder which contains microcode in 16-bit binary, then the ALU will receive the microcode which tells which function to compute in order.

Latch 1/ Latch 2:

Block Diagram Screenshot:

Figure 1.0: Latch 1



Description

This is the block diagram of Latch 1 which will take 8-bits for A (01 in hexadecimal to binary. Similarly, Latch 2 which will also take 8-bits for B (38 in hexadecimal to binary). Both Latches will store these inputs for the ALU core to use when each rising edge clock is cycled.

Truth Table:

Table 1.0: Latch 1 and Latch 2

Input	Reset		Input Hexadecimal	Input (8-bit Binary)	Output (Reset = 1)	Output (Reset = 0)
A	1	0	01	00000001	00000000	00000001
B	1	0	38	00111000	00000000	00111000

Description

When Reset is 1, the outputs will be 00000000. When the Reset is 0, the output will be whatever the input was.

VHDL code/ Waveform:

Figure 1.1: Latch Code

```
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 '0' 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
```

Figure 1.2: Wave form when Reset = 0

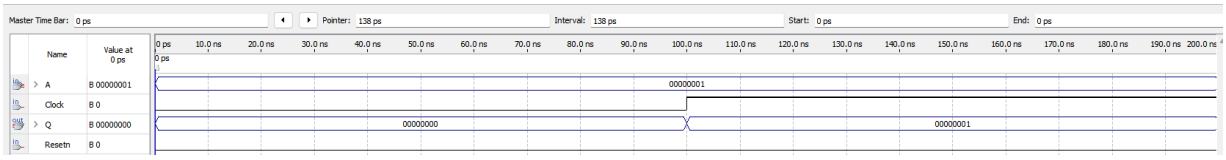
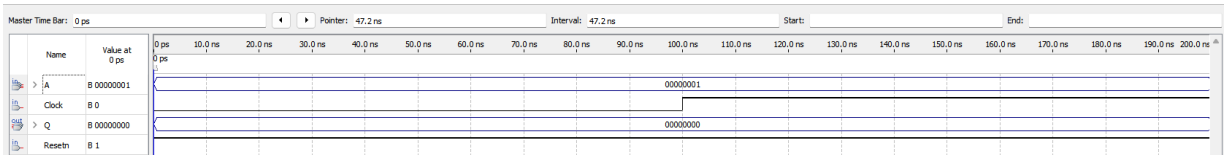


Figure 1.3: Wave form when Reset = 1



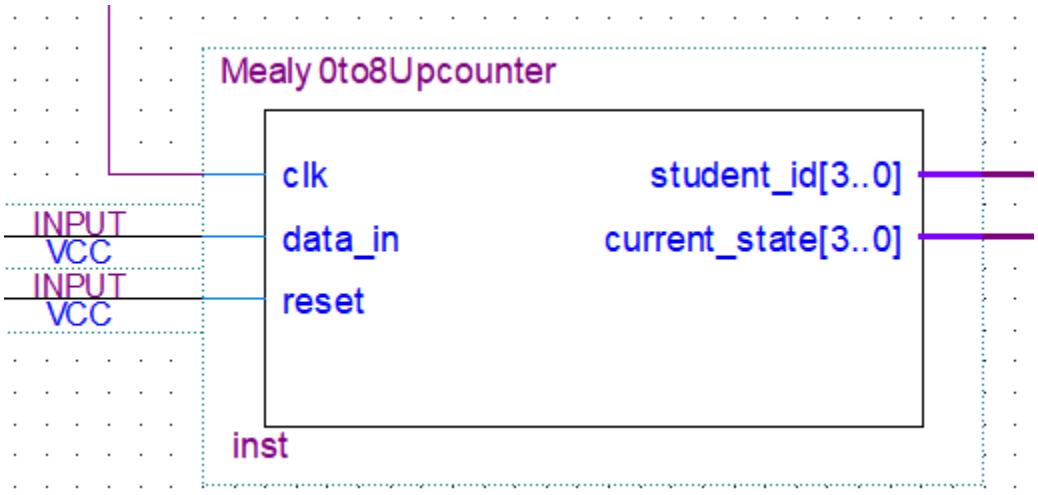
Description

The code for Latch 1 and Latch 2 are the same. Notice when Reset is 0, and clock is rising edge, the output will be whatever the input is. If Reset is 1, the output will always be zero in 8-bits.

Mealy FSM:

Block Diagram Screenshot:

Figure 2.0: Mealy FSM



Description

The mealy FSM block consists of clock, data_in and reset inputs. The output will be the student number (4-bits) at each state and current state (4-bits) which goes into the 4 to 16 decoder.

Truth Table:

Table 2.0: Mealy FSM							
Current State (Decimal)	Current State (Binary)	Student Number (Decimal)	Student Number (Binary)	Next State (Binary)		Output (Binary)	
				data_in		data_in	
				0	1	0	1
0	0000	8	1000	0000	0001	1000	0101
1	0001	5	0101	0001	0010	0101	0000
2	0010	0	0000	0010	0011	0000	0001
3	0011	1	0001	0011	0100	0001	0001
4	0100	1	0001	0100	0101	0001	0111
5	0101	7	0111	0101	0110	0111	0000
6	0110	0	0000	0110	0111	0000	0001
7	0111	1	0001	0111	1000	0001	0011
8	1000	3	0011	1000	0000	0011	1000
Description							
The first current state is my last digit of my student number, then the second state is my first student number, then it will continue onwards. For every rising edge, it will go to the next state and return the value when it enters the state for mealy FSM.							

VHDL Code/ Waveform:

Figure 2.1: FSM Mealy Code

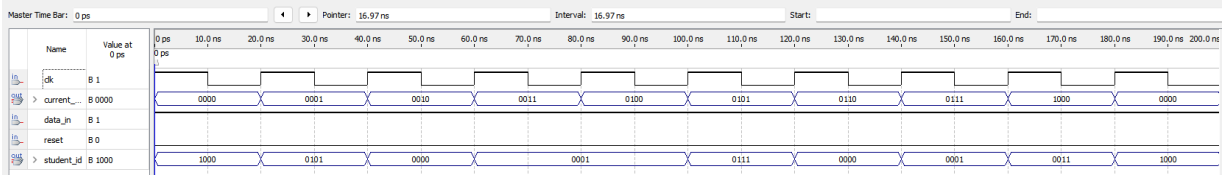
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   entity Mealy0to8Upcounter is
4       port
5       (
6           clk: in std_logic;
7           data_in: in std_logic;
8           reset: in std_logic;
9           student_id: out std_logic_vector(3 downto 0);
10          current_state: out std_logic_vector(3 downto 0)
11      );
12  end entity;
13  architecture fsm of Mealy0to8Upcounter is
14
15      type state_type is (s0,s1,s2,s3,s4,s5,s6,s7,s8);
16
17      signal yfsm: state_type;
18
19  begin
20      process (clk,reset)
21      begin
22          if reset = '1' then
23              yfsm <= s0;
24          elsif (clk'EVENT AND clk = '1') then
25
26              case yfsm is --how FSM moves (upcounter, cyling through states 0 to 8 )
27              when s0 =>
28                  if (data_in = '1') then
29                  yfsm <= s1;
30                  elsif (data_in = '0') then
31                  yfsm <= s0;
32                  end if;
33
34              when s1 =>
35                  if (data_in = '1') then
36                  yfsm <= s2;
37                  elsif (data_in = '0') then
38                  yfsm <= s1;
39                  end if;
40
41              when s2 =>
42                  if (data_in = '1') then
43                  yfsm <= s3;
44
45                  elsif (data_in = '0')
46                  then
47                  yfsm <= s2;
```

