Course Number:	COE328
Course Title:	Digital Systems
Semester/Year:	
Instructor:	
TA:	

Lab/Tutorial Report No.	6
Report Title:	Design of a Simple Central Processing Unit

Section No.	
Submission Date:	
Due Date:	

Student Name	Student ID	Signature
Ganojan		

Table of Contents

Introduction

The objective of this lab is to design a simple CPU (central processing unit). This is done by implementing an ALU (arithmetic logic unit) and using VHDL code to create a network of different components that work together to perform various simple calculations and operations on the FPGA board. The important aspects of the operation of the CPU is the input data, the storage units, the control unit and the ALU core. The input data consists of two 8-bit inputs, A and B, corresponding to the last 4 digits of the student's ID. This data will be inputted to the storage units, where they will temporarily store the data so they can be used by the ALU core to perform the necessary calculations. The control unit supplies the "opcode", or the instruction to the ALU, stating which calculation to perform. It is composed of an FSM and 4x16 decoder, and when working together, the decoder outputs the current calculation operation to the ALU depending on the current state of the FSM. All the components are connected to the same clock source so that every time the system is clocked on, the components work synchronously to almost instantaneously produce the correct and desired output to the FPGA board.

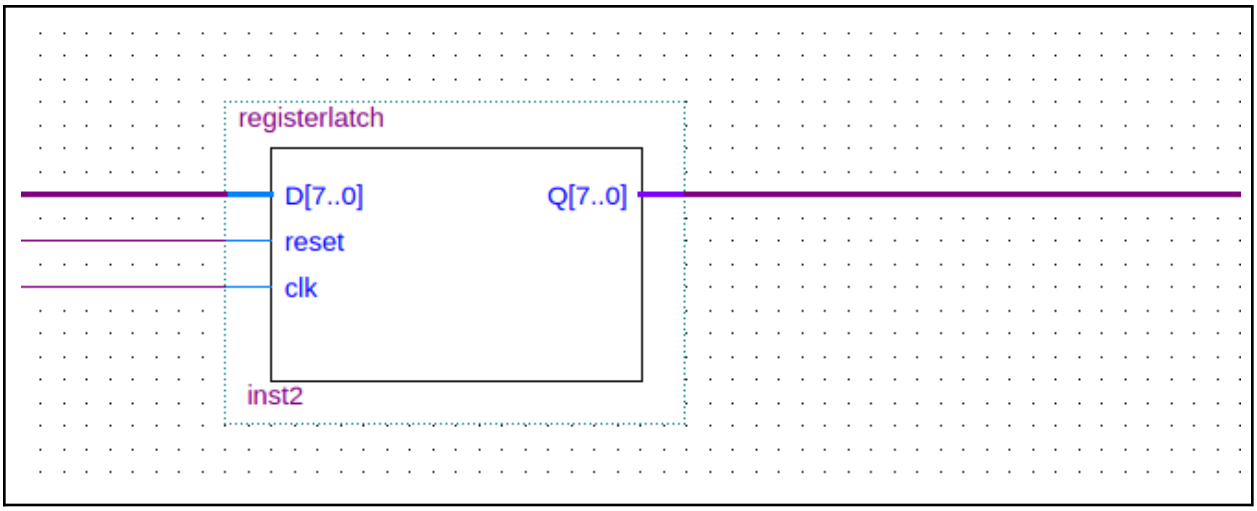
Components

Description of Four Important Components

The storage units will consist of 2 latches with each one storing an 8-bit input determined by the last 4 digits of the student's ID. Of the 4 digits, the first 2 digits will be converted to two 4-bit binary numbers (for an 8-bit input A at Latch 1) and the last 2 digits will be converted to two 4-bit binary numbers (for an 8-bit input B at Latch 2). The FSM will act as an up-counter to cycle through the 8 states, in accordance with the last 8 digits of the student's ID. As the FSM moves through each state, it will output the current state information which will be received by the 4-to-16 decoder. The decoder can then use this information to tell the ALU which action to perform depending on the current state of the system.

Latch1 and Latch2

Screenshot of Latch Block Diagram

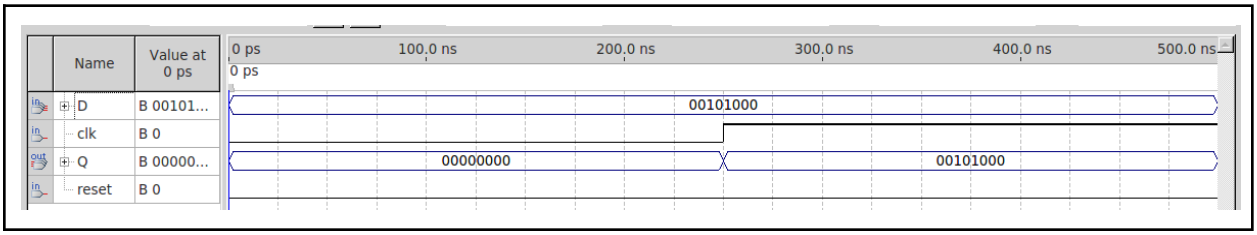


Latch Truth Table

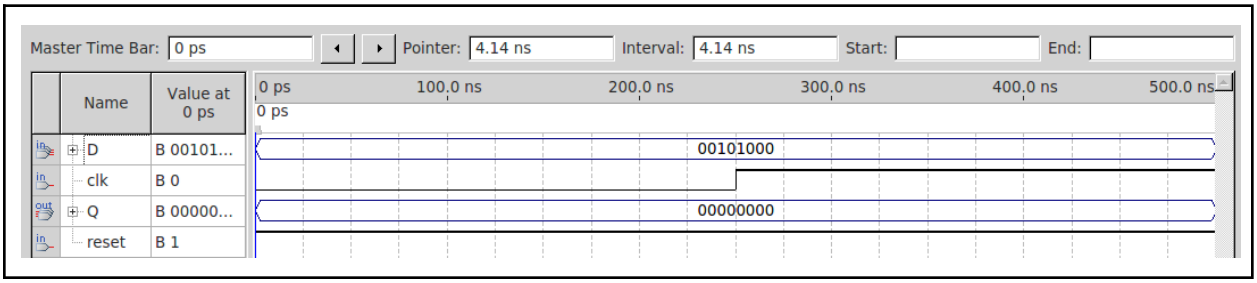
Latch 1 and Latch 2 Truth Table						
Input	**Reset**		**Input**		**Output**	
			Decimal	**8-Bit Binary**	**Reset = 0**	**Reset = 1**
A	1	0	28	00101000	00101000	00000000
B	1	0	65	01100101	01100101	00000000

