Toronto Metropolitan University

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

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

Instructor:	Reza Sedaghat
TA Name:	Ammad Shah

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

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

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Sin	Darren	501167165	8	

*By signing above you attest that you have contributed to this written lab report and confirm that all work you have contributed to this lab report is your own work. Any suspicion of copying or plagiarism in this work will result in an investigation of Academic Misconduct and may result in a “0” on the work, an “F” in the course, or possibly more severe penalties, as well as a Disciplinary Notice on your academic record under the Student Code of Academic Conduct, which can be found online at: https://www.torontomu.ca/senate/policies/pol60.pdf

Table of Contents

Introduction

The *Design of a Simple General Purpose Processor* lab is the culmination of all the concepts covered throughout this course. The main objective of this experiment is to design and implement an Arithmetic Logic Unit (ALU) using Quartus 13.0 for the purpose of performing a plethora of operations. These operations range from arithmetic to logical operations. Additionally, there are several other components that will be designed and implemented to assist the ALU. These components include Registers, which behave as storage units; and the 4:16 Decoder and Finite State Machine (FSM), which together make up the control unit. These components will cooperate together to form inputs and provide the ALU with data to process. The results will then be transmitted through Seven Segment Displays (SSEG) and displayed onto an FPGA board. All together, these elements are what make up the General Purpose Processor.

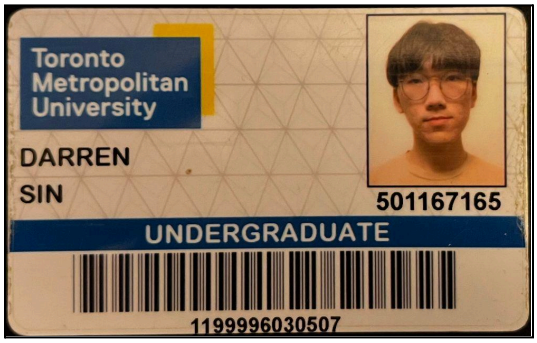

Figure 1. Student ID Card

Components

Latch 1

Latch 1 is a register designed for the purpose of temporarily storing input A. It consists of an input 8-bit vector, a reset switch, and an output 8-bit vector. It works by running on an alternating clock signal that takes in an 8-bit input and outputs it on a high signal. Specifically for this lab, this 8-bit input will be the binary conversion of the first two of the last four digits of the student number (i.e. $A = (71)_{16} = (0111\ 0001)_2$). The generated output will be transmitted to the ALU component.

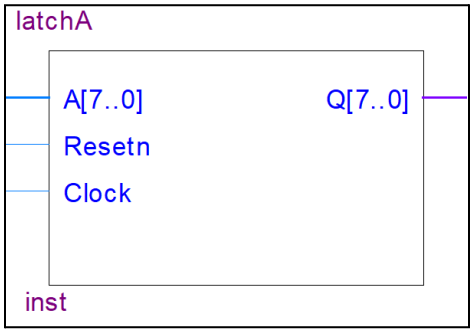

Figure 2. The Block Schematic for Latch 1

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY latchA IS
5      PORT(A:IN STD_LOGIC_VECTOR(7 DOWNTO 0);--8 bit A 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 latchA;
9
10  ARCHITECTURE Behavior OF latchA IS
11  BEGIN
12     PROCESS(Resetn, Clock)--Process takes reset and clock as inputs
13     BEGIN
14        IF Resetn = '0' THEN--when reset input 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 Behavior;
```

Figure 3. The VHDL Code for Latch 1

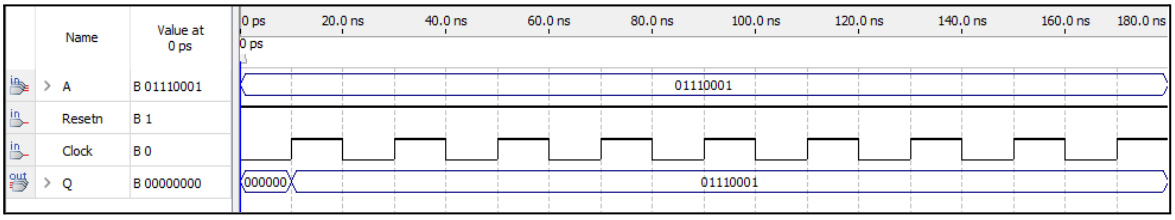

Figure 4. The Waveform for Latch 1 (Input A)

Latch 2

Latch 2 is modelled in the same way as Latch 1; however, it will be used to temporarily store input B. This input will be the binary conversion of the last two of the last four digits of the student number (i.e. $B = (65)_{16} = (0110\ 0101)_2$). Again, the generated output will be carried into the ALU component.

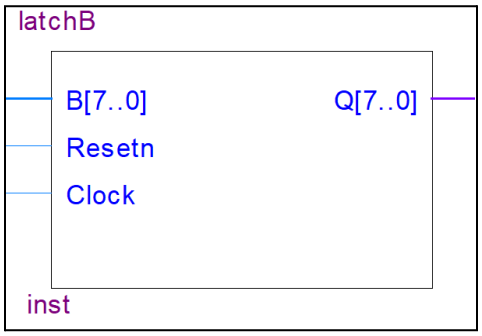

Figure 5. The Block Schematic for Latch 2

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY latchB IS
5      PORT(B:IN STD_LOGIC_VECTOR(7 DOWNTO 0);--8 bit A 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 latchB;
9
10  ARCHITECTURE Behavior OF latchB IS
11  BEGIN
12     PROCESS(Resetn, Clock)--Process takes reset and clock as inputs
13     BEGIN
14        IF Resetn = '0' THEN--when reset input 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 <= B;
18        END IF;
19     END PROCESS;
20  END Behavior;
```

Figure 6. The VHDL Code for Latch 2

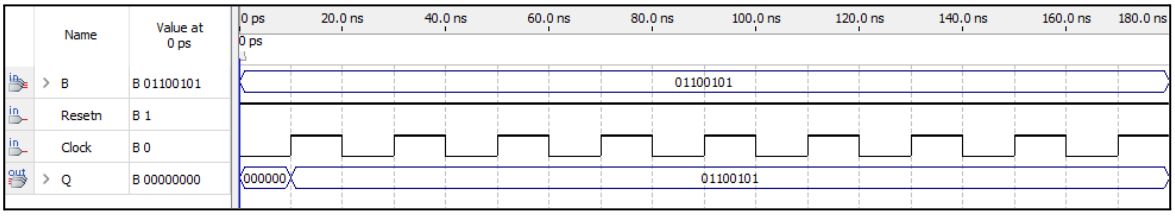

Figure 7. The Waveform for Latch (Input B)

4:16 Decoder

The 4:16 Decoder is one of the two components that make up the control unit. The purpose of it is to take in a 4-bit input and process it to produce a 16-bit output. Specifically, for the purpose of this lab, the 4:16 Decoder will take the 4-bit value of the current state from the FSM and generate microcode which will then be passed to the ALU. This microcode is crucial in determining which operation is executed in the ALU.

W0	W1	W2	W3	Y15	Y14	Y13	Y12	Y11	Y10	Y9	Y8	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 1. The Truth Table for the 4:16 Decoder