```
48              end if;
49
50          when s3 =>
51              if (data_in = '1') then
52              yfsm <= s4;
53
54              elsif (data_in = '0') then
55              yfsm <= s3;
56              end if;
57
58          when s4 =>
59              if (data_in = '1') then
60              yfsm <= s5;
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
```

```
93
94          end case;
95          end if;
96      end process;
97
98      process (yfsm, data_in)
99      begin
100        case yfsm is
101
102           when s0 => --at s0
103              current_state <= "0000"; --default current state
104              if (data_in = '1') then
105              student_id <= "1000"; --8
106              elsif (data_in = '0') then
107              student_id <= "0101"; --5
108              end if;
109
110           when s1 => --at s1
111              current_state <= "0001"; --default current state
112              if (data_in = '1') then
113              student_id <= "0101"; --5
114              elsif (data_in = '0') then
115              student_id <= "0000"; --0
116              end if;
117
118           when s2 => --at s2
119              current_state <= "0010"; --default current state
120              if (data_in = '1') then
121              student_id <= "0000"; --0
122              elsif (data_in = '0') then
123              student_id <= "0001"; --1
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"; --7
146                 elsif (data_in = '0') then
147                 student_id <= "0000"; --0
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 <= "0000"; --0
154                 elsif (data_in = '0') then
155                 student_id <= "0001"; --1
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 <= "0001"; --1
162                 elsif (data_in = '0') then
163                 student_id <= "0011"; --3
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 <= "0011"; --3
170                 elsif (data_in = '0') then
171                 student_id <= "1000"; --8
172                 end if;
173         end case;
174 
175     end process;
176 
177 end fsm;
```

Figure 2.2: Wave form when data_in = 1



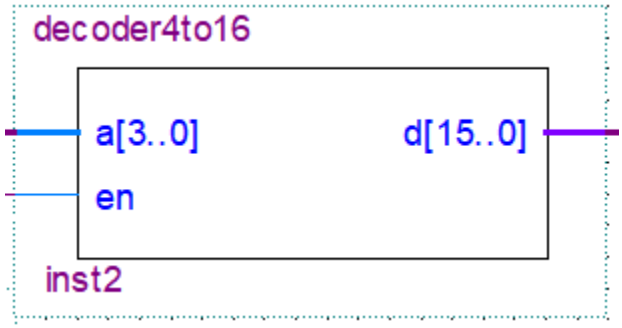
Description

The waveform for data_in = 1 will match the truth table respective at each clock signal. We will never use when data_in = 0.

4 to 16 Decoder:

Block Diagram Screenshot:

Figure 3.0: 4 to 16 Decoder



Description

The block diagram for 4 to 16 decoder takes inputs of 4-bit binary from the FSM states, the current states will correspond to a certain microcode output of 16-bit binary. The enable will turn the decoder on/ off.

Truth Table:

Table 3.0: 4 to 16 Decoder	
Enable is 1: **ON**	
Input From FSM (4-bit Binary)	Output Decoder (16-bit Binary)
S0: 0000	0000000000000001
S1: 0001	0000000000000010
S2: 0010	0000000000000100
S3: 0011	0000000000001000
S4: 0100	0000000000010000
S5: 0101	0000000000100000
S6: 0110	0000000001000000
S7: 0111	0000000010000000
S8: 1000	0000000100000000
S9: dddd	dddddddddddddddd
. . .	. . . . .
S15: dddd	dddddddddddddddd
Description	
When the decoder is on (enable is 1), the decoder will take inputs from the FSM (4-bit Binary) and the decoder will output (16-bit Binary). The 16 bit binary will be the microcode instruction for each function to run in the ALU core. We will not care about anything after state 9 to state 15 since we only have 9 functions for our GPU.	

VHDL Code/ Waveform:

Figure 3.1: 4 to 16 Decoder Code