Latch Waveforms

Reset = 0



Reset = 1



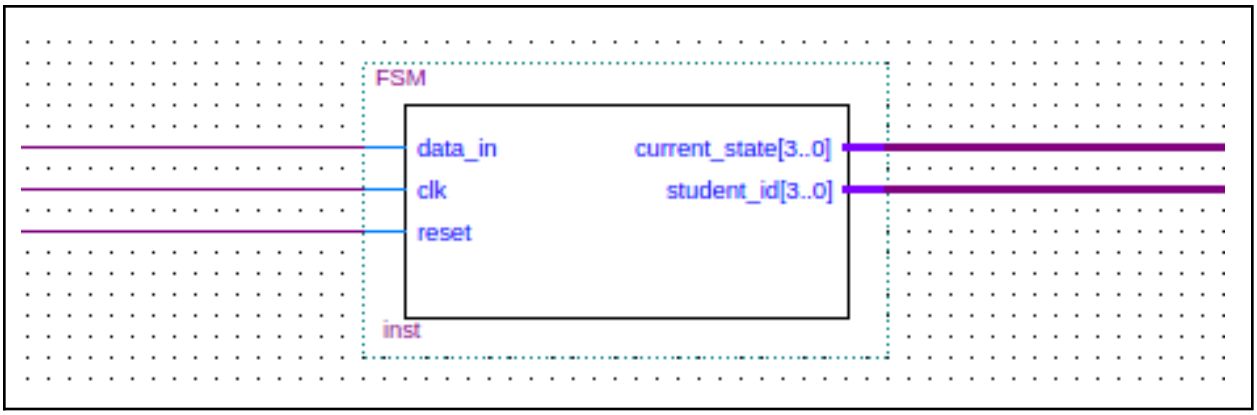
From the above waveforms, it can be seen that the output of the latch is whatever the input was when the system is clocked while Reset = 0. When Reset is set to 1, only 00000000 goes to the output.

Latch VHDL Code

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY registerlatch IS
5           port(    D     : in   std_logic_vector (7 DOWNTO 0); -- 8 bit input
6           reset, clk     : in   std_logic;
7                    Q     : out  std_logic_vector (7 DOWNTO 0)); -- 8 bit output
8   END registerlatch;
9
10  ARCHITECTURE Behavior OF registerlatch IS
11  BEGIN
12     PROCESS(reset,clk)
13     BEGIN
14        if reset = '1' then
15           Q <= "00000000";
16        elsif(clk'EVENT AND clk = '1') then
17           Q <= D;
18        END if;
19     END PROCESS;
20  END Behavior;
```

Moore FSM

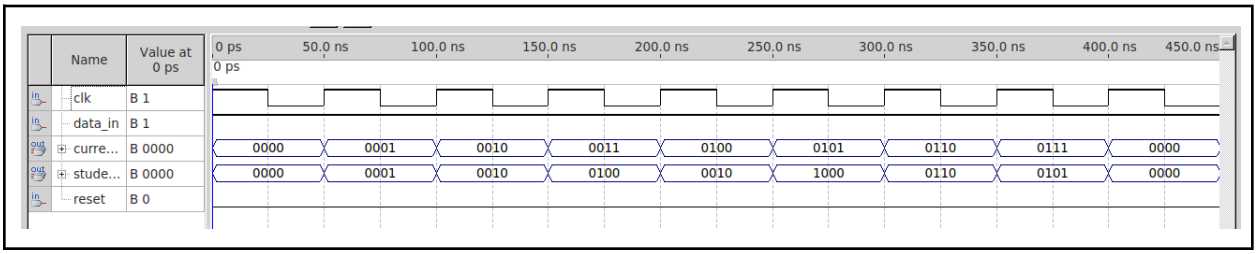
Screenshot of FSM Block Diagram



FSM Truth Table

Moore FSM Truth Table					
Current State		**Next State**		**Output (Student ID)**	
Decimal	**Binary (4-bit)**	**Data = 0**	**Data = 1**	**Decimal**	**Binary (4-bit)**
0	0000	0000	0001	0	0000
1	0001	0001	0010	1	0001
2	0010	0010	0011	2	0010
3	0011	0011	0100	4	0100
4	0100	0100	0101	2	0010
5	0101	0101	0110	8	1000
6	0110	0110	0111	6	0110
7	0111	0111	0000	5	0101

FSM Waveform (data_in = 1)



In the above waveform, it can be seen that the FSM outputs the correct student ID digit in 4-bit binary as the state goes up one by one. It is important to note that the data_in = 0 and reset = 1 waveform possibilities are not included as they are not of any use to us.

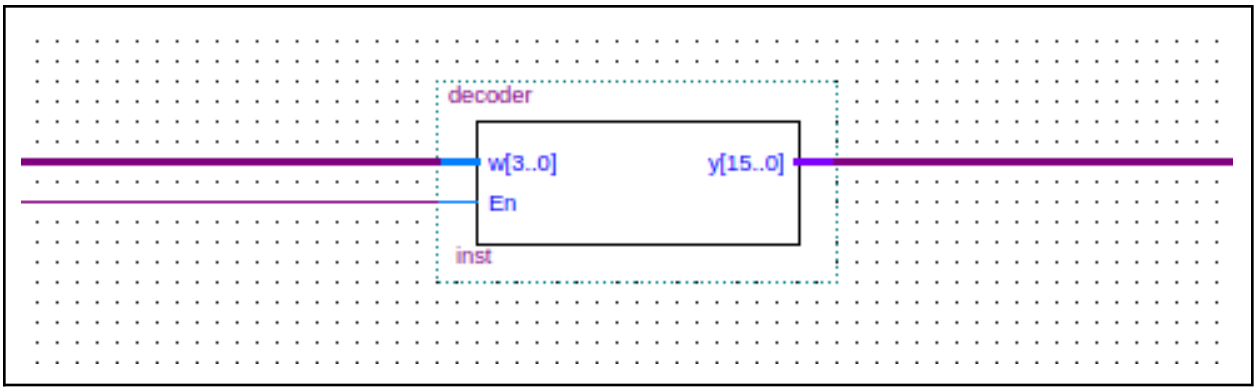
FSM VHDL Code

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   entity FSM IS
5         port (data_in, clk, reset     : in  std_logic;
6                          current_state : out std_logic_vector(3 DOWNTO 0);
7                            student_id  : out std_logic_vector(3 DOWNTO 0));
8   end FSM;
9
10  architecture fsm of FSM is
11     type state_type is (s0,s1,s2,s3,s4,s5,s6,s7);
12     signal yfsm : state_type;
13  begin
14     process (clk,reset)
15     begin
16        if reset = '1' then
17           yfsm <= s0;
18        elsif (clk'EVENT AND clk = '1') then --POSITIVE EDGE TRIGGER
19           case yfsm is
20              when s0 =>
21                 if data_in = '1' then
22                    yfsm <= s1;
23                 end if;
24              when s1 =>
25                 if data_in = '1' then
26                    yfsm <= s2;
27                 end if;
28              when s2 =>
29                 if data_in = '1' then
30                    yfsm <= s3;
31                 end if;
32              when s3 =>
33                 if data_in = '1' then
34                    yfsm <= s4;
35                 end if;
```

```
36              when s4 =>
37                 if data_in = '1' then
38                    yfsm <= s5;
39                 end if;
40              when s5 =>
41                 if data_in = '1' then
42                    yfsm <= s6;
43                 end if;
44              when s6 =>
45                 if data_in = '1' then
46                    yfsm <= s7;
47                 end if;
48              when s7 =>
49                 if data_in = '1' then
50                    yfsm <= s0;
51                 end if;
52           end case;
53        end if;
54     end process;
55
56     process (yfsm)
57     begin
58        case yfsm is
59           when s0 => current_state <= "0000";
60              student_id <= "0000"; --d2
61           when s1 => current_state <= "0001";
62              student_id <= "0001"; --d3
63           when s2 => current_state <= "0010";
64              student_id <= "0010"; --d4
65           when s3 => current_state <= "0011";
66              student_id <= "0100"; --d5
67           when s4 => current_state <= "0100";
68              student_id <= "0010"; --d6
69           when s5 => current_state <= "0101";
70              student_id <= "1000"; --d7
71           when s6 => current_state <= "0110";
72              student_id <= "0110"; --d8
73           when s7 => current_state <= "0111";
74              student_id <= "0101"; --d9
75        end case;
76     end process;
77  end fsm;
```