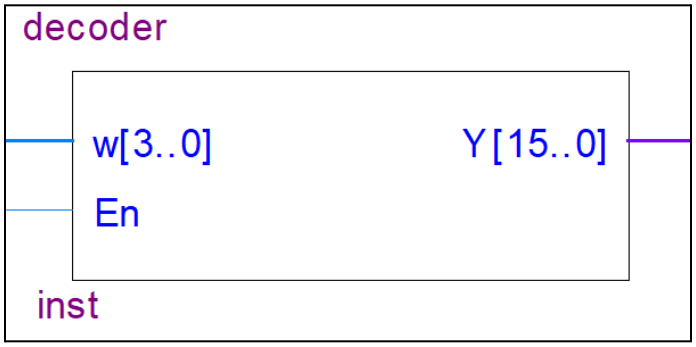

Figure 8. The Block Schematic for the 4:16 Decoder

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3
4   ENTITY decoder 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 decoder ;
10
11  ARCHITECTURE Behavior OF decoder IS
12     SIGNAL Enw : STD_LOGIC_VECTOR(4 DOWNTO 0) ;
13  BEGIN
14     Enw <= En & w ;
15     WITH Enw SELECT
16     Y <=     "1000000000000000" WHEN "10000",
17              "0100000000000000" WHEN "10001",
18              "0010000000000000" WHEN "10010",
19              "0001000000000000" WHEN "10011",
20              "0000100000000000" WHEN "10100",
21              "0000010000000000" WHEN "10101",
22              "0000001000000000" WHEN "10110",
23              "0000000100000000" WHEN "10111",
24              "0000000010000000" WHEN "11000",
25              "0000000001000000" WHEN "11001",
26              "0000000000100000" WHEN "11010",
27              "0000000000010000" WHEN "11011",
28              "0000000000001000" WHEN "11100",
29              "0000000000000100" WHEN "11101",
30              "0000000000000010" WHEN "11110",
31              "0000000000000001" WHEN "11111",
32
33              "0000000000000000" WHEN OTHERS;
34
35  END Behavior ;
```

Figure 9. The VHDL for the 4:16 Decoder

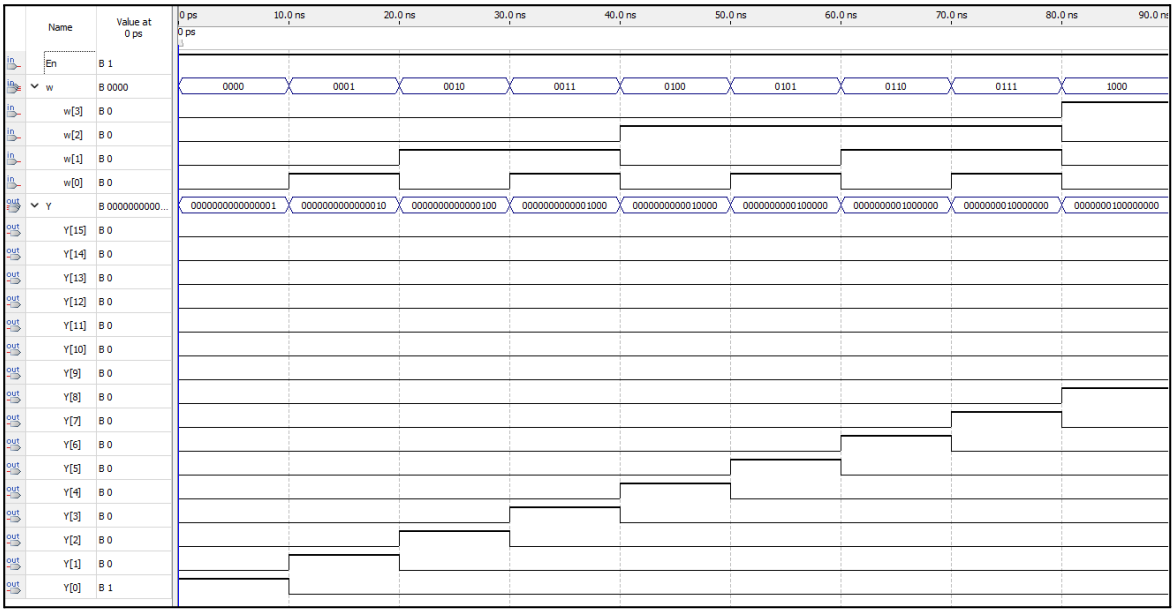

Figure 10. The Waveform for 4:16 Decoder

Finite State Machine (FSM)

The Finite State Machine, also referred to as FSM, is the last of the two components that make up the control unit. This specific FSM is designed to model a Moore Structure machine and works by cycling through 9 states consecutively off a rising clock. It will start from state 0 and end at state 8, before returning back to state 0 and starting the process all over again. For this lab, it will cycle through each state and output a 4-bit value for the current state in addition to a 4-bit value for a student ID digit corresponding to that state. The output for the current state will be passed to the 4:16 Decoder to produce a unique microcode, while the student ID digit will be passed to a seven segment unit to be displayed.

		Current State				**Next State**			
Student ID	**State**	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0
5	0	0	0	0	0	0	0	0	1
0	1	0	0	0	1	0	0	1	0
1	2	0	0	1	0	0	0	1	1
1	3	0	0	1	1	0	1	0	0

6	4	0	1	0	0	0	1	0	1
7	5	0	1	0	1	0	1	1	0
1	6	0	1	1	0	0	1	1	1
6	7	0	1	1	1	1	0	0	0
5	8	1	0	0	0	0	0	0	0

Table 2. The Truth Table for the FSM

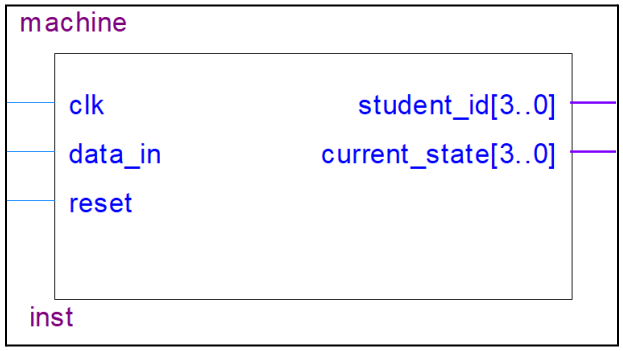

Figure 11. The Block Schematic for FSM

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity machine 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
14 architecture fsm of machine is
15
16     type state_type is (s0, s1, s2, s3, s4, s5, s6, s7, s8);
17
18     signal yfsm : state_type;
19
20 begin
21     process(clk, reset)
22     begin
23         if reset = '1' then
24             yfsm <= s0;
25         elsif (clk'EVENT AND clk = '1') then
26
27             case yfsm is
28                 when s0 =>
29                     IF (data_in = '0') then
30                         yfsm <= s0;
31                     ELSE
32                         yfsm <= s1;
33                     END IF;
34                 when s1 =>
35                     IF (data_in = '0') then
36                         yfsm <= s1;
37                     ELSE
38                         yfsm <= s2;
39                     END IF;
40                 when s2 =>
41                     IF (data_in = '0') then
42                         yfsm <= s2;
43                     ELSE
44                         yfsm <= s3;
45                     END IF;
```

```
46              when s3 =>
47                 IF (data_in = '0') then
48                    yfsm <= s3;
49                 ELSE
50                    yfsm <= s4;
51                 END IF;
52              when s4 =>
53                 IF (data_in = '0') then
54                    yfsm <= s4;
55                 ELSE
56                    yfsm <= s5;
57                 END IF;
58              when s5 =>
59                 IF (data_in = '0') then
60                    yfsm <= s5;
61                 ELSE
62                    yfsm <= s6;
63                 END IF;
64              when s6 =>
65                 IF (data_in = '0') then
66                    yfsm <= s6;
67                 ELSE
68                    yfsm <= s7;
69                 END IF;
70              when s7 =>
71                 IF (data_in = '0') then
72                    yfsm <= s7;
73                 ELSE
74                    yfsm <= s8;
75                 END IF;
76              when s8 =>
77                 IF (data_in = '0') then
78                    yfsm <= s8;
79                 ELSE
80                    yfsm <= s0;
81                 END IF;
82              END CASE;
83           END IF;
84        END process;
```

```
85
86      process (yfsm, data_in)
87      begin
88         case yfsm is
89            when s0 =>
90               student_id <= "0101";
91               current_state <= "0000";
92            when s1 =>
93               student_id <= "0000";
94               current_state <= "0001";
95            when s2 =>
96               student_id <= "0001";
97               current_state <= "0010";
98            when s3 =>
99               student_id <= "0001";
100              current_state <= "0011";
101           when s4 =>
102              student_id <= "0110";
103              current_state <= "0100";
104           when s5 =>
105              student_id <= "0111";
106              current_state <= "0101";
107           when s6 =>
108              student_id <= "0001";
109              current_state <= "0110";
110           when s7 =>
111              student_id <= "0110";
112              current_state <= "0111";
113           when s8 =>
114              student_id <= "0101";
115              current_state <= "1000";
116        END CASE;
117     END process;
118  END fsm;
```

Figure 12. The VHDL Code for FSM

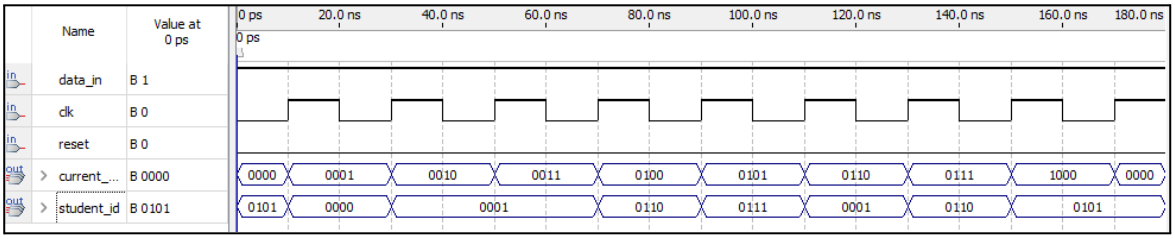

Figure 13. The Waveform for the FSM

Problem Set 1

The Arithmetic Logic Unit–also referred to as the ALU–is the core component that is responsible for all the calculations and operations. It is composed of a clock, two 8-bit input vectors for the values of *A* and *B*, and a 16-bit input vector OP for the microcode passed by the 4:16 Decoder. It also consists of two 4-bit output vectors–R1 and R2–that represent the first and last 4-bits of the result respectively, and neg that accounts for a negative result. The ALU works by taking in the 8-bit input from the two latches and performing arithmetic or logical operations based on the microcode given by the 4:16 Decoder. As a result of the ALU running on a rising clock, it will constantly check and update the operation given that the microcode from the 4:16 Decoder changes. This will allow for 9 unique operations that correspond to the 9 unique microcodes. After the operation is computed, the ALU will split the 8-bit result vector into two 4-bit outputs. This procedure is done in order to display the results corresponding hexadecimal value onto the FPGA board through the SSEG units. Additionally, if the result yields a negative value, the neg output will be a high signal and displayed onto the FPGA through its own SSEG unit. For the operations and results that correspond to the student number, refer to **Table 2**. Furthermore, refer to **Figure 19-27** for the results shown on the FPGA board.

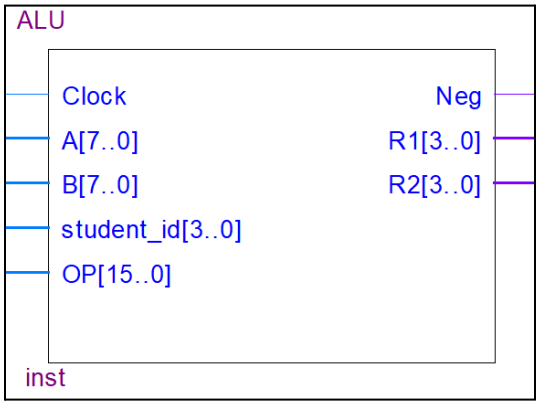

Figure 14. The Block Schematic for ASU_1

```
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

entity ALU is
   port(Clock : in std_logic; --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);
         R2 : out unsigned(3 downto 0));
end ALU;

architecture calculation of ALU is --temporary signal declariations
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" =>
                  --Do Subtraction
                  --"Neg" bit set if required.
                  IF Reg1 < Reg2 THEN
                     Result <= (Reg1 + (NOT Reg2 + 1)) ;
                     Neg <= '1';
                  ELSE
                     Result <= (Reg1 - Reg2) ;
                  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
                  Result <= (Reg1 XOR Reg2);
                  Neg <= '0';
               WHEN "0000000010000000" =>
                  --Do Boolean OR
                  Result <= (Reg1 OR Reg2);
                  Neg <= '0';
               WHEN "0000000100000000" =>
                  --Do Boolean XNOR
                  Result <= (Reg1 XNOR Reg2);
                  Neg <= '0';
               WHEN OTHERS =>
                  --Don't care, do nothing
                  Result <= "--------";
                  Neg <= '0';
            end case;
         end if;
      end process;
    R1 <= Result(3 downto 0);
    R2 <= Result(7 downto 4);
end calculation;
```

Figure 15. The VHDL Code for ALU_1

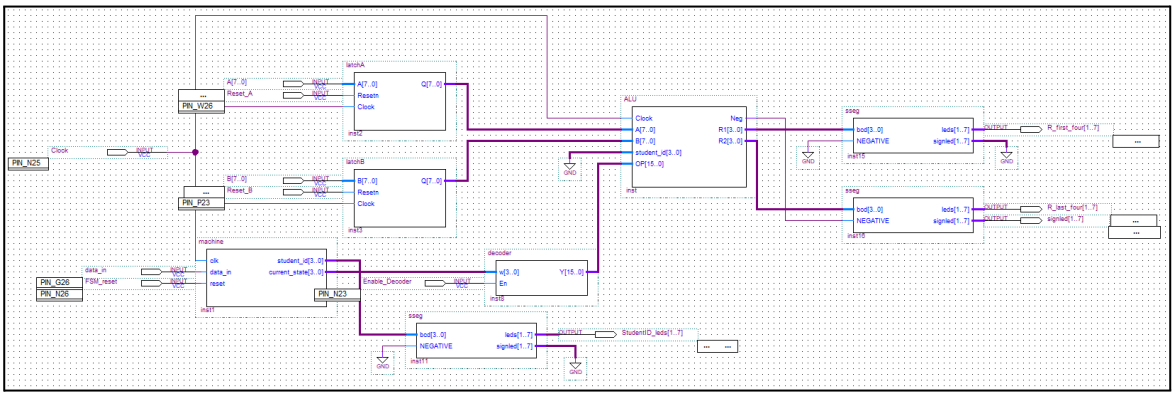

Figure 16. The Block Schematic for GPP_1

Function #	Microcode	Operation / Function	Binary Result	Hexadecimal Result
1	0000000000000001	Sum(**A**, **B**)	1101 0110	D6
2	0000000000000010	Diff(**A**, **B**)	0000 1100	0C
3	0000000000000100	$\overline{A}$	1000 1110	8E
4	0000000000001000	$\overline{A \cdot B}$	1001 1110	9E
5	0000000000010000	$\overline{A + B}$	1000 1010	8A
6	0000000000100000	$A \cdot B$	0110 0001	61
7	0000000001000000	$A \oplus B$	0001 0100	14
8	0000000010000000	$A + B$	0111 0101	75
9	000000010000000	$\overline{A \oplus B}$	1110 1011	EB