```
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;
```

Figure 3.2: Wave form when Enable = 1

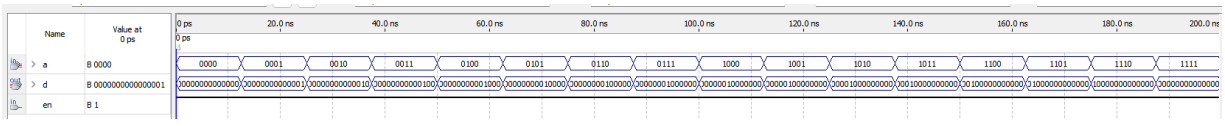
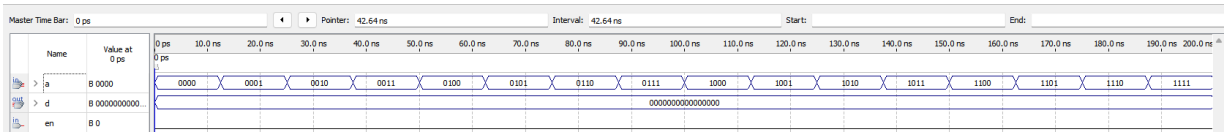


Figure 3.3: Wave form when Enable = 0

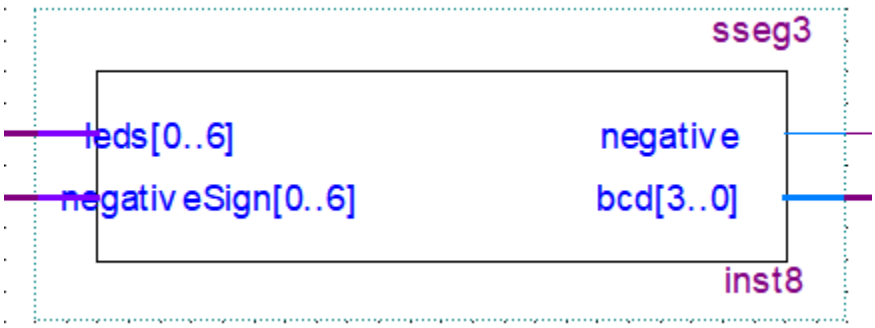


Description

The code for Latch 1 and Latch 2 are the same. Notice when Reset is 0, and clock is rising edge, the output will be whatever the input is. If Reset is 1, the output will always be zero in 8-bits.

Seven Segment Display for GPU3 Only:

Block Diagram Screenshot:

Figure 4.0: SSEG3 for Problem Set 3

Description
The SSEG3 modified for problem 3 only takes in 4-bits from ALU. Then it will output Y or N in SSEG

Truth Table:

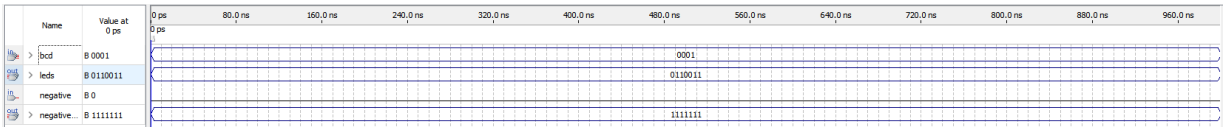
Table 4.0: SSEG3		
Input	**Output (abcdefg)**	**Y/N**
0000	1110110	N
0001	0110011	Y
Description		
The modified SSEG3 only takes in 4-bit binary 0000 or 0001, 0000 will output a N in Seven Segment Code and 0001 will output a Y in Seven Segment Code.		

VHDL Code/ Waveform:

Figure 4.1: SSEG3 Code

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY sseg3 IS
5       PORT (negative : IN STD_LOGIC;
6             bcd : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
7             leds, negativeSign: OUT STD_LOGIC_VECTOR(0 TO 6));
8   END sseg3;
9
10  ARCHITECTURE Behavior OF sseg3 IS
11  BEGIN
12      PROCESS (bcd)
13      BEGIN
14          if negative = '1' then
15              negativeSign <= "1111110";
16          else
17              negativeSign<= "1111111";
18
19          END if;
20
21          CASE bcd IS -- abcdefg
22              WHEN "0000" => leds <= "1110110";   -- n
23              WHEN "0001" => leds <= "0110011";   -- y
24              WHEN OTHERS => leds <= "-------";
25          END CASE;
26      END PROCESS;
27  END Behavior;
```

Figure 4.2: Waveform of SSEG3



Description

The code and waveforms will show Y or N depending on the input (output from ASU3). The negative component isn't connected and will not matter.

Seven Segment Display for GPU1, GPU2 & GPU3:

Block Diagram Screenshot:

Figure 5.0: Seven Segment Display

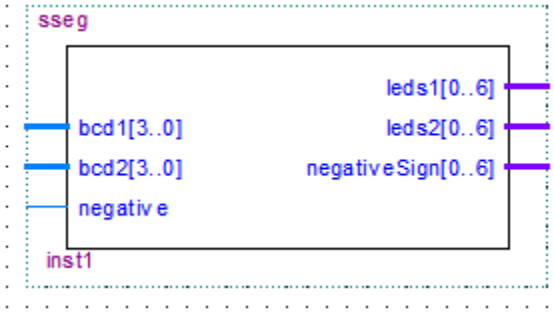
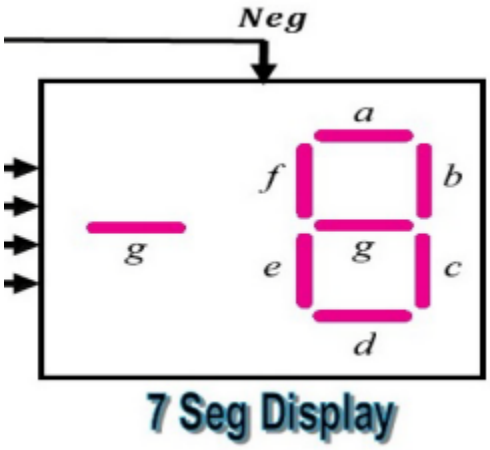


Figure 5.1: Layout of SSEG



Description

The Seven Segment Display Block consists of two 4-bit binary inputs, where bcd1 takes in R1 (4-bit Least Significant Bit of result) and bcd2 takes in R2 (4-bit Most Significant Bit of result). The output with leds1 will display the hexadecimal of R1 in seven segment code. Similarly the output of leds2 will display the hexadecimal of R2 in seven segment code. The output called negativeSign will display in seven segment code as well. The seven segment code will be in the form “abcdef”. Note if your board is low-active, 0 is when the segments are on. If your board is high-active, 1 is when the segments are on. When a negative signal is given, the SSEG will display the g-segment which is a negative sign. Note that this was used for GPU1, GPU2 and GPU3.

Truth Table:

Table 5.0: Seven Segments for high-active		
Negative (abcdefg)		
0000001		
Inputs (4-bit Binary)	**Outputs (abcdefg: “1” means on)**	**Hexadecimal value**
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
Description		
Since our board was high-active, the 1 represents the segment being lit when it's on. The SSEG will display in hexadecimal. It takes in 4-bit binary input and it will display the values in hexadecimal on the SSEG. If the result is a negative number, it will display only the g-segment for a negative sign.		

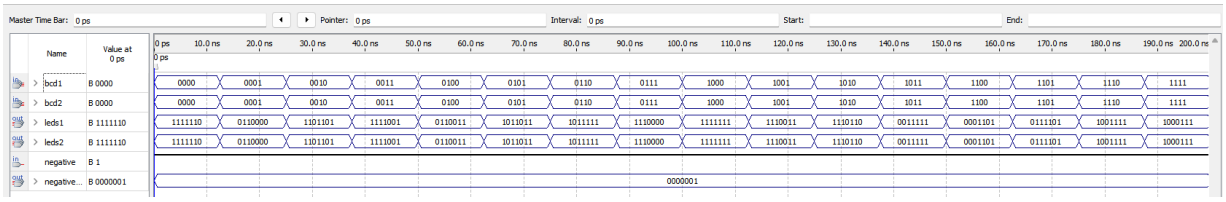
VHDL Code/ Waveform:

Figure 5.2: Seven Segment Display Code

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

ENTITY sseg IS
    PORT (
          negative : IN STD_LOGIC;
          bcd1 , bcd2 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          leds1, leds2, negativeSign: OUT STD_LOGIC_VECTOR(0 TO 6)
             );
END sseg;