4-to-16 Decoder

Screenshot of Decoder Block Diagram

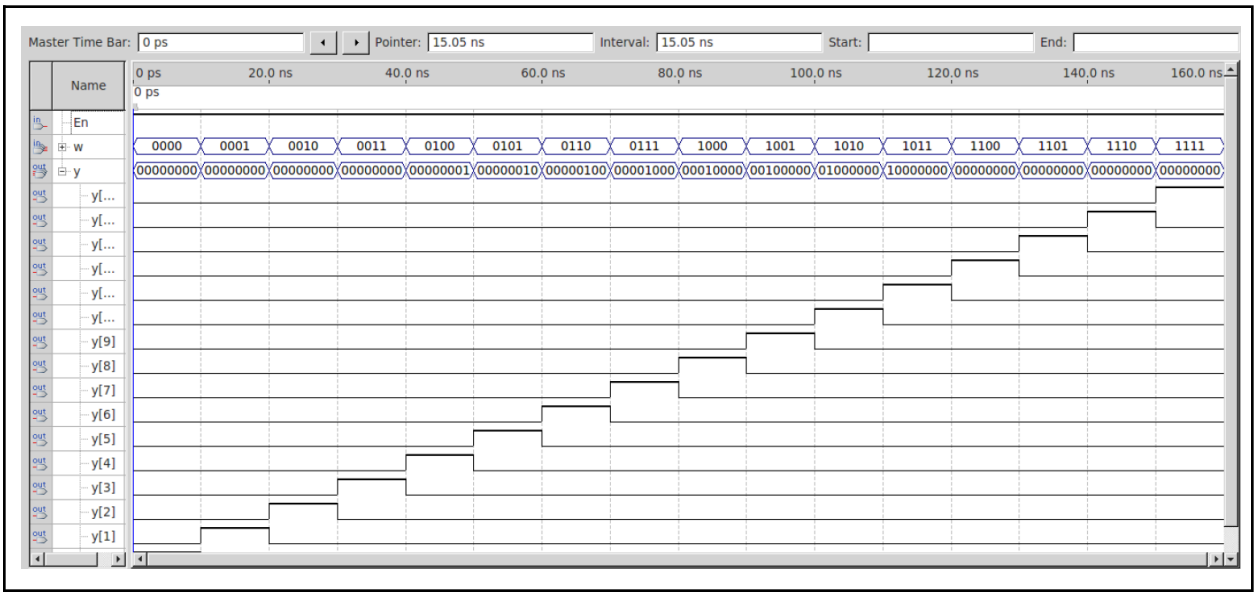


Decoder Truth Table

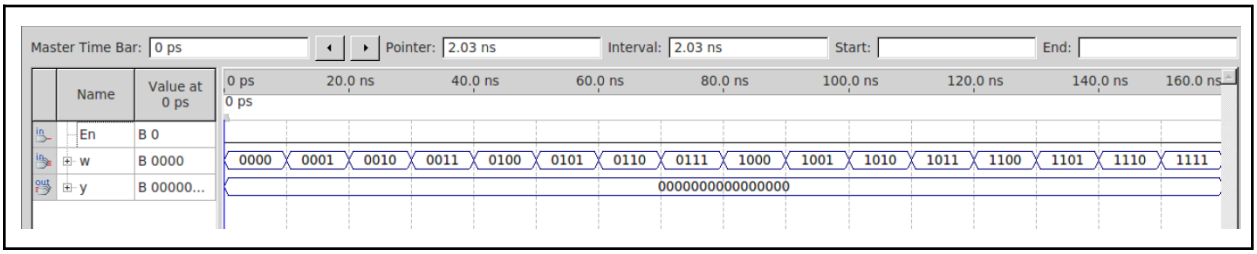
4-to-16 Decoder Truth Table		
Enable = 1		
FSM Current State	Input From FSM (4-bit Binary)	Output (16-bit Binary)
0	0000	0000000000000001
1	0001	0000000000000010
2	0010	0000000000000100
3	0011	0000000000001000
4	0100	0000000000010000
5	0101	0000000000100000
6	0110	0000000001000000
7	0111	0000000010000000
8	dddd	dddddddddddddddd
9	dddd	dddddddddddddddd
…	…	…
15	dddd	dddddddddddddddd

4-to-16 Decoder Waveform

Enable = 1



Enable = 0



As can be seen from the waveforms above, the correct opcode is output to the ALU when the Enable = 1. When Enable = 0, the output ‘0000000000000000’ is output, which means nothing happens in our system.

4-to-16 Decoder VHDL Code

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

ENTITY decoder IS
   PORT ( w  : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          En : IN STD_LOGIC ;
          y  : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
   END decoder;