Table 3. ALU Operations and Results for Problem Set 1

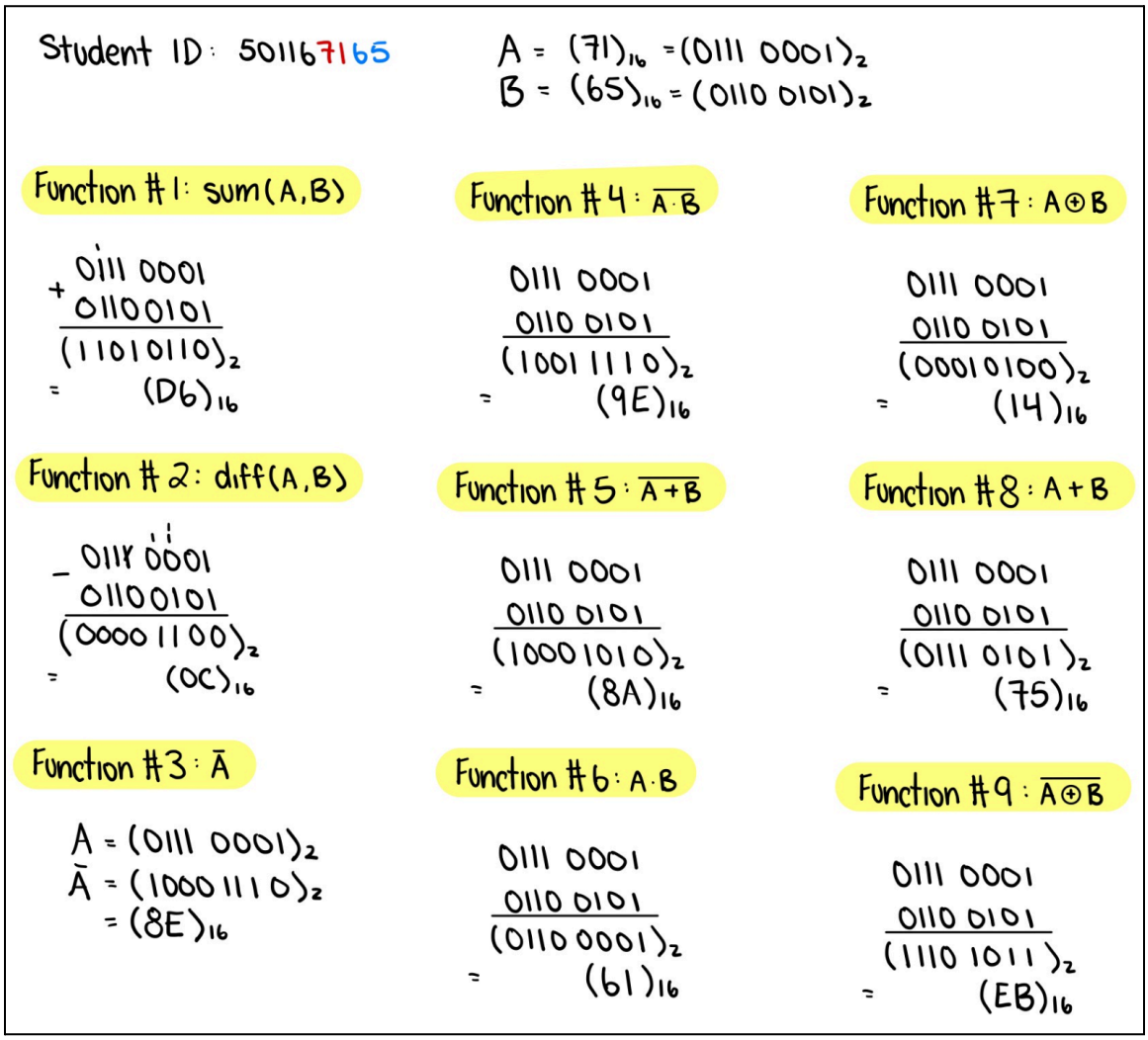

Figure 17. Handwritten Calculations for each Operation in Problem 1

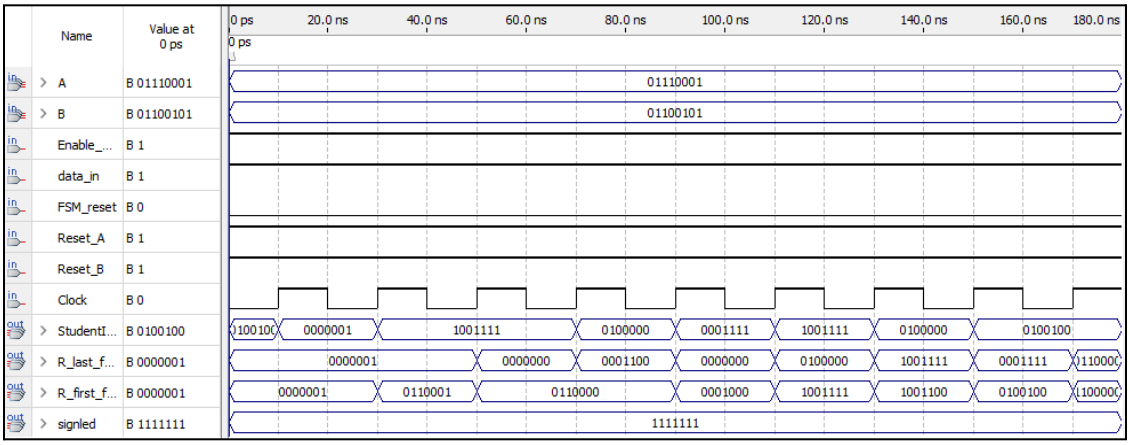

Figure 18. The Waveform for GPP_1

For the figures below, the hexadecimal values for operations are displayed on HEX0 and HEX2 for the FPGA board. In other words, there will be a gap in between the two values. Since the SSEG in between is not programmed, it will display its default setting which is an 8. Furthermore, upon inspection, it can be seen that the operation and student digits are not in sync with each other where the student digit displayed is one clock ahead of the operation. This is because the ALU runs on a rising clock that executes its operations when the clock goes from a low to high signal. The rising clock will take the stored microcode and run the operation based on it. It is important to note that it takes the stored microcode and not the current microcode that updates from the clock. Since the clock changes the student number and microcode in the FSM, it will display the next number and the operation for the previous microcode. This explains why the operation and student number are not synchronized. This happens for all problem sets (i.e. ALU_1, ALU_2, and ALU_3).