ARCHITECTURE Behavior OF sseg IS
BEGIN
    PROCESS (negative)
    BEGIN
        IF negative = '1' THEN
          negativeSign <=  "0000001";

          ELSE
          negativeSign <= "0000000";

          END IF;
   END PROCESS;

   PROCESS (bcd1)
      BEGIN
            CASE bcd1 IS -- abcdefg
            WHEN "0000" => leds1 <=  "1111110";
            WHEN "0001" => leds1 <=  "0110000";
            WHEN "0010" => leds1 <=  "1101101";
            WHEN "0011" => leds1 <=  "1111001";
            WHEN "0100" => leds1 <=  "0110011";
            WHEN "0101" => leds1 <=  "1011011";
            WHEN "0110" => leds1 <=  "1011111";
            WHEN "0111" => leds1 <=  "1110000";
            WHEN "1000" => leds1 <=  "1111111";
            WHEN "1001" => leds1 <=  "1110011";
            WHEN "1010" => leds1 <=  "1110110";
            WHEN "1011" => leds1 <=  "0011111";
            WHEN "1100" => leds1 <=  "0001101";
            WHEN "1101" => leds1 <=  "0111101";
            WHEN "1110" => leds1 <=  "1001111";
            WHEN "1111" => leds1 <=  "1000111";
            WHEN OTHERS => leds1 <=  "1001111";
        END CASE;
    END PROCESS;
```

```
47
48      PROCESS (bcd2)
49        BEGIN
50              CASE bcd2 IS -- abcdefg
51              WHEN "0000" => leds2 <=  "1111110";
52              WHEN "0001" => leds2 <=  "0110000";
53              WHEN "0010" => leds2 <=  "1101101";
54              WHEN "0011" => leds2 <=  "1111001";
55              WHEN "0100" => leds2 <=  "0110011";
56              WHEN "0101" => leds2 <=  "1011011";
57              WHEN "0110" => leds2 <=  "1011111";
58              WHEN "0111" => leds2 <=  "1110000";
59              WHEN "1000" => leds2 <=  "1111111";
60              WHEN "1001" => leds2 <=  "1110011";
61              WHEN "1010" => leds2 <=  "1110110";
62              WHEN "1011" => leds2 <=  "0011111";
63              WHEN "1100" => leds2 <=  "0001101";
64              WHEN "1101" => leds2 <=  "0111101";
65              WHEN "1110" => leds2 <=  "1001111";
66              WHEN "1111" => leds2 <=  "1000111";
67              WHEN OTHERS => leds2 <=  "1001111";
68          END CASE;
69      END PROCESS;
70
71  END Behavior;
```

Figure 5.3: Waveform for Negative numbers



Description

The code is for a high-active SSEG board, where 1 in the SSEG is when the segment lights are on. When negative is 1, the negative SSEG will give g segment as 1. Which means the number in hexadecimal is negative.

ALU1 Problem Set 1:

Description:

The purpose of ALU1 is to compute the basic arithmetic and logical functions of 8-bit input A and 8-bit input B in binary from the Storage Unit which consists of two Latches. The Control Unit will give the microcode each time a new rising edge clock occurs. The microcode acts like an instruction to tell the ALU1 when to compute. The ALU will handle the computation when each microcode is called. The result of the computation is in 8-bit, but the SSEG only takes in 4-bit. The result will be broken down into R2 being the first four most significant bits and R1 being the last four Least Significant bits. There will be 3 SSEG which displays in hexadecimal format where the first one will display R2, the second will display R1 and finally the third will display the negative sign if the computation is a negative number (function 3 will have a negative output). Each microcode will have a different function associated (refer to Table 6.0). Each component of the GPU1 will be connected to the same clock. The output should be going along the FSM up counter which will have nine states for nine functions. After the 9 rising edge clock, it will start from the beginning and count up once more. When the reset of each component is pressed, it should go back to the beginning. Below I have attached the Block Schematic BDF Screenshot, the truth table of the expected output for each computation/ microcode instructions, the VHDL files, and finally the Waveforms.

Block Schematic BDF Screenshot:

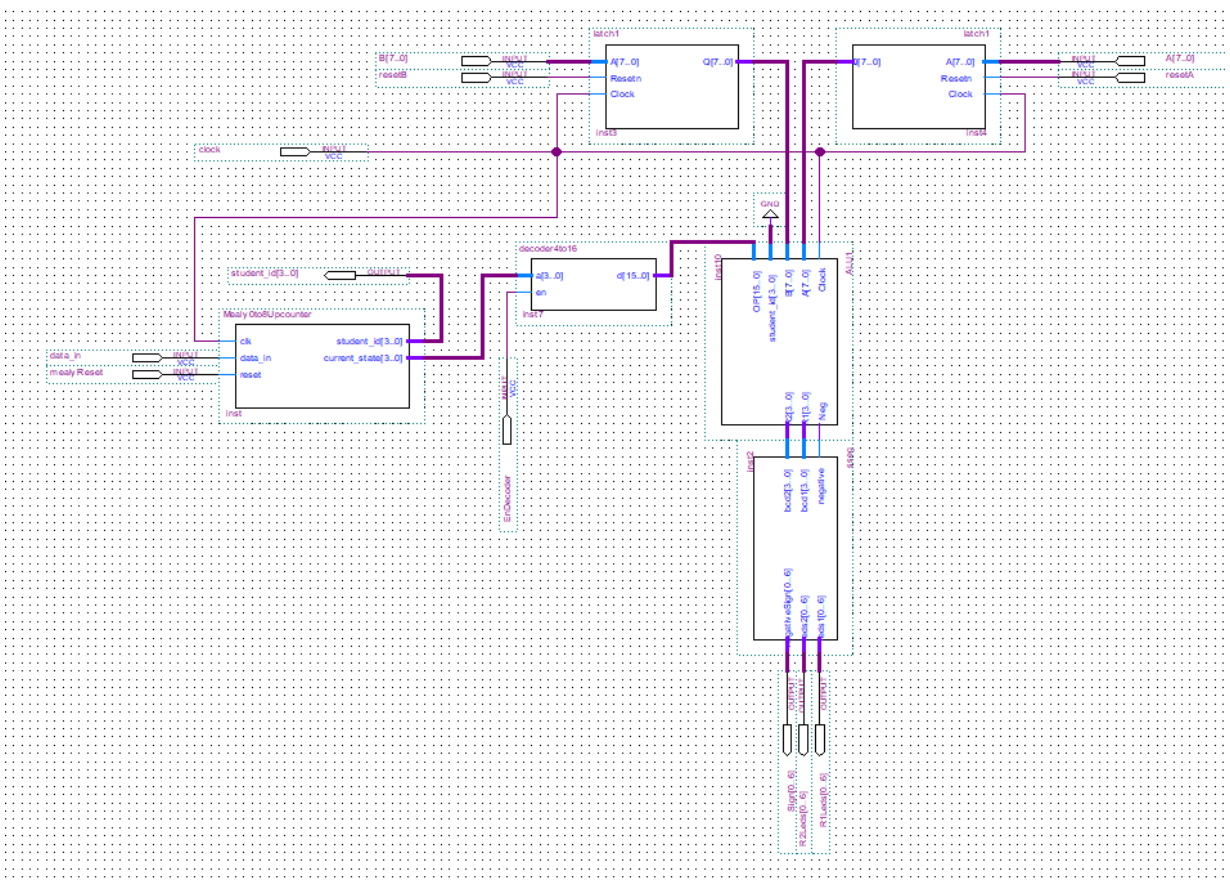
Figure 6.0: GPU1/ ALU1

Description
BDF of ALU1 for problem 1.

Table of decoder's Microcode & Result:

Table 6.0 Microcode for GPU1		
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	$\overline{A \oplus B}$
Description		
This table shows the microcode operation for GPU1 in ALU1 core. Each microcode corresponds to a specific function of arithmetic and logical computation.		

Table 6.1: Truth Table for Problem Set 1						
Input of Latch1 (A)			**Input of Latch2 (B)**			
Hexadecimal	01		Hexadecimal	38		
Binary	00000001		Binary	00111000		
ASU1		**Result in Binary**		**abcdefg**		
Function #	Operation	Result 2	Result 1	SSEG2	SSEG1	Sign SSEG
1	Sum(A,B)	0011	1001	3: 1111001	9: 1110011	0000000
2	Diff(A,B)	0011	0111	3: 1111001	7: 1110000	0000001
3	$\overline{A}$	1111	1110	F: 1000111	E: 1001111	0000000
4	$\overline{A \bullet B}$	1111	1111	F: 1000111	F: 1000111	0000000
5	$\overline{A + B}$	1100	0110	C: 0001101	6: 1011111	0000000
6	$A \bullet B$	0000	0000	0: 1111110	0: 1111110	0000000
7	$A \oplus B$	0011	1001	3: 1111001	9: 1110011	0000000
8	$A + B$	0011	1001	3: 1111001	9: 1110011	0000000
9	$\overline{A \oplus B}$	1100	0110	C: 0001101	6: 1011111	0000000
Description						
This truth table shows the expected output from the SSEG in hexadecimal for each microcode function. The GPU will have output of Result2 (Four Most Significant Bit) and Result1 (Four Least Significant Bit).						

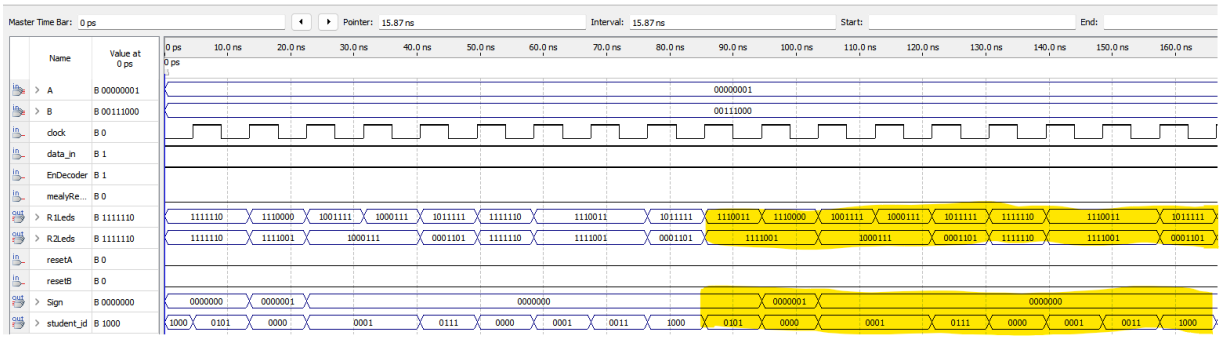
VHDL Code/ Waveform:

6.1: Code for ALU1