ARCHITECTURE Behavior OF decoder IS
BEGIN
   PROCESS (w)
      BEGIN
         if (En = '0') then
            y <= "0000000000000000";
         else
            case w is
               WHEN "0000" => y <= "0000000000000001";
               WHEN "0001" => y <= "0000000000000010";
               WHEN "0010" => y <= "0000000000000100";
               WHEN "0011" => y <= "0000000000001000";
               WHEN "0100" => y <= "0000000000010000";
               WHEN "0101" => y <= "0000000000100000";
                WHEN "0110" => y <= "0000000001000000";
                WHEN "0111" => y <= "0000000010000000";
                WHEN "1000" => y <= "0000000100000000";
                WHEN "1001" => y <= "0000001000000000";
                WHEN "1010" => y <= "0000010000000000";
                WHEN "1011" => y <= "0000100000000000";
                WHEN "1100" => y <= "0001000000000000";
                WHEN "1101" => y <= "0010000000000000";
                WHEN "1110" => y <= "0100000000000000";
                WHEN "1111" => y <= "1000000000000000";
                WHEN OTHERS => y <= "0000000000000000";
            END CASE;
         END IF;
   END PROCESS;
END Behavior;
```

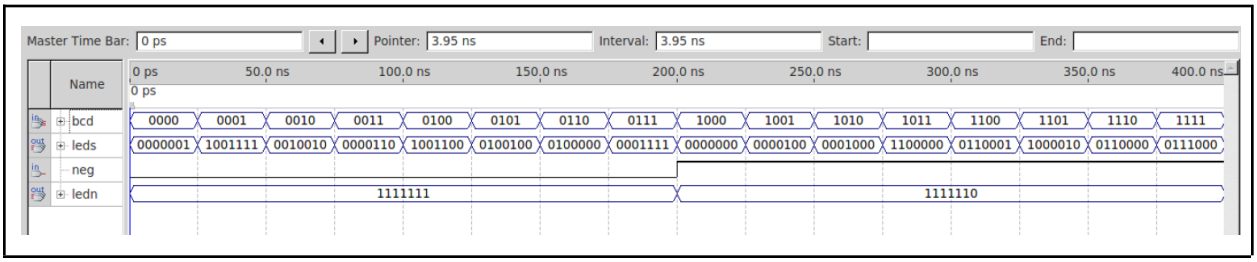
SSEG (for the Result)

Screenshot of SSEG (Result) Block Diagram

SSEG (Result) Truth Table

SSEG (Result) Truth Table		
Neg = 0		**Neg = 1**
Negative SSEG Display = 1111111		Negative SSEG Display = 1111110
BCD Input (4-bit Binary)	**BCD Input (Hexadecimal)**	**SSEG**
0000	0	0000001
0001	1	1001111
0010	2	0010010
0011	3	0000110
0100	4	1001100
0101	5	0100100
0110	6	0100000
0111	7	0001111
1000	8	0000000
1001	9	0000100
1010	A	0001000
1011	B	1100000
1100	C	0110001
1101	D	1000010
1110	E	0110000
1111	F	0111000

SSEG (Result) Waveform



From the waveform, it's seen that the SSEG outputs the correct hexadecimal number for each of the 4-bit binary inputs. It can also be seen that when the 'Neg' signal is 0, the SSEG outputs nothing on the display (represented by all 1s) and when 'Neg' is 1, the SSEG outputs a negative sign to the SSEG.

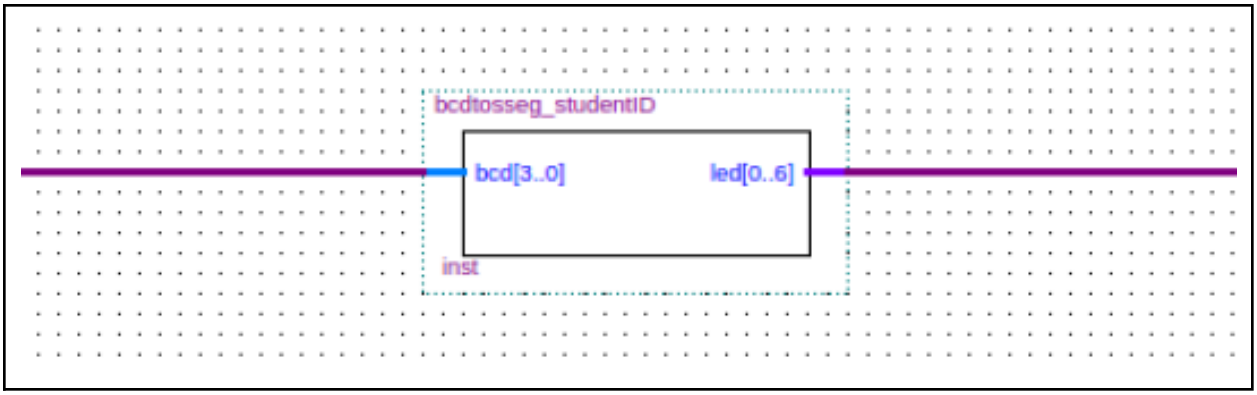
SSEG (Result) VHDL Code

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3
4   ENTITY bcdtosseg_result IS
5   PORT (    bcd : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
6             neg : IN STD_LOGIC;
7      leds, ledn : OUT STD_LOGIC_VECTOR(0 TO 6) );
8   END bcdtosseg_result;
9
10  ARCHITECTURE Behavior OF bcdtosseg_result IS
11  BEGIN
12  PROCESS (bcd, neg)
13
14  BEGIN
15
16  IF (neg = '1') then
17     ledn <= ("1111110");
18  ELSE
19     ledn <= ("1111111");
20  END IF;
21
22  CASE bcd IS            --abcdefg
23  WHEN "0000" => leds <= ("0000001");
24  WHEN "0001" => leds <= ("1001111");
25  WHEN "0010" => leds <= ("0010010");
26  WHEN "0011" => leds <= ("0000110");
27  WHEN "0100" => leds <= ("1001100");
28  WHEN "0101" => leds <= ("0100100");
29  WHEN "0110" => leds <= ("0100000");
30  WHEN "0111" => leds <= ("0001111");
31  WHEN "1000" => leds <= ("0000000");
32  WHEN "1001" => leds <= ("0000100");
33  WHEN "1010" => leds <= ("0001000");
34  WHEN "1011" => leds <= ("1100000");
35  WHEN "1100" => leds <= ("0110001");
36  WHEN "1101" => leds <= ("1000010");
37  WHEN "1110" => leds <= ("0110000");
38  WHEN "1111" => leds <= ("0111000");
39  WHEN OTHERS => leds <= ("-------");
40  END CASE;
41
42  END PROCESS;
43
44  END Behavior;
```

SSEG (Student ID)

This SSEG is almost identical to the Result SSEG; the difference is that there is no ‘Neg’ input and negative led output due to not needing a negative sign for displaying student ID digits. Also the BCD decimal inputs beyond 9 is not included due to student ID digits not going over that.