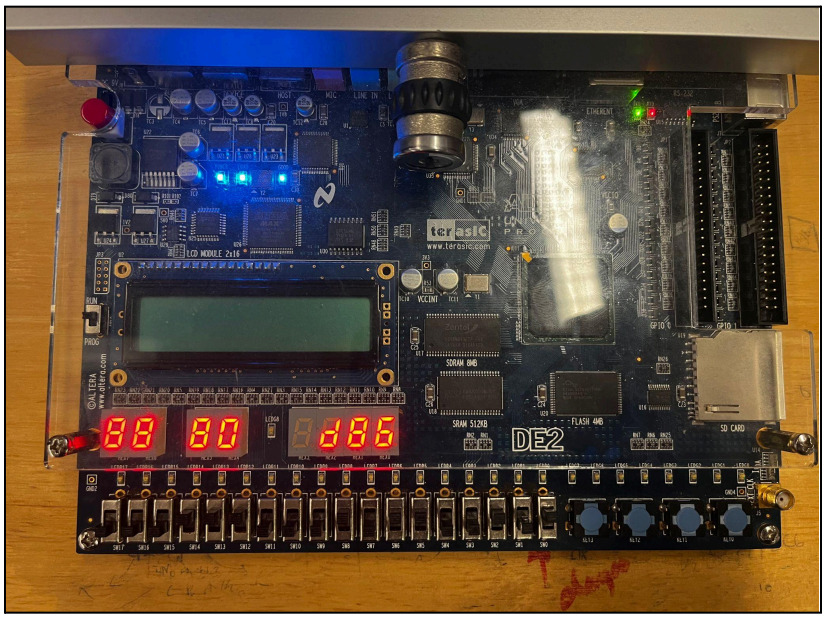

Figure 19. The FPGA Result for Problem Set 1, Function #1

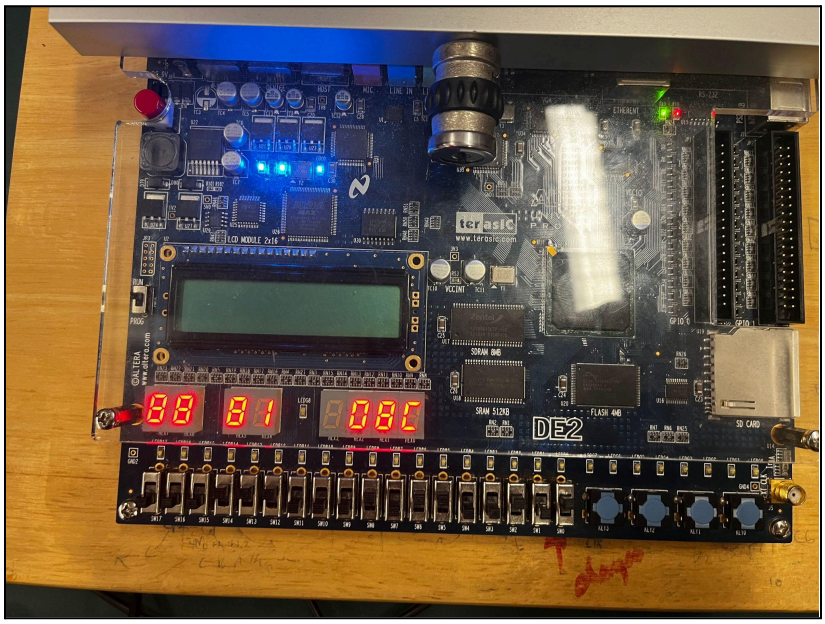

Figure 20. The FPGA Result for Problem Set 1, Function #2

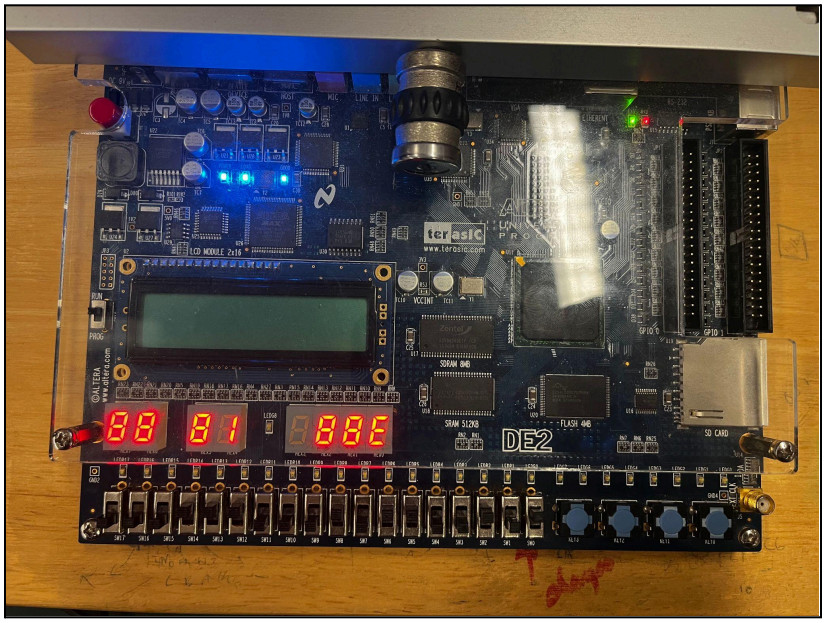

Figure 21. The FPGA Result for Problem Set 1, Function #3

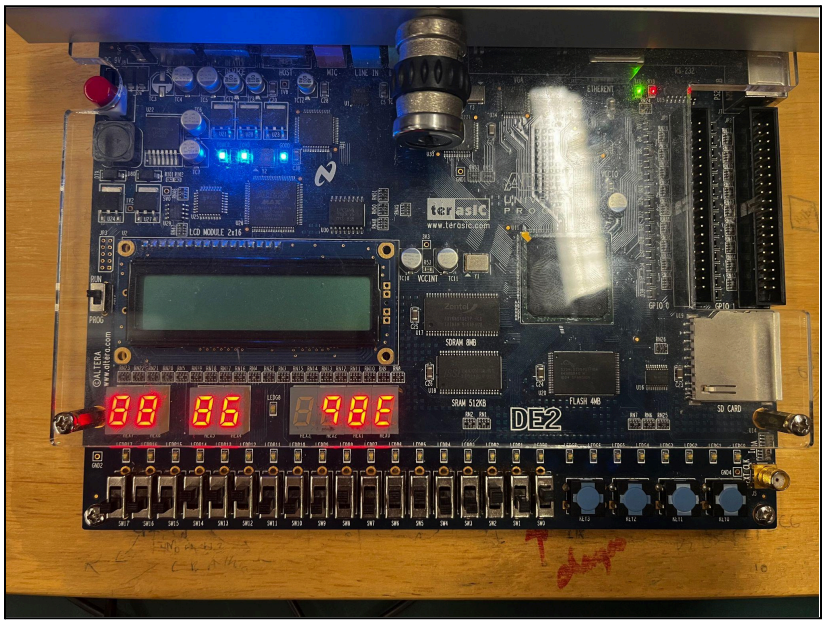

Figure 22. The FPGA Result for Problem Set 1, Function #4

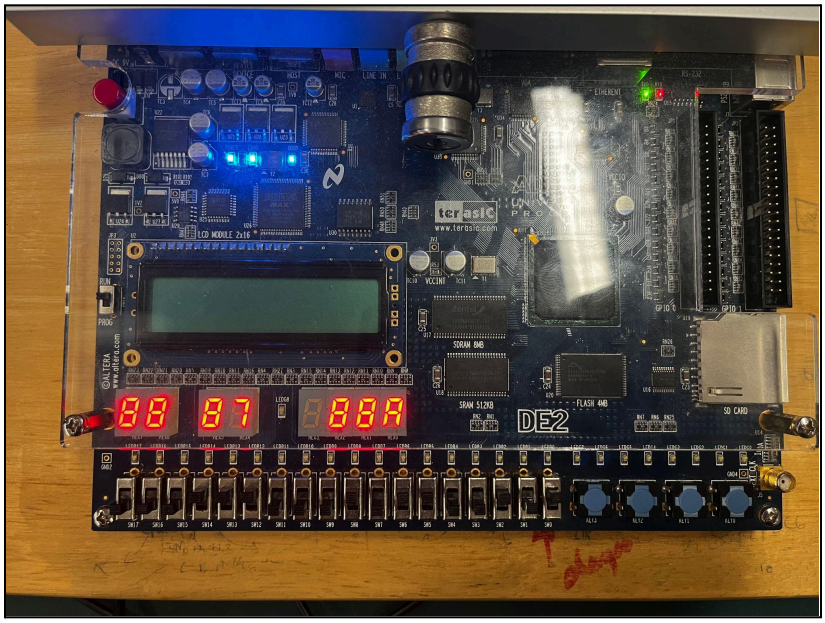

Figure 23. The FPGA Result for Problem Set 1, Function #5

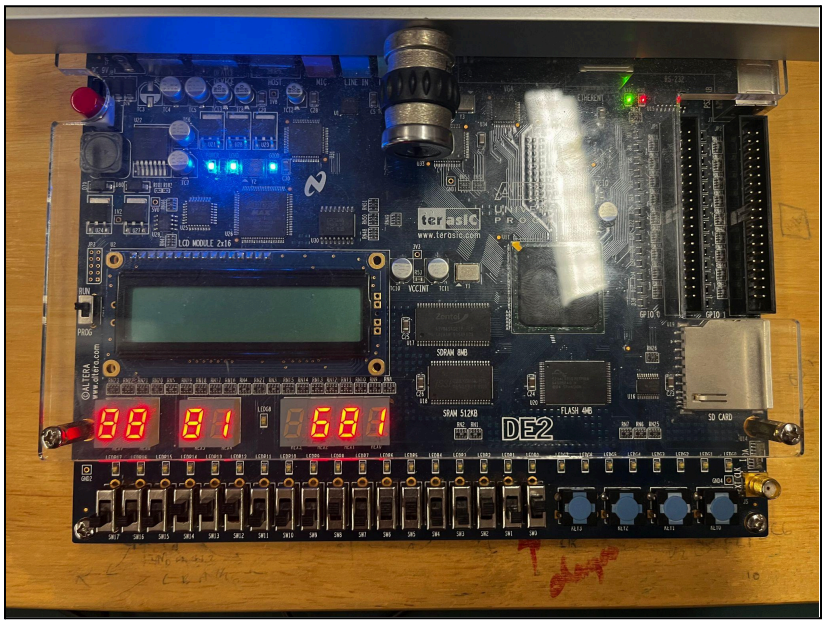

Figure 24. The FPGA Result for Problem Set 1, Function #6

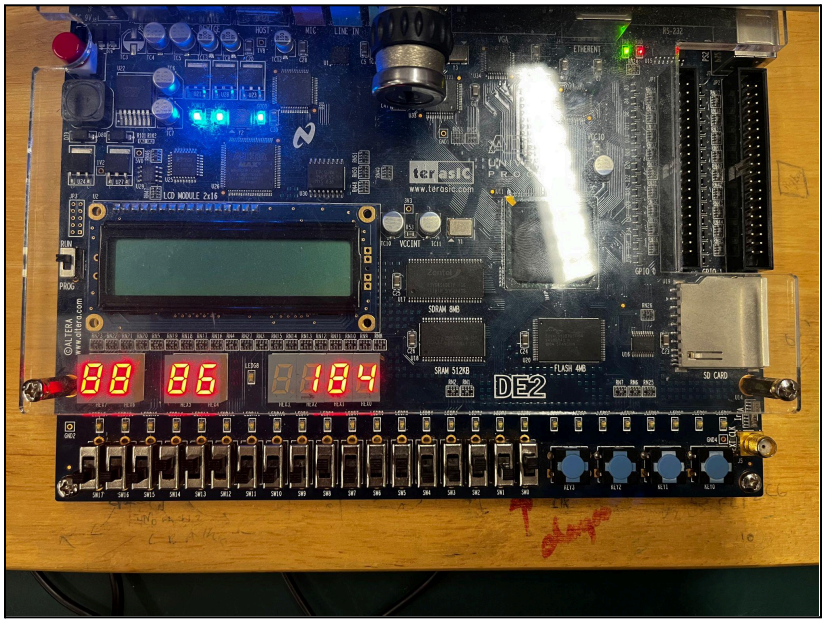

Figure 25. The FPGA Result for Problem Set 1, Function #7

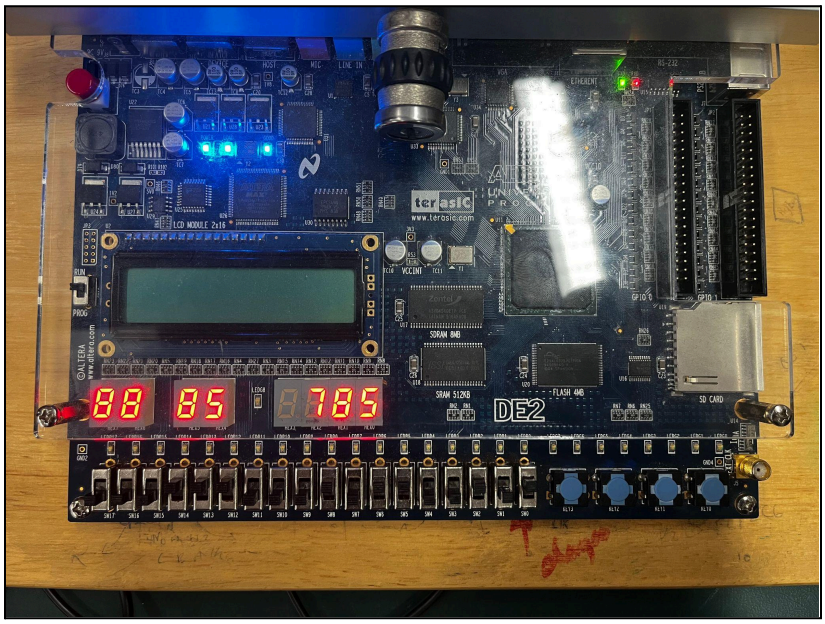

Figure 26. The FPGA Result for Problem Set 1, Function #8

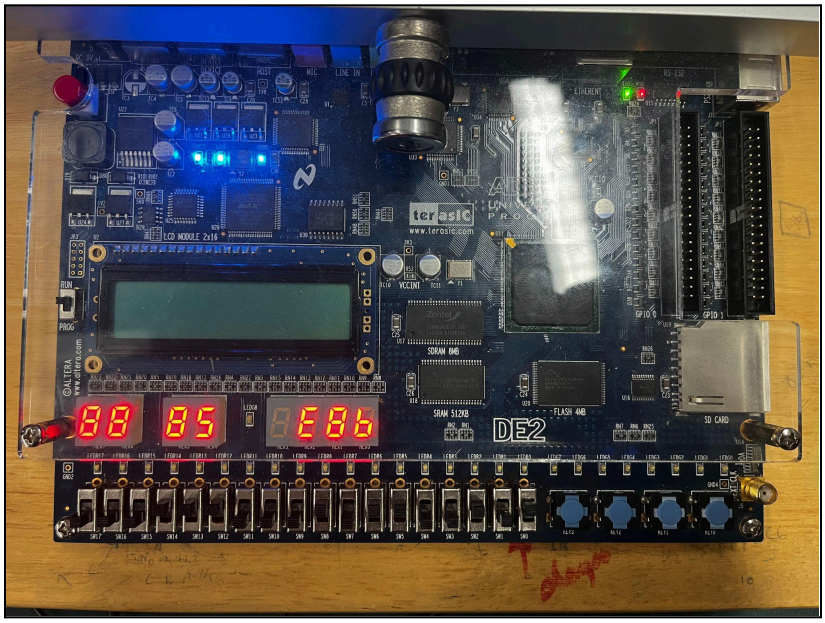

Figure 27. The FPGA Result for Problem Set 1, Function #9

Problem Set 2: Option B

For Problem Set 2, it will follow the same structure as Problem Set 1. However, for this Problem Set, the operations for the ALU will be different. It will match the operations stated for option B. For the operations and results corresponding to the student, refer to **Table 3**. Everything else including the inputs and outputs will stay the same for all components. The ALU will still run on a rising clock, updating the operation given that the microcode updates. All results will display on the FPGA through SSEG units. For the results on the FPGA board, refer to **Figure 33-41.**.

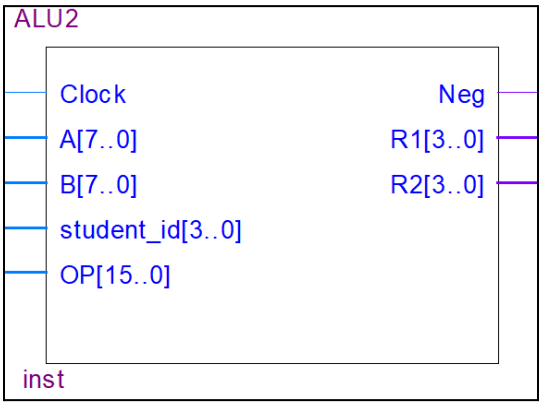

Figure 28. The Block Schematic for ALU_2

```
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);
        R2 : out unsigned(3 downto 0));
end ALU2;

architecture calculation of ALU2 is --temporary signal declariations
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" =>
                  --Swap the lower and upper 4 bits of A
                  Result <= shift_left(Reg1(3 downto 0) + "00000000", 4) + Reg1(7 downto 4);
                  Neg <= '0';
               WHEN "0000000000000010" =>
                  --Produce the result of ORing A and B
                  Result <= Reg1 OR Reg2;
                  Neg <= '0';
               WHEN "0000000000000100" =>
                  --Decrement B by 5
                  Result <= Reg2 - "0101";
                  Neg <= '0';
               WHEN "0000000000001000" =>
                  --Invert all bits of A
                  Result <= (Not Reg1);
                  Neg <= '0';
               WHEN "0000000000010000" =>
                  --Invert the bit-significance order of A
                  for i in 0 to 7 loop
                     Result(i) <= Reg1(7-i);
                  end loop;
                  Neg <= '0';
               WHEN "0000000000100000" =>
                  --Find the greater value of A and B and produce the results(Max(A,B))
                  IF Reg1 >= Reg2 THEN
                     Result <= Reg1;
                     Neg <= '1';
                  ELSE
                     Result <= Reg2;
                  END IF;
                  Neg <= '0';
               WHEN "0000000001000000" =>
                  --Produce the difference between A and B
                  IF Reg1 < Reg2 THEN
                     Result <= (Reg1 + (NOT Reg2 + 1));
                     Neg <= '1';
                  ELSE
                     Result <= (Reg1 - Reg2);
                  END IF;
               WHEN "0000000010000000" =>
                  --Produce the result of XNORing A and B
                  Result <= (Reg1 XNOR Reg2);
                  Neg <= '0';
               WHEN "0000000100000000" =>
                  --Rotate B to left by three bits
                  Result <= rotate_left(Reg2, 3);
                  Neg <= '0';
               WHEN OTHERS =>
                  --Don't care, do nothing
                  Result <= "--------";
                  Neg <= '0';
         end case;
      end if;
   end process;
   R1 <= Result(3 downto 0);
   R2 <= Result(7 downto 4);
end calculation;
```