```
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" =>
      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';

```

```
45
46          WHEN "0000000000001000"=>
47          -- Do Boolean NAND
48          Result <= NOT(Reg1 AND Reg2);
49          Neg <= '0';
50
51          WHEN "0000000000010000"=>
52          -- Do Boolean NOR
53          Result <= NOT(Reg1 OR Reg2);
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 OR
63          Result <= Reg1 OR Reg2;
64          Neg <= '0';
65
66          WHEN "0000000010000000"=>
67          -- Do Boolean XOR
68          Result <= Reg1 XOR 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;
```

Figure 6.2: Waveform of GPU1



Description

Note that the output is correct in the yellow highlighted region. Sign, R2Leds, R1Leds outputs are in terms of SSEG code where 1 means lit on SSEG and 0 means off in SSEG. Recall that the format for SSEG is abcdef, which corresponds to the Seven Segments on/off in SSEG. student_id is in the binary of student number. The first function should be

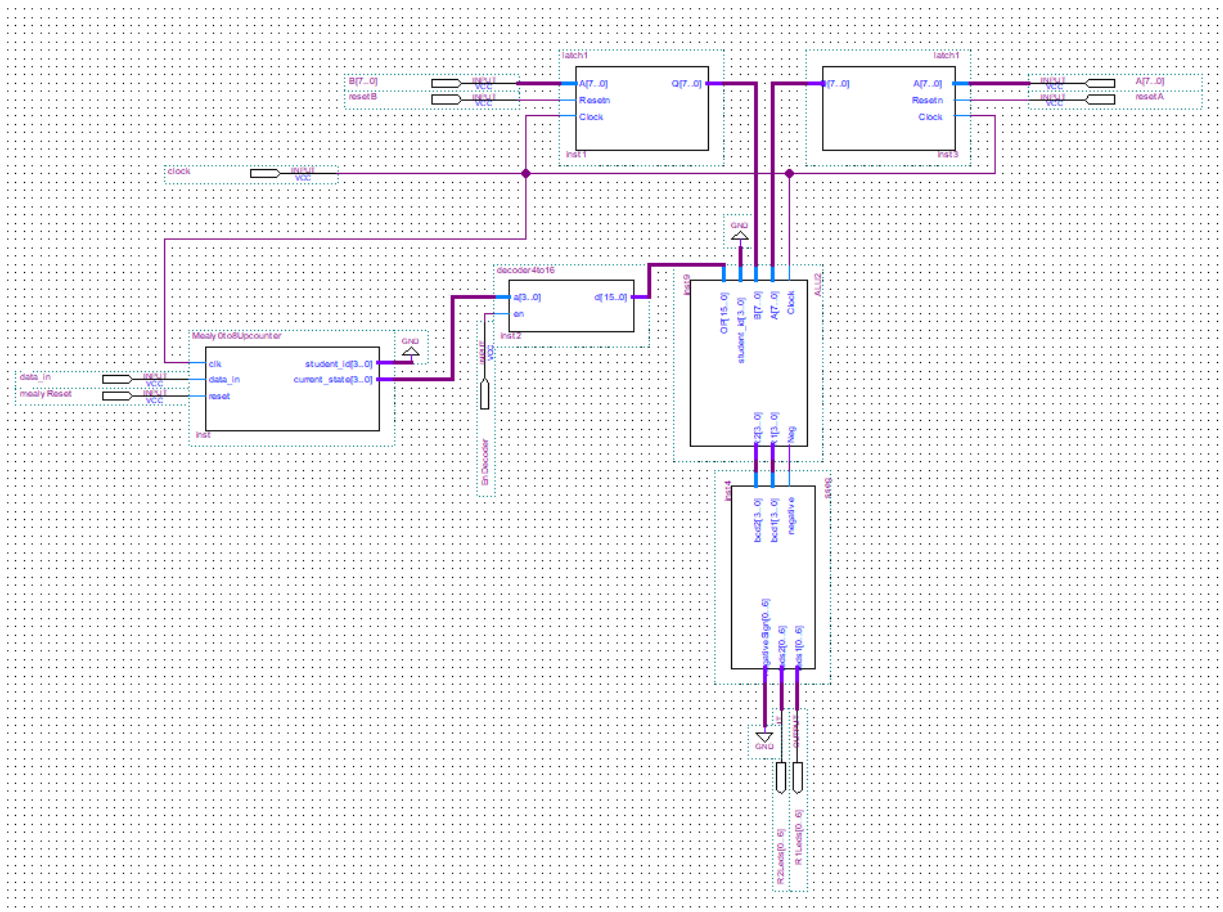
ALU2 Problem Set 2:

Description:

The workings of ALU2 is very similar to ALU1 which was used to compute the basic arithmetic and logical functions of 8-bit input A and 8-bit input B in binary from the Storage Unit which consists of two Latches. Just for GPU1, the Control Unit will also give the microcode each time a new rising edge clock occurs. The microcode acts like an instruction to tell the ALU2 when to compute. The ALU2 will handle the computation when each microcode is called. The result of the computation is in 8-bit, but the SSEG only takes in 4-bit. The result will be broken down into R2 being the first four most significant bits and R1 being the last four Least Significant bits. There will be 2 SSEG which display in hexadecimal format where the first one will display R2, the second will display R1. Each microcode will have a different function associated (refer to Table 7.0). Each component of the GPU2 will be connected to the same clock. The output should be going along the FSM up counter which will have nine states for nine functions. After the 9 rising edge clock, it will start from the beginning and count up once more. When the reset of each component is pressed, it should go back to the beginning. Below I have attached the Block Schematic BDF Screenshot, the truth table of the expected output for each computation/ microcode instructions, the VHDL files, and finally the Waveforms. Note that we did part g for the problem set.

Block Schematic BDF Screenshot:

Figure 7.0: GPU2/ ALU2



Description

BDF of ALU2 for problem 2.

Table of decoder's Microcode & Result:

Table 7.0: Microcode for ALU2 (g)		
Function	**Microcode (From Decoder)**	**Operation**
1	0000000000000001	Invert Significant-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	Print A when Even-bits of A becomes Even-bits of B
8	0000000010000000	$\overline{A \oplus B}$
9	0000000100000000	ROR B by 3 bits
Description		
This table shows the microcode operation for GPU2 in ALU2 core. Each microcode corresponds to a specific function of arithmetic and logical computation.		

Table 7.1: Table Truth Table for Problem Set 2 (g)					
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 Significant-bit A	1000	0000	8: 1111111	0: 1111110
2	SHL A by 4, all 1 after	0001	1111	1: 0110000	F: 1000111
3	Invert Upper 4-bit B	1100	1000	C: 0001101	8: 1111111
4	Min(A, B)	0000	0001	0: 1111110	1: 0110000
5	Sum(A,B) add 4	0010	1011	3: 1111001	D: 0111101
6	A add 3	0000	0100	0: 1111110	4: 0110011
7	Print A when Even-bits of A becomes Even-bits of B	0001	0000	1: 0110000	0: 1111110
8	$\overline{A \oplus B}$	1100	0110	C: 0001101	6: 1011111
9	ROR B by 3 bits	0000	0111	0: 1111110	7: 1110000
Description					
This truth table shows the expected output from the SSEG in hexadecimal for each microcode function. The GPU will have output of Result2 (Four Most Significant Bit) and Result1 (Four Least Significant Bit).					

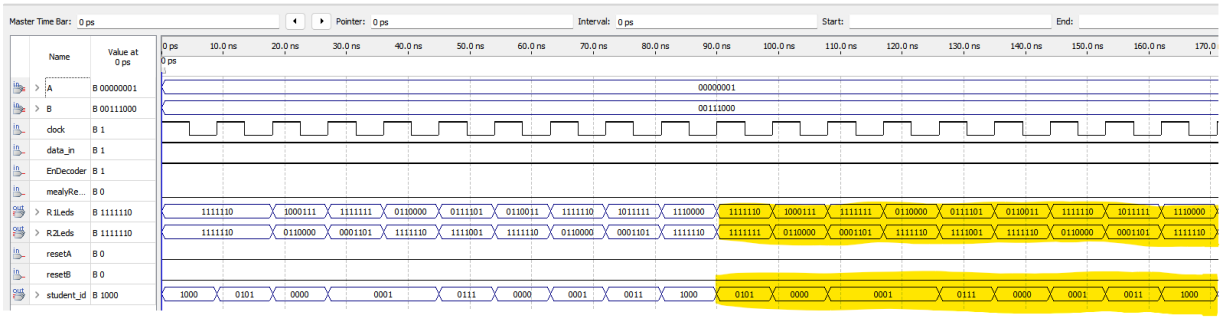
VHDL Code/ Waveform:

Figure 7.1: Code for ALU2