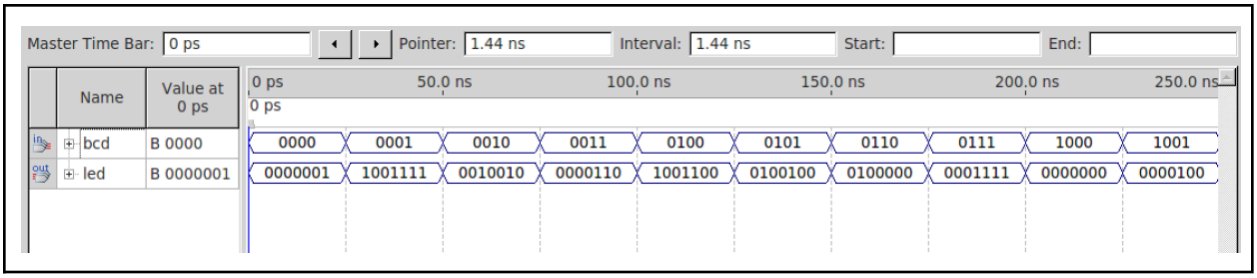
Screenshot of SSEG (Student ID) Block Diagram



SSEG (Student ID) Truth Table

SSEG (Student ID) Truth Table		
BCD Input (4-bit Binary)	**BCD Input (Decimal)**	**SSEG**
0000	0	0000001
0001	1	1001111
0010	2	0010010
0011	3	0000110
0100	4	1001100
0101	5	0100100
0110	6	0100000
0111	7	0001111
1000	8	0000000
1001	9	0000100
1010	d	ddddddd
1011	d	ddddddd
1100	d	ddddddd
1101	d	ddddddd
1110	d	ddddddd
1111	d	ddddddd