Figure 29. The VHDL Code for ALU_2

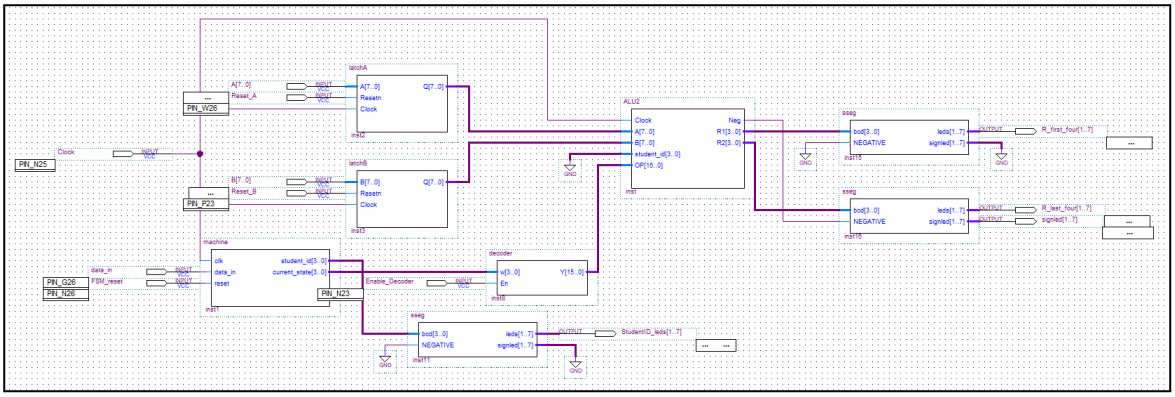

Figure 30. The Block Schematic for GPP_2

Function #	Microcode	Operation / Function	Binary Result	Hexadecimal Result
1	0000000000000001	Swap the lower and upper 4 bits of **A**	0001 0111	17
2	0000000000000010	Produce the result of ORing **A** and **B**	0111 0101	75
3	0000000000000100	Decrement **B** by 5	0110 0000	60
4	0000000000001000	Invert all bits of **A**	1000 1110	8E
5	0000000000010000	Invert the bit-significance of **A**	1000 1110	8E
6	0000000000100000	Find the greater value of **A** and **B** and produce the results (Max(**A**, **B**))	0111 0001	71
7	0000000001000000	Produce the difference between **A** and **B**	0000 1100	0C
8	0000000010000000	Produce the result of XNORing **A** and **B**	1110 1011	EB
9	0000000100000000	Rotate **B** to the left by three bits (ROL)	0010 1011	2B

Table 4. ALU Operations and Results for Problem Set 2

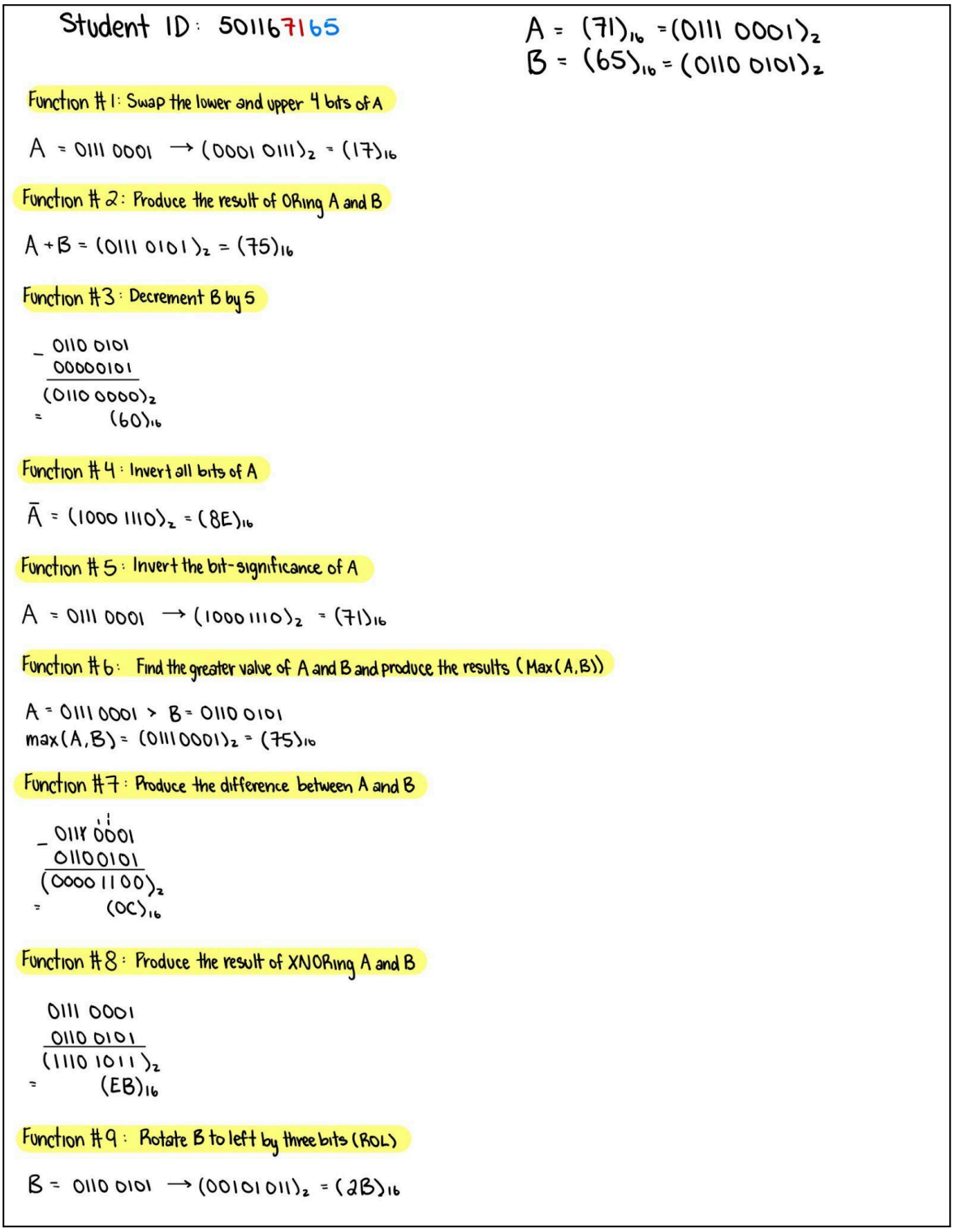

Figure 31. Handwritten Calculations for each Operation in Problem 2

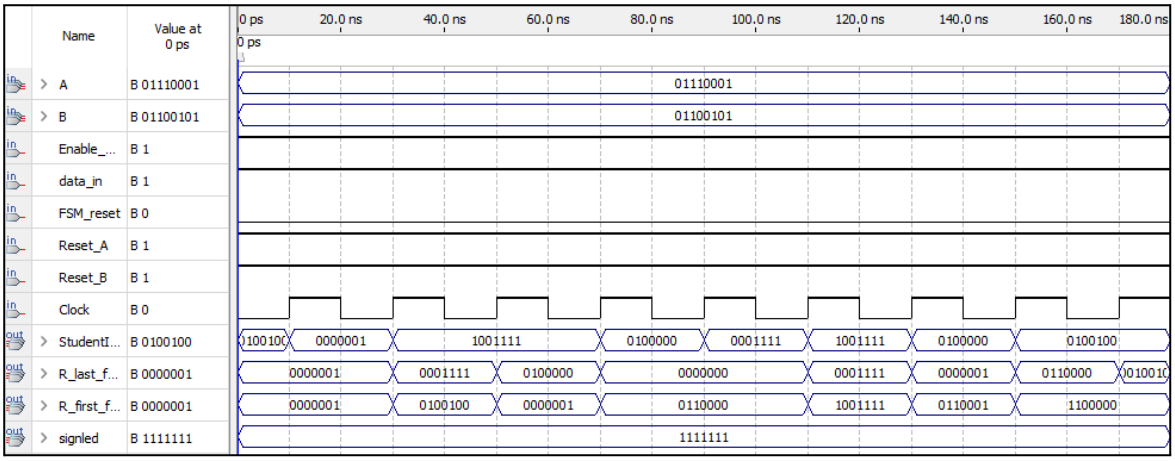

Figure 32. The Waveform for GPP_2

As explained in Problem Set 1, the student number and the operations are not synchronized.