```
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 ALU2 is
6   port( Clock : in std_logic; -- input clock signal
7         A,B : in unsigned(7 downto 0); -- 8-bit inputs from latches A and B
8         student_id : in unsigned(3 downto 0); -- 4 bit student id from FSM
9         OP : in unsigned(15 downto 0); -- 16-bit selector for Operation from Decoder
10        Neg: out std_logic; -- is the result negative ? Set-ve bit output
11        R1 : out unsigned(3 downto 0); -- lower 4-bits of 8-bit Result Output
12        R2 : out unsigned(3 downto 0)); -- higher 4-bits of 8-bit Result Output
13  end ALU2;
14  architecture calculation of ALU2 is -- temporary signal declarations.
15  signal Reg1,Reg2,Result : unsigned(7 downto 0) := (others => '0');
16  signal Reg4 : unsigned (0 to 7);
17  begin
18  Reg1 <= A; -- temporarily store A in Reg1 local variable
19  Reg2 <= B; -- temporarily store B in Reg2 local variable
20  process(Clock, OP)
21  begin
22     if(rising_edge(Clock)) THEN -- Do the calculation @ positive edge of clock cycle.
23        case OP is
24              WHEN "0000000000000001" =>
25                    Result(0)<= Reg1(7);
26                    Result(1)<= Reg1(6);
27                    Result(2)<= Reg1(5);
28                    Result(3)<= Reg1(4);
29                    Result(4)<= Reg1(3);
30                    Result(5)<= Reg1(2);
31                    Result(6)<= Reg1(1);
32                    Result(7)<= Reg1(0);
33                    --Neg<='0';
34
35
36              WHEN "0000000000000010" =>
37
38                     Result<= Reg1 sll 4; --note was 3
39                     Result(0)<='1';
40                     Result(1)<='1';
41                     Result(2)<='1';
42                     Result(3)<='1';
```

```
44              WHEN "0000000000000100" =>
45                    Result(7)<= Not Reg2(7);
46                    Result(6)<= Not Reg2(6);
47                    Result(5)<= Not Reg2(5);
48                    Result(4)<= Not Reg2(4);
49                    Result(3)<=  Reg2(3);
50                    Result(2)<=  Reg2(2);
51                    Result(1)<=  Reg2(1);
52                    Result(0)<=  Reg2(0);
53
54
55              WHEN "0000000000001000" =>
56                     if(Reg1<=Reg2) then
57                         Result<= Reg1;
58                     else
59                         Result<= Reg2;
60                     end if;
61
62              WHEN "0000000000010000" => Result <= Reg1 + Reg2+ "00000100"; --note before was 4
63
64              WHEN "0000000000100000" => Result<= Reg1 + "00000011";
65
66              WHEN "0000000001000000" =>
67
68                    Result(0) <= Reg2(0);
69                    Result(1) <= Reg1(1);
70                    Result(2) <= Reg2(2);
71                    Result(3) <= Reg1(3);
72                    Result(4) <= Reg2(4);
73                    Result(5) <= Reg1(5);
74                    Result(6) <= Reg2(6);
75                    Result(7) <= Reg1(7);
76
77
78              WHEN "0000000010000000" =>  Result <= (Reg1 XNOR Reg2);
79
80              WHEN "0000000100000000" =>  Result<= Reg2 ROR 3;
81
82              WHEN OTHERS =>
83                    Result<= "--------";
84
85
86            end case;
87        end if;
88    end process;
89  R1 <= Result(3 downto 0); -- Since the output seven segments can
90  R2 <= Result(7 downto 4); -- only 4-bits, split the 8-bit to two 4-bits.