SSEG (Student ID) Waveform

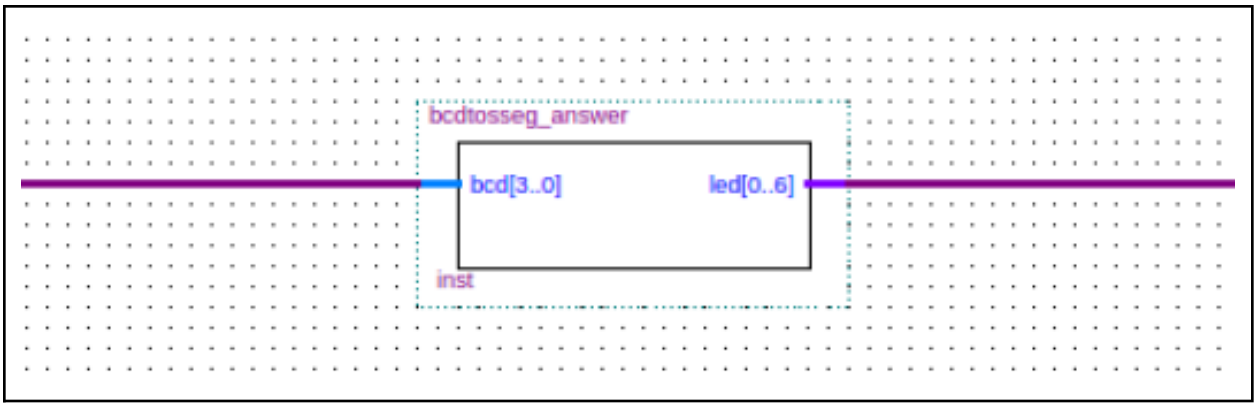


SSEG (Student ID) VHDL Code

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3
4   ENTITY bcdtosseg_studentID IS
5   PORT ( bcd : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
6   led : OUT STD_LOGIC_VECTOR(0 TO 6) );
7   END bcdtosseg_studentID;
8
9   ARCHITECTURE Behavior OF bcdtosseg_studentID IS
10  BEGIN
11  PROCESS (bcd)
12
13  BEGIN
14
15  CASE bcd IS            --abcdefg
16  WHEN "0000" => led <= ("0000001");
17  WHEN "0001" => led <= ("1001111");
18  WHEN "0010" => led <= ("0010010");
19  WHEN "0011" => led <= ("0000110");
20  WHEN "0100" => led <= ("1001100");
21  WHEN "0101" => led <= ("0100100");
22  WHEN "0110" => led <= ("0100000");
23  WHEN "0111" => led <= ("0001111");
24  WHEN "1000" => led <= ("0000000");
25  WHEN "1001" => led <= ("0000100");
26  WHEN OTHERS => led <= ("-------");
27  END CASE;
28
29  END PROCESS;
30
31  END Behavior;
```

SSEG (Answer) (for ALU 3 only)

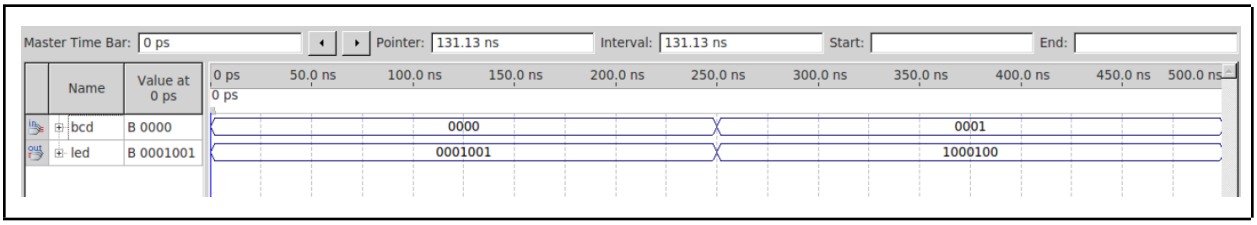
Screenshot of SSEG (Answer) Block Diagram



SSEG (Answer) Truth Table

SSEG (for ALU3) Truth Table		
Input	**SSEG (‘0’ means ON)**	**Yes or No**
0000	0001001	n
0001	1000100	y

SSEG (Answer) Waveform



From the waveform, it is observed that the display outputs ‘n’ when 0000 is read by the SSEG and ‘y’ is output when 0001 is read by the SSEG.

SSEG (Answer) VHDL Code

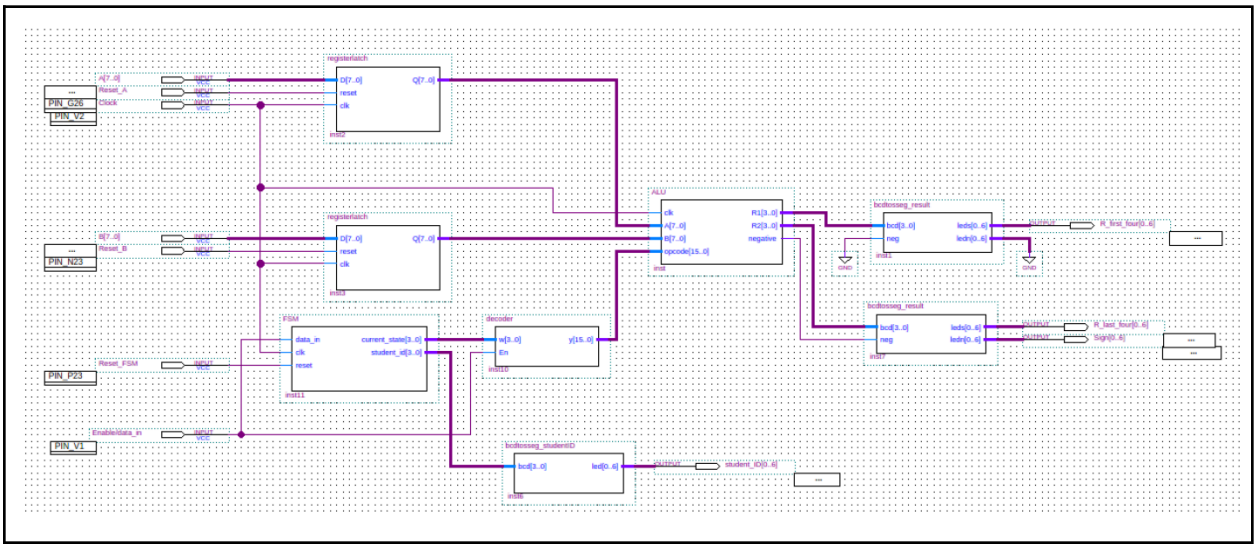
```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3
4   ENTITY bcdtosseg_answer IS
5   PORT ( bcd : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
6   led : OUT STD_LOGIC_VECTOR(0 TO 6) );
7   END bcdtosseg_answer;
8
9   ARCHITECTURE Behavior OF bcdtosseg_answer IS
10  BEGIN
11  PROCESS (bcd)
12
13  BEGIN
14
15  CASE bcd IS            --abcdefg
16  WHEN "0000" => led <= ("0001001"); -- no
17  WHEN "0001" => led <= ("1000100"); -- yes
18  WHEN OTHERS => led <= ("-------"); -- do nothing
19  END CASE;
20
21  END PROCESS;
22
23  END Behavior;
```

ALU1 Problem Set 1

Description

The purpose of ALU1 is to perform basic arithmetic and Boolean operations given 8-bit inputs A and B from the two storage latches. The ALU1 core will receive the opcode instruction of what operation to do at each rising edge of the clock signal. Once received, the action will be performed using both inputs or either input, depending on the operation at hand. The result of the operation will be an 8-bit output. Since the seven segment displays can only take a 4-bit binary input, the result will be split into two 4-bit results (R2 and R1) which will be split across two seven segment displays. For this ALU, there will be a total of 3 SSEGs. One SSEG will display the result R1 which is the first four MSBs (Most Significant Bits) of the result. Another SSEG will display the result R2 which is the four LSBs (Least Significant Bits) of the result. The final SSEG will display a negative sign if the computation of an operation results in a negative result (namely function 2). Each of the 8 microcodes gives instructions to the ALU to perform a different function. This information is located in the table below. As mentioned previously every component of the CPU is connected to the same clock. As the FSM up-counts through the different states and corresponding student ID digits, the ALU1 will cycle through performing the different functions supplied by the opcode. Once all the functions have been performed and the FSM cycles back to loop the sequence, the ALU will also loop from the beginning to do the operations again.

Screenshot of ALU1 Block Schematic File (BDF)



Purpose of Inputs and Outputs for ALU1