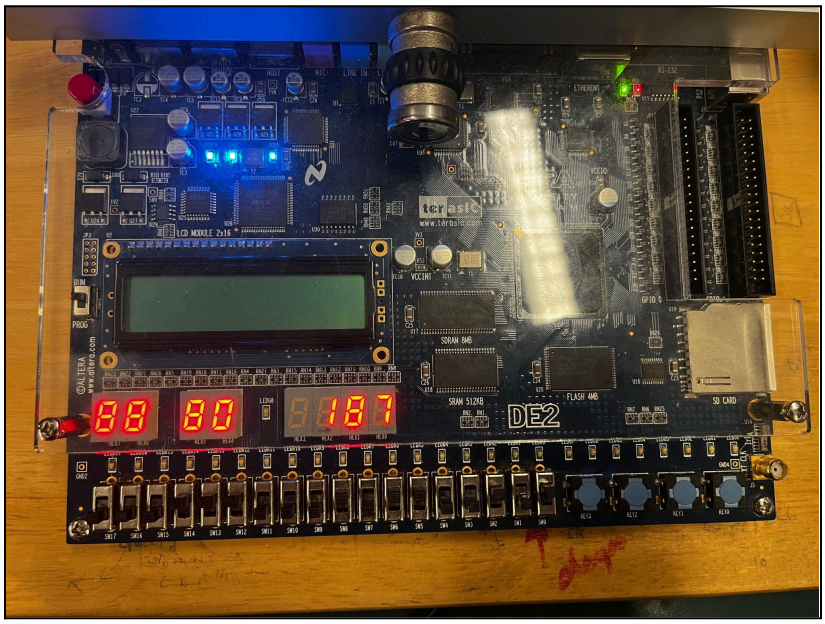

Figure 33. The FPGA Result for Problem Set 2, Function #1

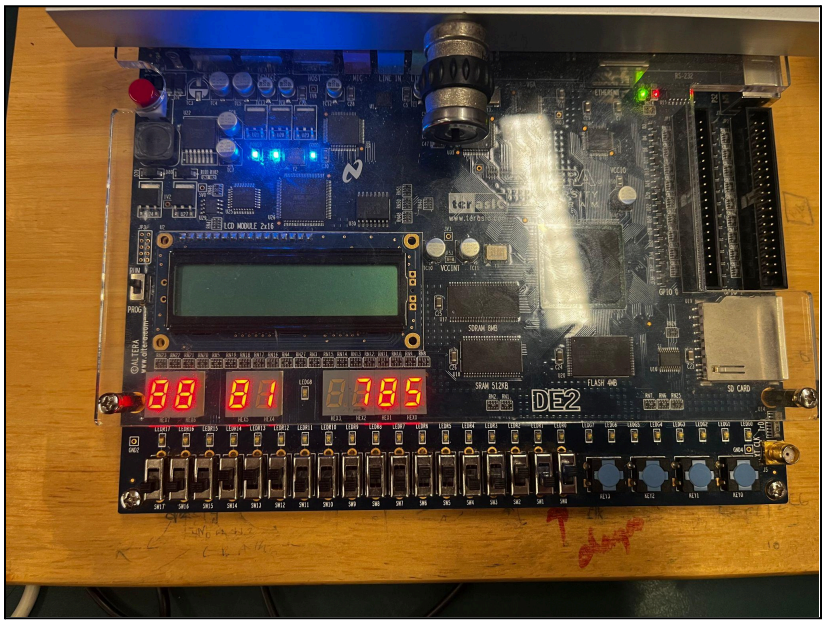

Figure 34. The FPGA Result for Problem Set 2, Function #2

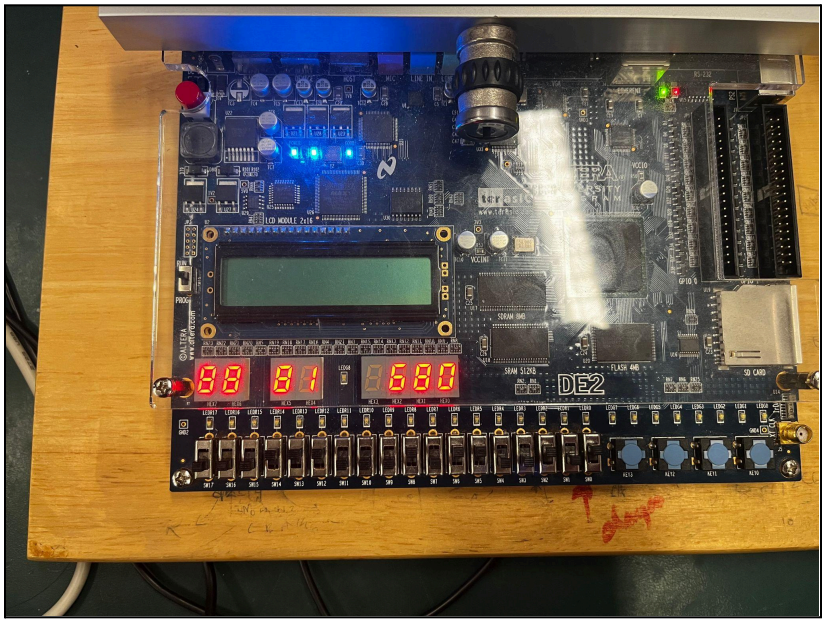

Figure 35. The FPGA Result for Problem Set 2, Function #3

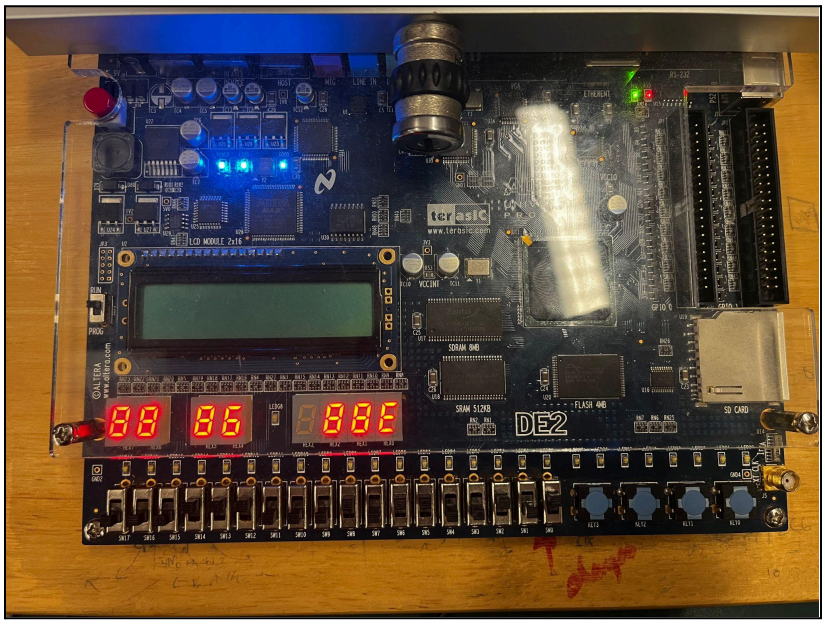

Figure 36. The FPGA Result for Problem Set 2, Function #4

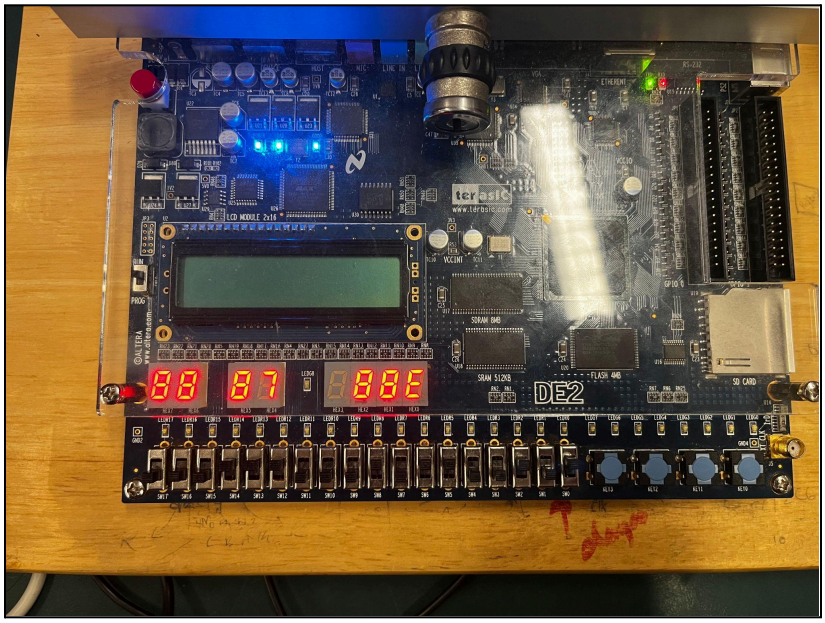

Figure 37. The FPGA Result for Problem Set 2, Function #5

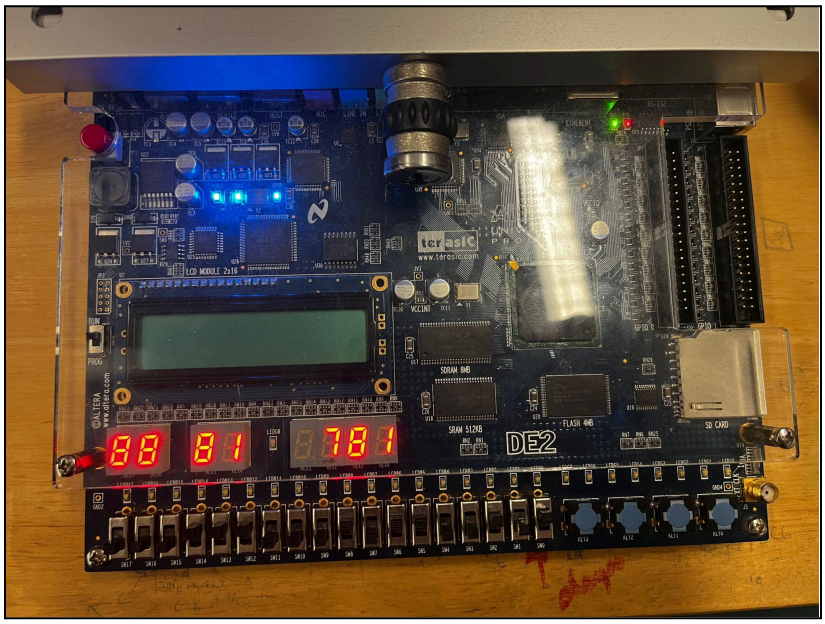

Figure 38. The FPGA Result for Problem Set 2, Function #6

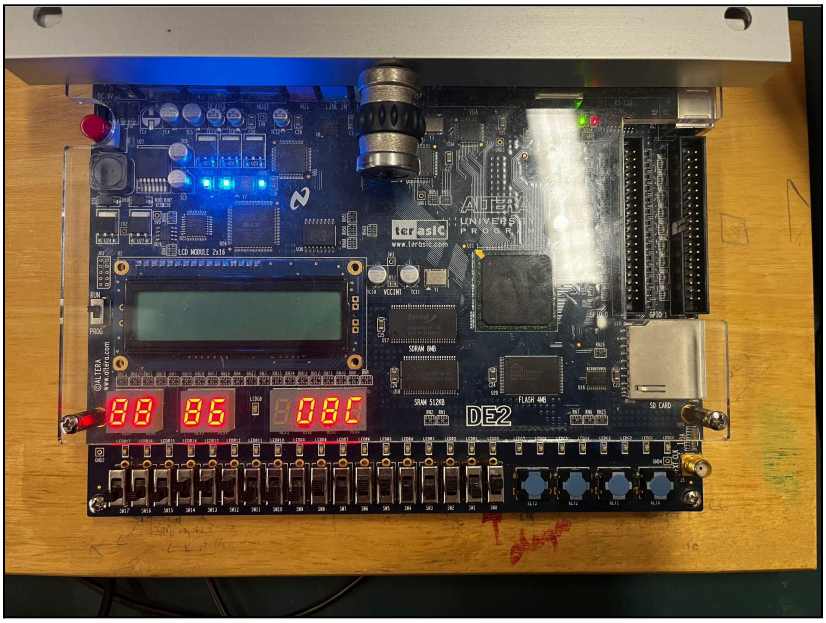

Figure 39. The FPGA Result for Problem Set 2, Function #7

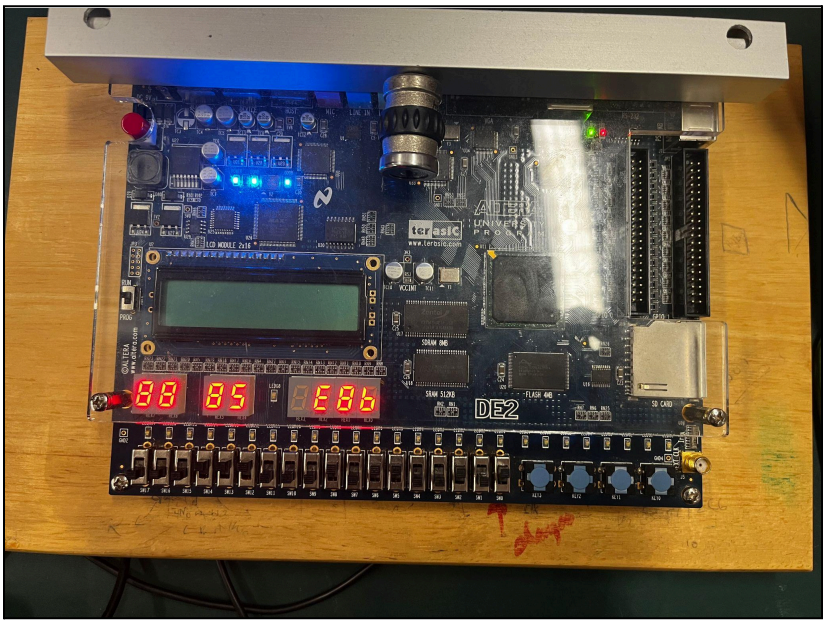

Figure 40. The FPGA Result for Problem Set 2, Function #8

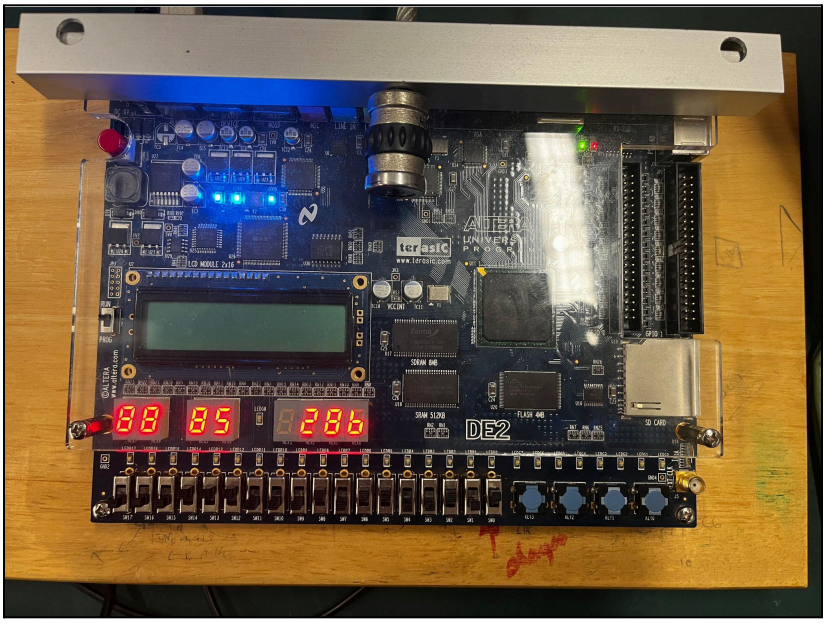

Figure 41. The FPGA Result for Problem Set 2, Function #9

Problem Set 3: Option B

Problem Set 3 will model the same structure as Problem Set 1 and 2, with a few minor modifications. Specifically, it will take and implement the instructions given from option B. For each digit of the student ID, the FPGA will display a ‘y’ for even digits, and ‘n’ for odd digits. In order to replicate this, the ALU has to be modified such that it will take the 4-bit output of the current digit of the student ID from the FSM as input. Thus, a 4-bit vector input student_id will be utilized in the ALU. Then, using the modulus operation in the VHDL code, the following line–student_id mod ‘0010’–can be executed to check for evenness. If the operation returns a remainder of 0, it is even, otherwise it is odd. For each case, there will be a unique 4-bit code that will be passed to a modified SSEG unit that will display a ‘y’ or ‘n’ on the FPGA board for even and odd digits respectively. Similar to the first two problem sets, the ALU will be running on a rising clock that will update and check the current student ID digit that is passed by the FSM. Furthermore, the student ID digit from the FSM will be displayed from a second SSEG unit in addition to output from the ALU. For the operations and results that correspond to the student number, refer to **Table 5**. Furthermore, refer to **Figure 46-54** for the results shown on the FPGA board.