```

Figure 7.2: Waveform of GPU2



Description

Note that the output is correct in the yellow highlighted region. Sign, R2Leds, R1Leds outputs

are in terms of SSEG code where 1 means lit on SSEG and 0 means off in SSEG. Recall that the format for SSEG is abcdef, which corresponds to the Seven Segments on/off in SSEG. student_id is in the binary of student number.

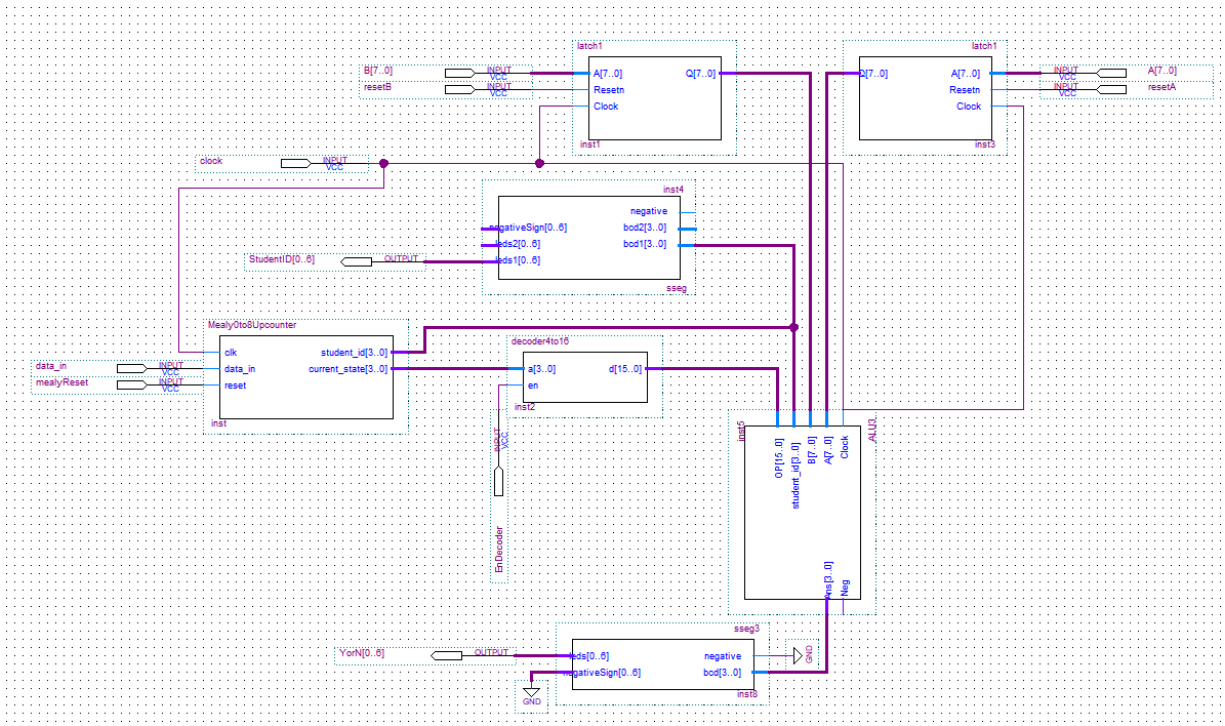
ALU3 Problem Set 3:

Description:

In problem set 3, the GPU will still contain the 4 primary components. The main difference of ALU3 is that the microcode functions are different. The methods of retrieving inputs are still the same (retrieved from the Storage Unit) and retrieving the microcontroller is also the same (retrieved from the Control Unit). The output of ALU3 will also be in 4-bits but it will be acquired from the least significant bit of 00000000 when the comparison is false and 00000001 when comparison is true which depends on the function detailed in Table 8.0. The ALU will only output the 4-bits which SSEG3 will output in the display either a Y or N in SSEG code (refer to Table 8.1). The details of functionalities and outputs of the GPU3 will be shown below. Note that we did part g of the problem set.

Block Schematic BDF Screenshot:

Figure 8.0: GPU3/ ALU3



Description

BDF of ALU3 for problem 3.

Table of decoder's Microcode & Result:

Table 8.0: Microcode of ASU3 (g)		
Function	**Microcode (From Decoder)**	**Operation**
1	0000000000000001	Compare the 1st of student number with both bits of A in hexadecimal
2	0000000000000010	Compare the 2nd of student number with both bits of A in hexadecimal
3	0000000000000100	Compare the 3rd number of student number with both bits of A in hexadecimal
4	0000000000001000	Compare the 4th number of student number with both bits of A in hexadecimal
5	0000000000010000	Compare the 5th number of student number with both bits of A in hexadecimal
6	0000000000100000	Compare the 6th number of student number with both bits of A in hexadecimal
7	0000000001000000	Compare the 7th number of student number with both bits of A in hexadecimal
8	0000000010000000	Compare the 8th number of student number with both bits of A in hexadecimal
9	0000000100000000	Compare the 9th number of student number with both bits of A in hexadecimal
Description		
This table shows the microcode operation for GPU3 in ALU3 core. Each microcode corresponds to a specific function of arithmetic and logical computation.		

Table: 8.1: Truth Table for Problem Set 3 (g)					
Input of Latch1 (A) ONLY					
Hexadecimal		01			
Binary		00000001			
ASU3		**A in Binary**		**SSEG**	
Function #	Student ID	First 4-bits	Last 4-bits	abcdefg	Y or N
1	5: 0101	0000	0001	1110110	N
2	0: 0000	0000	0001	0110011	Y
3	1: 0001	0000	0001	0110011	Y
4	1: 0001	0000	0001	0110011	Y
5	7: 0111	0000	0001	1110110	N
6	0: 0000	0000	0001	0110011	Y
7	1: 0001	0000	0001	0110011	Y
8	3: 0011	0000	0001	1110110	N
9	8: 1000	0000	0001	1110110	N
Description					
When the ALU3 checks for current state, it will check for the first 4 bits of A (MSB) and check for the last 4 bits of A (LSB). If one of them matches the 4 bit student number, it will output 0000 or 0001 from ALU3. The SSEG3 will take in 0000 or 0001 and give Y for 0001 or N for 0000.					

VHDL Code/ Waveform:

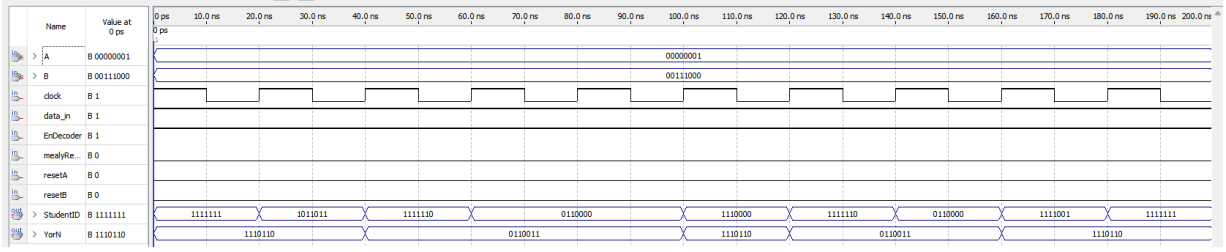
Figure 8.1: Code for ALU2