The ALU1 takes the inputs A and B as they are what will be used to perform the operations. The clock's purpose is to let the FSM know when to perform the given operation. The opcode input will inform the ALU of the current operation to be completed. Regarding the outputs of the ALU, R2 and R1 are the split outputs of the 8-bit result from the computation. The negative sign output sends data to the negative sign SSEG regarding whether a negative sign is required for the computation that was done.

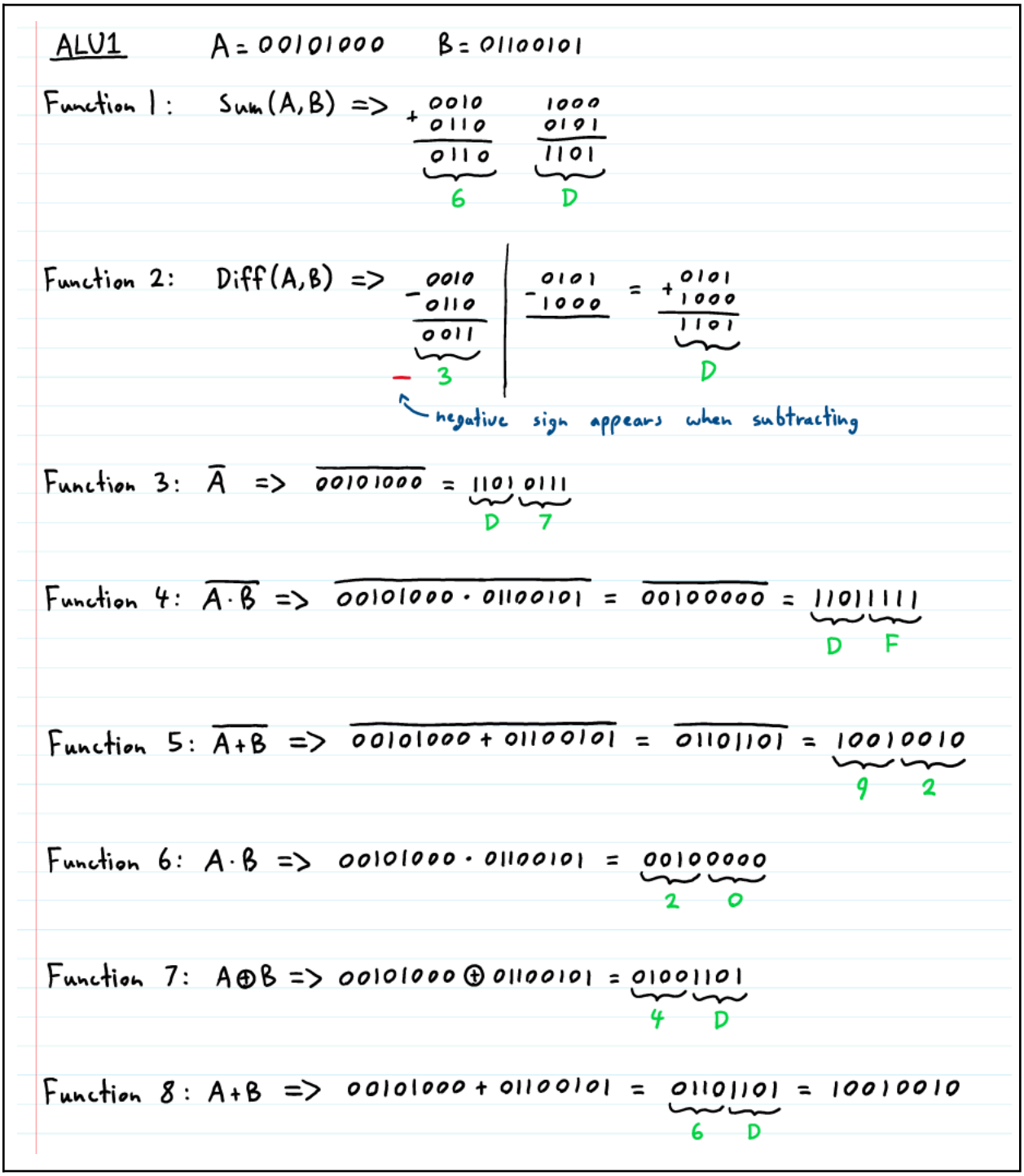
Table of Microcodes Generated by the Decoder for ALU1

Function #	Microcode (from 4-to-16 Decoder)	Operation
1	0000000000000001	Sum(A,B)
2	0000000000000010	Diff(A,B)
3	0000000000000100	$\overline{A}$
4	0000000000001010	$\overline{A \cdot B}$
5	0000000000010000	$\overline{A + B}$
6	0000000000100000	$A \cdot B$
7	0000000001000000	$A \oplus B$
8	0000000010000000	$A + B$

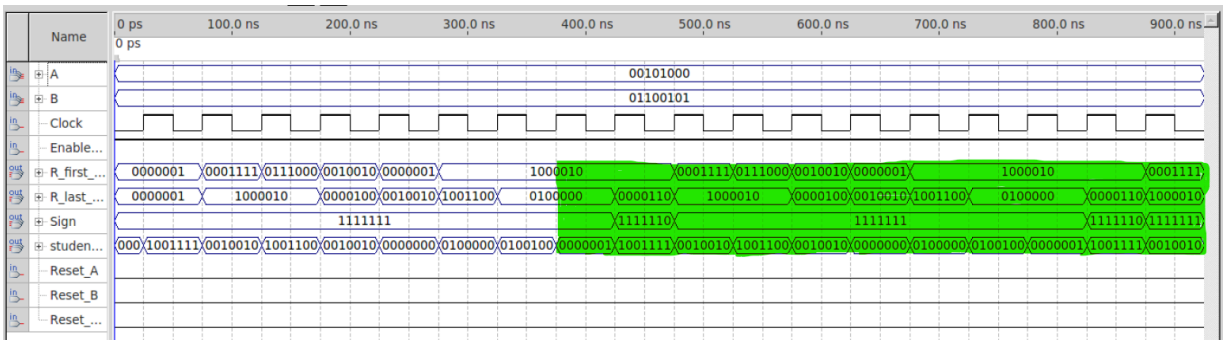
ALU1 Complete Truth Table for the Various Operations

ALU1 Truth Table								
Input A				Input B				
Decimal		Binary (8-bit)		Decimal		Binary (8-bit)		
28		00101000		65		01100101		
ALU Process	Results (4-bit Binary)		Results (Hexadecimal)		Results SSEG Display		Negative Sign	
Function #	R2	R1	R2	R1	SSEG (R2)	SSEG (R1)	Yes or No?	SSEG
1	0110	1101	6	D	0100000	1000010	No	1111111
2	0011	1101	3	D	0000110	1000010	Yes	1111110
3	1101	0111	D	7	1000010	0001111	No	1111111
4	1101	1111	D	F	1000010	0111000	No	1111111
5	1001	0010	9	2	0000100	0010010	No	1111111
6	0010	0000	2	0	0010010	0000001	No	1111111
7	0100	1101	4	D	1001100	1000010	No	1111111
8	0110	1101	6	D	0100000	1000010	No	1111111

Handwritten Work for ALU1 Operations



Screenshot of Complete Waveform for ALU1



Note: The output is correct in the highlighted region. The highlighted region includes cycled states and their outputs as well. All the outputs are in terms of the 7-bit binary for the SSEG display. ‘0’ means the segment is on and ‘1’ means the segment is off.

ALU1 VHDL Code

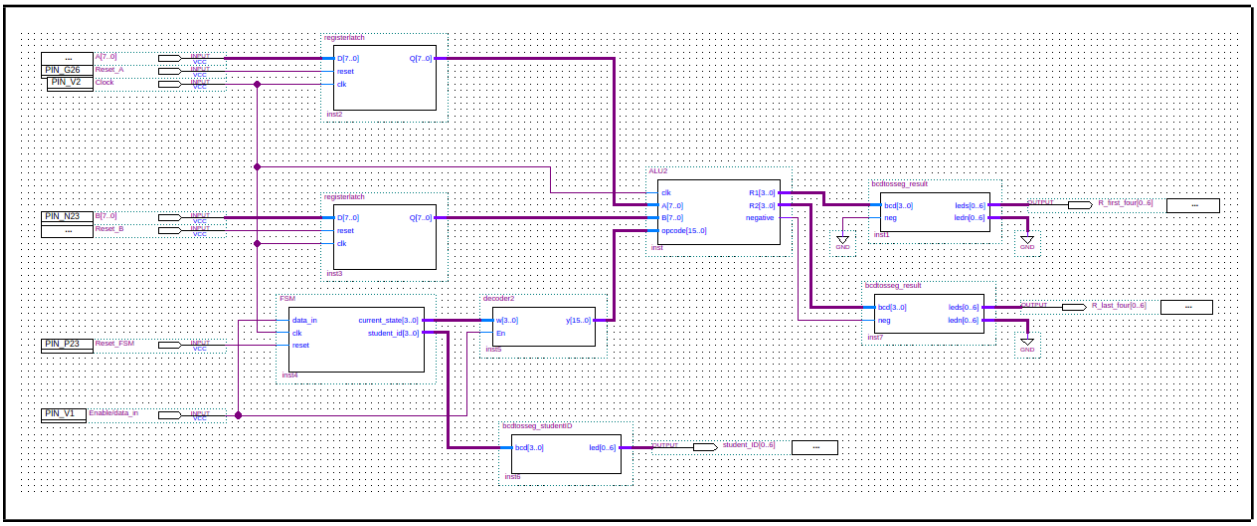
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.all;

ENTITY ALU IS
PORT(          clk : in STD_LOGIC;
               A,B : in STD_LOGIC_VECTOR(7 DOWNTO 0); -- 8-bit inputs A & B from Reg. 1 & Reg. 2
            opcode : in STD_LOGIC_VECTOR(15 DOWNTO 0); -- 8-bit opcode from Decoder
            R1, R2 : out STD_LOGIC_VECTOR(3 DOWNTO 0); -- two 4-bit results
            negative : out STD_LOGIC);
END ALU;

ARCHITECTURE calculation OF ALU IS
SIGNAL Reg1, Reg2, Result : STD_LOGIC_VECTOR(7 DOWNTO 0) := (OTHERS => '0');

BEGIN
Reg1 <= A;
Reg2 <= B;

PROCESS(clk,opcode)
BEGIN
   if(rising_edge(clk)) then
      case opcode is

      WHEN "0000000000000001" =>  -- Addition of both numbers
      Result <= Reg1 + Reg2;
      negative <= '0';

      WHEN "0000000000000010" =>  -- Subtraction of both numbers
      if (Reg2 > Reg1) then -- Accounting for a negative result with a negative sign
         Result <= Reg2 - Reg1;
         negative <= '1';
      else                  -- Accounting for non-negative result with no negative sign
         Result <= Reg1 - Reg2;
         negative <= '0';
      end if;

      WHEN "0000000000000100" =>  -- Inverting A operation
      Result <= NOT(Reg1);
      negative <= '0';

      WHEN "0000000000001000" =>  -- A NAND B operation
      Result <= Reg1 NAND Reg2;
      negative <= '0';

      WHEN "0000000000010000" =>  -- A NOR B operation
      Result <= Reg1 NOR Reg2;
      negative <= '0';

      WHEN "0000000000100000" =>  -- A AND B operation
      Result <= Reg1 AND Reg2;
      negative <= '0';

      WHEN "0000000001000000" =>  -- A XOR B operation
      Result <= Reg1 XOR Reg2;
      negative <= '0';

      WHEN "0000000010000000" =>  -- A OR B operation
      Result <= Reg1 OR Reg2;
      negative <= '0';

      WHEN OTHERS => --don't care, do nothing

      end case;
   end if;
END PROCESS;
R1 <= Result(3 DOWNTO 0); --Result split into two 4-bit outputs and displayed on 2 pairs of seven segment displays
R2 <= Result(7 DOWNTO 4);
END calculation;
```