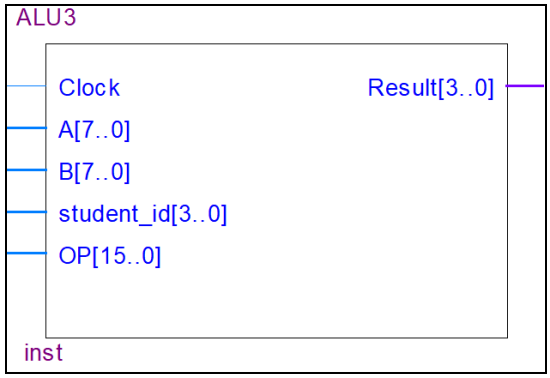

Figure 42. The Block Schematic for ALU_3

```vhdl
LIBRARY ieee;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

entity ALU3 is
   port(Clock : in std_logic; --input clock signal
         A, B : in unsigned(7 downto 0); --8-bit inputs from latches A and B
         student_id : in unsigned(3 downto 0); --4 bit student id from FSM
         OP : in unsigned(15 downto 0); --16-bit selector for Operation from Decoder
         Result : out unsigned(3 downto 0));
end ALU3;

architecture calculation of ALU3 is --temporary signal declariations
signal Reg1, Reg2 : unsigned(7 downto 0) :=(others => '0');
signal Result1 : unsigned(3 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, student_id)
   begin
      if(rising_edge(Clock)) THEN --Do the calculation @ positive edge of clock cycle
         if(student_id mod "0010") = 0 THEN
            Result1 <= "0001";
         else
            Result1 <= "0000";
         end if;
      end if;
   end process;
Result <= Result1;
end calculation;
```

Figure 43. The VHDL Code for GPP_3

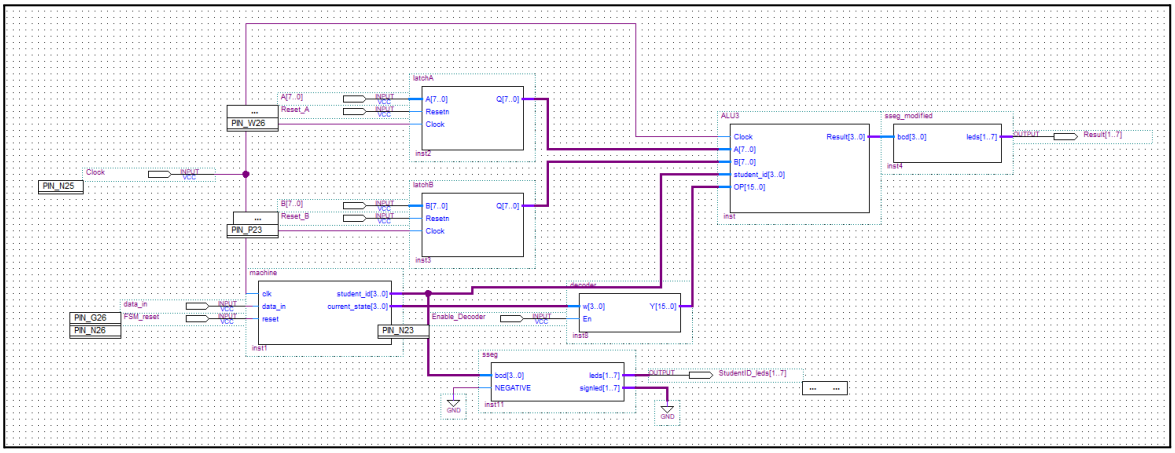

Figure 44. The Block Schematic for GPP_3

Student Number	State	Output	FPGA Output
5	0	n	n
0	1	y	n
1	2	n	y
1	3	n	n
6	4	y	n
7	5	n	y
1	6	n	n
6	7	y	n
5	8	n	y

Table 5. ALU Results for Problem Set 3

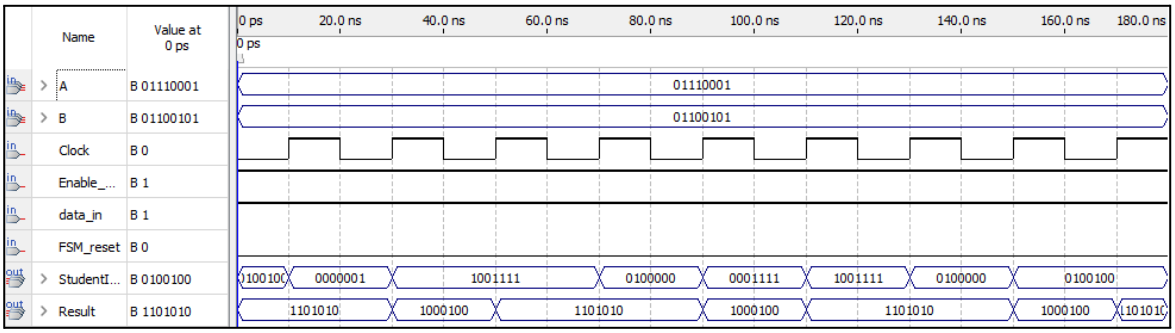

Figure 45. The Waveform for GPP_3

As explained in Problem Set 1, the student number and the operations are not synchronized. For the figures below, the operation that displays ‘y’ for even digits and ‘n’ for odd digits are not synced. It displays the result for the previous student digit. For example, in **Figure 46**, it displays ‘n’ for 0 because the operation is checking the previous number 5. For the next number in **Figure 47**, it displays ‘y’ for 1 because it is checking the previous number 0.