```
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, B: 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  Reg2 <= B;
22
23  PROCESS(Clock, OP)
24  BEGIN
25  CASE OP IS
26     WHEN "0000000000000001" =>
27     if (Reg1(3 downto 0) = student_id) then
28        Result <= "00000001";
29     elsif (Reg1(7 downto 4) = student_id) then
30        Result <= "00000001";
31     else
32        Result <= "00000000";
33     end if;
34
35     WHEN "0000000000000010" =>
36     if (Reg1(3 downto 0) = student_id) then
37        Result <= "00000001";
38     elsif (Reg1(7 downto 4) = student_id) then
39        Result <= "00000001";
40     else
41        Result <= "00000000";
42     end if;
43
44
45     WHEN "0000000000000100" =>
46     if (Reg1(3 downto 0) = student_id) then
```

```
44      
45          WHEN "0000000000000100" =>
46          if (Regl(3 downto 0) = student_id) then
47             Result <= "00000001";
48          elsif (Regl(7 downto 4) = student_id) then
49             Result <= "00000001";
50          else
51             Result <= "00000000";
52          end if;
53      
54          WHEN "0000000000001000" =>
55          if (Regl(3 downto 0) = student_id) then
56             Result <= "00000001";
57          elsif (Regl(7 downto 4) = student_id) then
58             Result <= "00000001";
59          else
60             Result <= "00000000";
61          end if;
62      
63          WHEN "0000000000010000" =>
64          if (Regl(3 downto 0) = student_id) then
65             Result <= "00000001";
66          elsif (Regl(7 downto 4) = student_id) then
67             Result <= "00000001";
68          else
69             Result <= "00000000";
70          end if;
71      
72          WHEN "0000000000100000" =>
73          if (Regl(3 downto 0) = student_id) then
74             Result <= "00000001";
75          elsif (Regl(7 downto 4) = student_id) then
76             Result <= "00000001";
77          else
78             Result <= "00000000";
79          end if;
```

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

Figure 8.2: Waveform of GPU3



Description

Since our first mealy state would be the student number of 8, it will print 8 in hexadecimal in the SSEG.

Conclusion:

Overall, in Lab 6 we were able to design and program in VHDL of a General-Purpose Processor Unit (GPU). We are able to finalize what we learn throughout the semester to compute arithmetic and logical operations in the ALU. I understood the importance of the 4 components which make up the GPU. Without them, we would not be able to perform the operation. The GPU can perform multiple tasks from only 2 inputs from the user once. Without the Storage unit, which acts as a place to store the inputs (which acts as the memory for the ALU), the ALU will not be able to access them again and again after completing one function after another. The control unit was also very important since it gave the instructions for the ALU when to conduct the operation. The importance of the clock signal to be synchronized in each component is very crucial since without it, the ALU and its main components will not be able to move from one function to another. With the basic understanding of how the GPU works, I can imagine the possibilities that logical systems can do to perform certain tasks to improve the efficiency in day to day tasks.