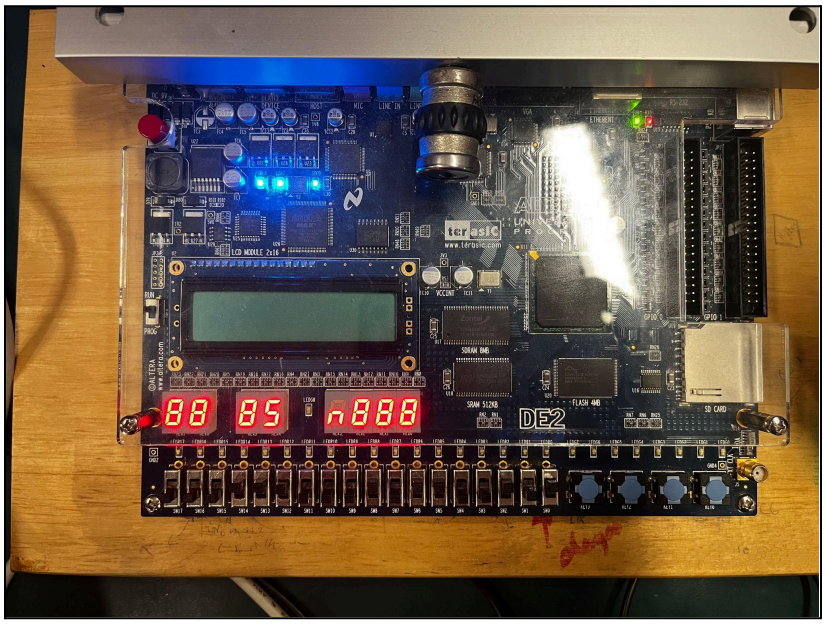

Figure 46. The FPGA Result for Problem Set 3, Student Digit 1

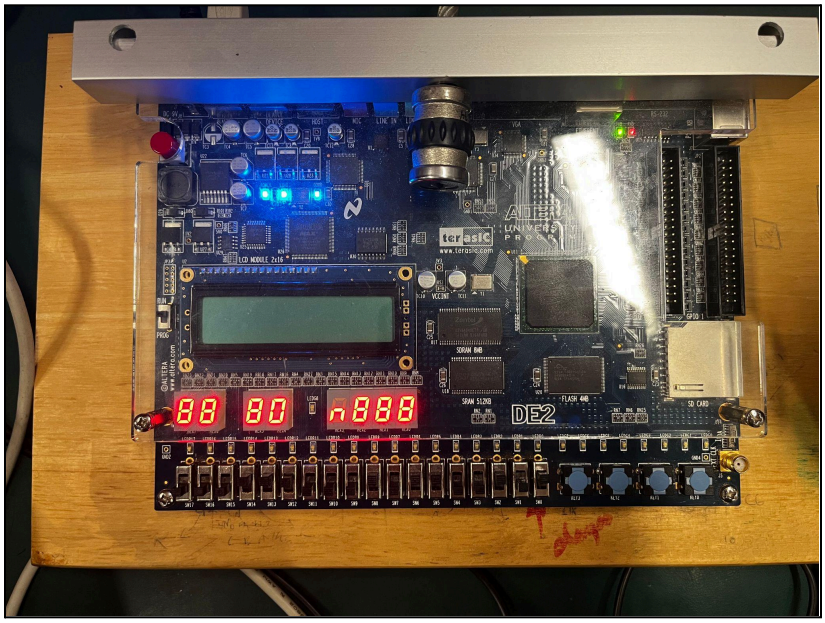

Figure 47. The FPGA Result for Problem Set 3, Student Digit 2

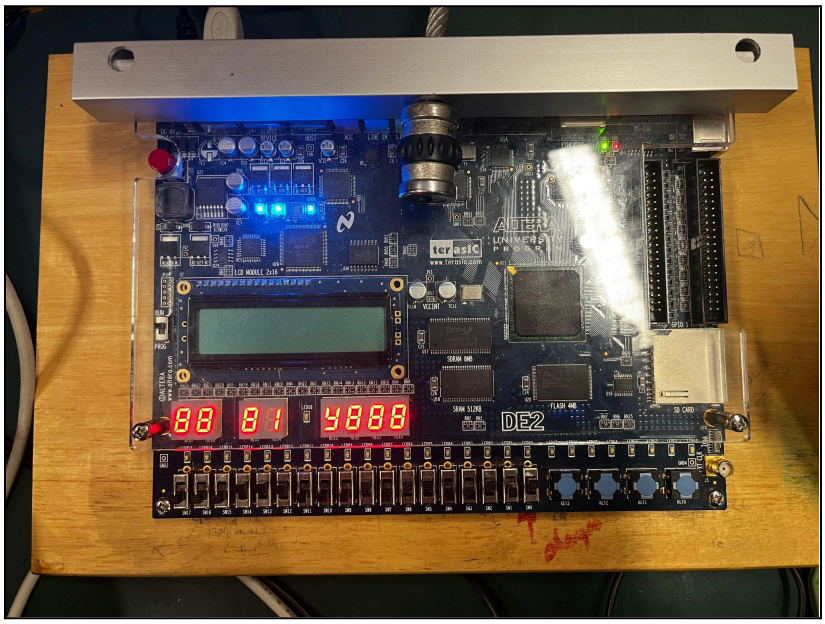

Figure 48. The FPGA Result for Problem Set 3, Student Digit 3

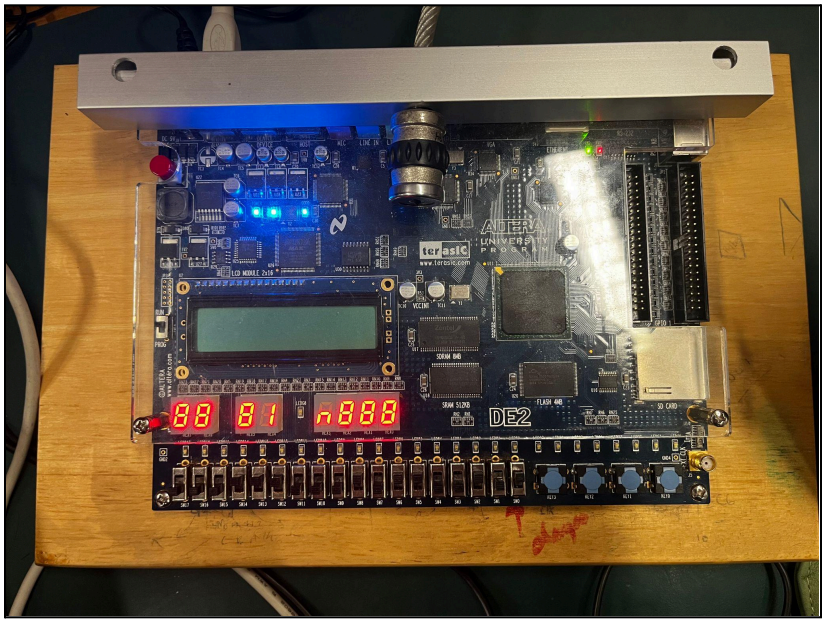

Figure 49. The FPGA Result for Problem Set 3, Student Digit 4

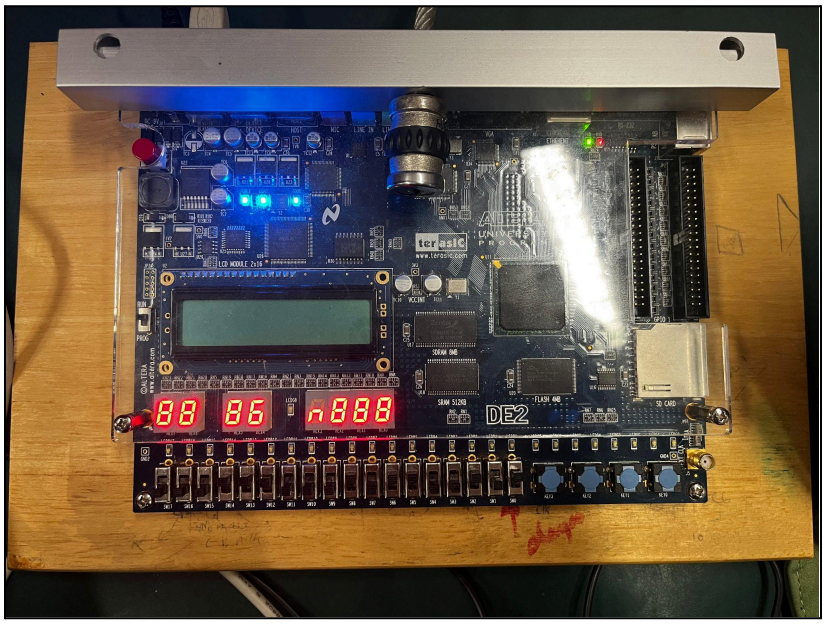

Figure 50. The FPGA Result for Problem Set 3, Student Digit 5

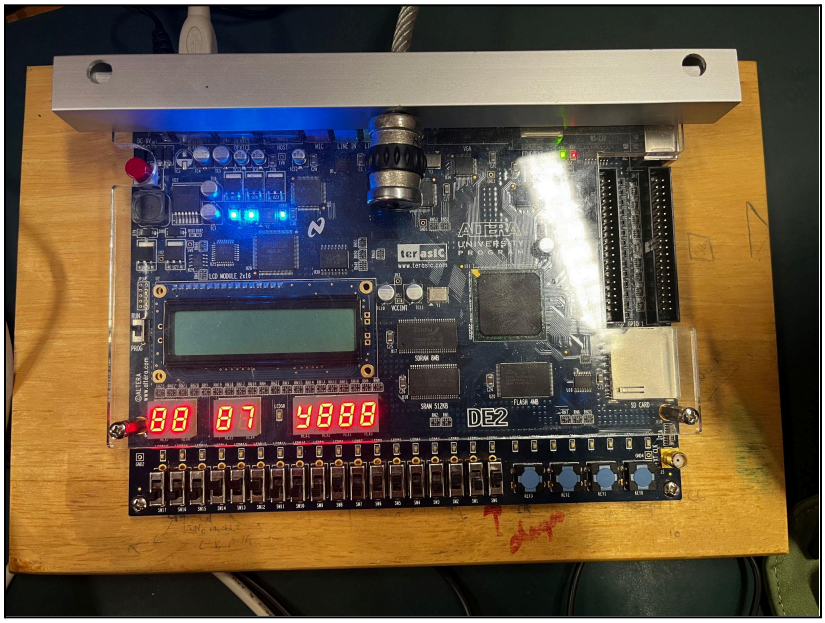

Figure 51. The FPGA Result for Problem Set 3, Student Digit 6

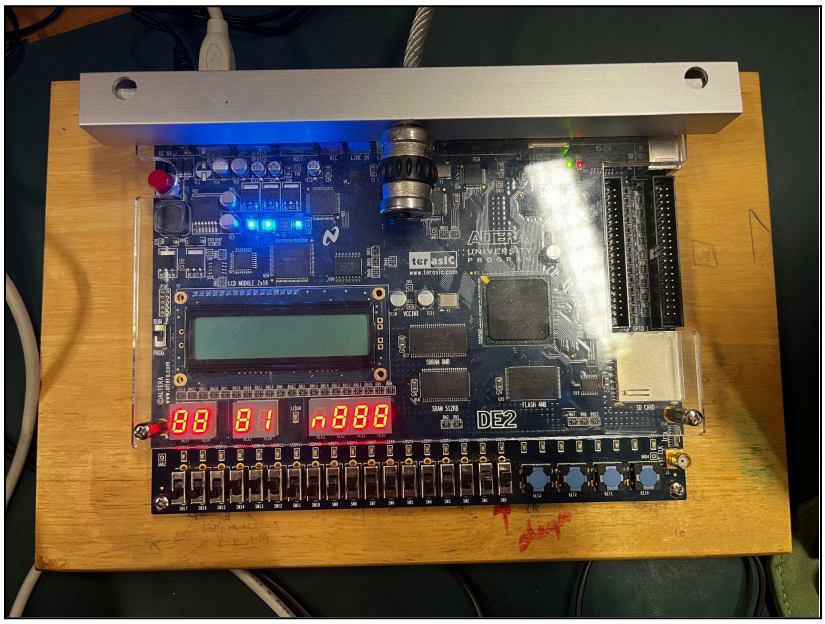

Figure 52. The FPGA Result for Problem Set 3, Student Digit 7

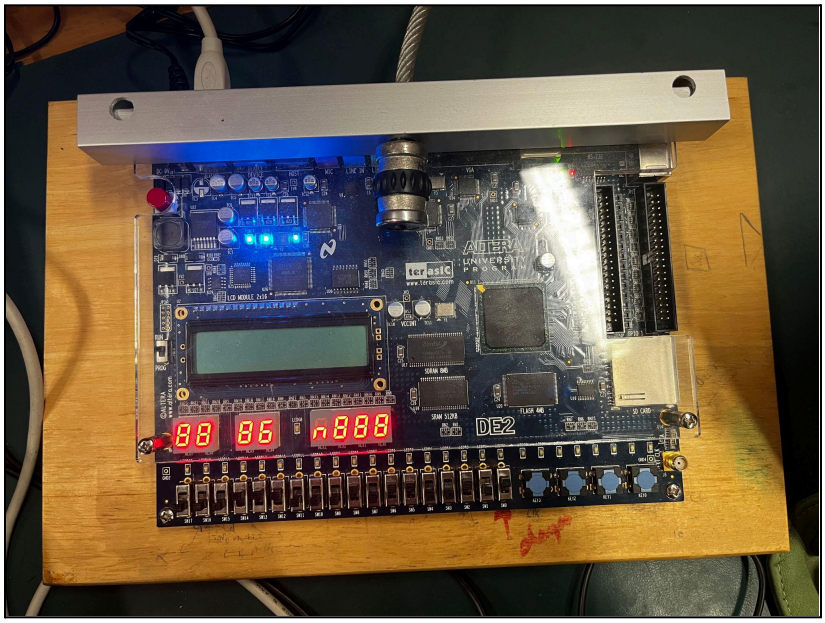

Figure 53. The FPGA Result for Problem Set 3, Student Digit 8

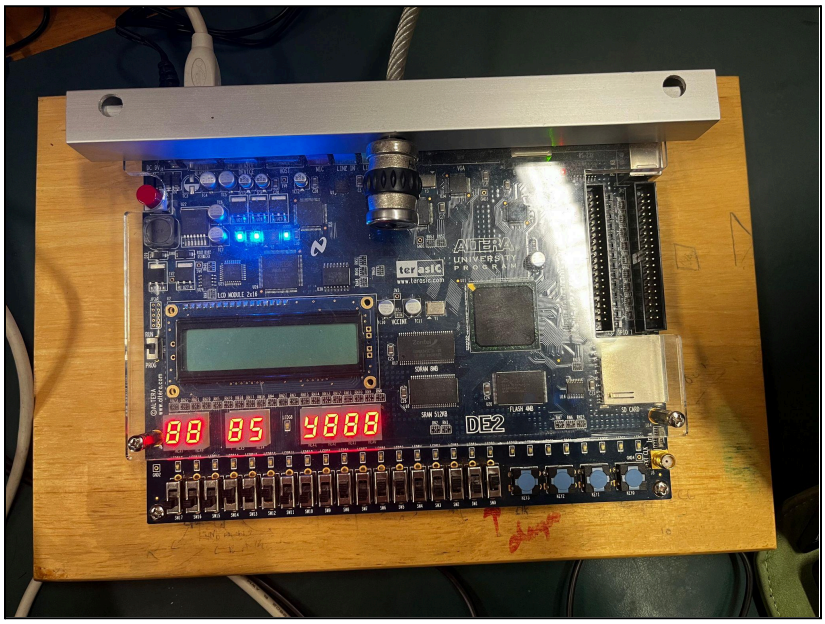

Figure 54. The FPGA Result for Problem Set 3, Student Digit 9

Conclusion

The focus of this lab was to construct and create an ALU component that could conduct several operations in the VHDL environment based on multiple parameters. These parameters include several inputs, the current state from the FSM, and the student ID number. A number of components had to be coded for the purpose of aiding the ALU and providing it with such parameters. These components include the seven-segment display, storage unit, control unit, and arithmetic logic unit. The creation of these components were possible through the use of material and knowledge from previous laboratories. After assembling these parts, the FPGA board implementation performed as intended. Thus, the creation, compilation, and testing of three distinct microprocessors were successfully accomplished in this lab.