ALU2 Problem Set 2

Description

ALU2 will work very similarly to ALU1. ALU1 was used to perform basic arithmetic and Boolean operations given 8-bit inputs A and B from the two storage latches. The ALU2 core, just like ALU1, will receive the opcode instruction of what operation to do at each rising edge of the clock signal. Once received, the action will be performed using both inputs or either input, depending on the operation at hand. The result of the operation will be an 8-bit output. Since the seven segment displays can only take a 4-bit binary input, the result will be split into two 4-bit results (R2 and R1) which will be split across two seven segment displays. For this ALU, there will be a total of 2 SSEGs. One SSEG will display the result R1 which is the first four MSBs (Most Significant Bits) of the result. Another SSEG will display the result R2 which is the four LSBs (Least Significant Bits) of the result. Each of the 8 microcodes gives instructions to the ALU to perform a different function. The set of microcodes is from Problem (d). This information is located in the table below. Every component of the CPU is connected to the same clock. As the FSM up-counts through the different states and corresponding student ID digits, the ALU2 will cycle through performing the different functions supplied by the opcode. Once all the functions have been performed and the FSM cycles back to loop the sequence, the ALU will also loop from the beginning to do the operations again.

Screenshot of ALU2 Block Schematic File (BDF)



Purpose of Inputs and Outputs for ALU2

The ALU2 takes the inputs A and B as they are what will be used to perform the operations. The clock's purpose is to let the FSM know when to perform the given operation. The opcode input will inform the ALU of the current operation to be completed. Regarding the outputs of the ALU, R2 and R1 are the split outputs of the 8-bit result from the computation.

Table of Microcodes Generated by the Decoder for ALU2

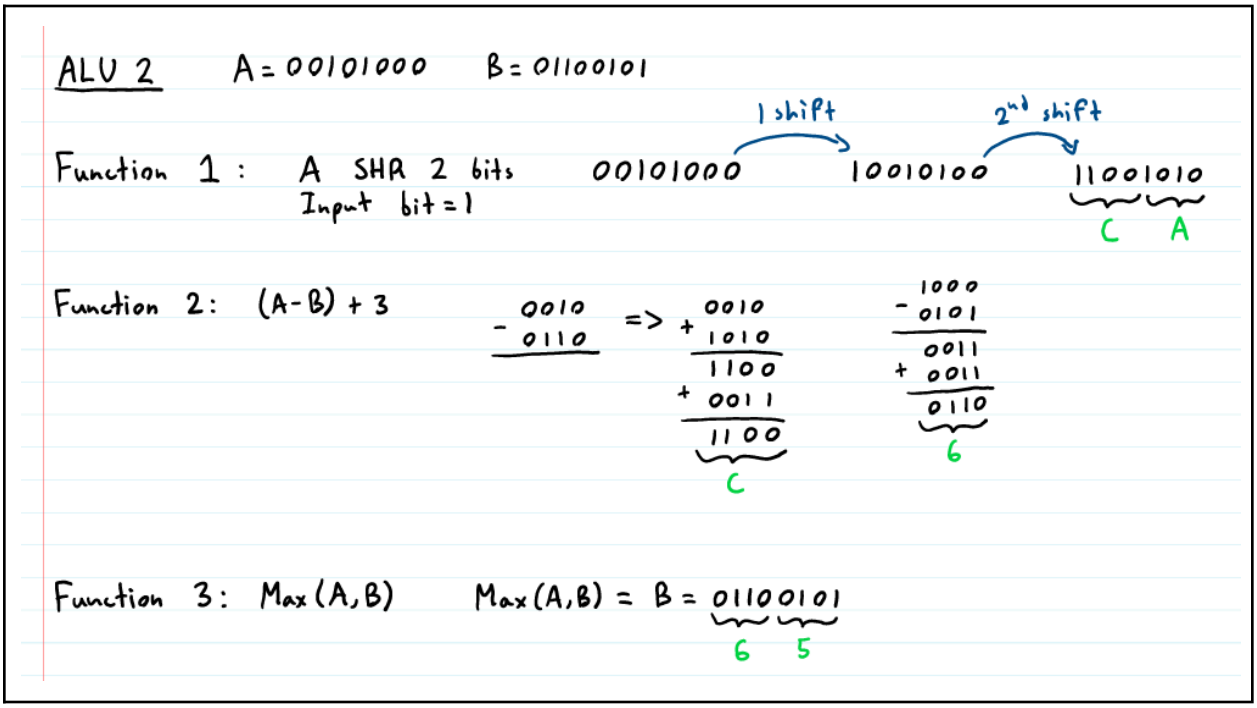
Problem Set: d)

Function #	Microcode (from 4-to-16 Decoder)	Operation
1	0000000000000001	Shift **A** to right by two bits, input bit = 1 (SHR)
2	0000000000000010	Produce the difference of **A** and **B** and then increment by 3
3	0000000000000100	Find the greater value of **A** and **B** and produce the results (Max(**A**,**B**))
4	0000000000001010	Swap the upper 4 bits of **A** by the lower 4 bits of **B**
5	0000000000010000	Increment **A** by 1
6	0000000000100000	Produce the result of ANDing **A** and **B**
7	0000000001000000	Invert the upper four bits of **A**
8	0000000010000000	Rotate **B** to left by 3 bits (ROL)

ALU2 Complete Truth Table for the Various Operations

ALU2 Truth Table						
Input A			**Input B**			
Decimal	**Binary (8-bit)**		**Decimal**	**Binary (8-bit)**		
28	00101000		65	01100101		
ALU Operation	**Results (4-bit Binary)**		**Results (Hexadecimal)**		**SSEG Display**	
Function #	**R2**	**R1**	**R2**	**R1**	**SSEG (R2)**	**SSEG (R1)**
1	1100	1010	C	A	0110001	0001000
2	1100	0110	C	6	0110001	0100000
3	0110	0101	6	5	0100000	0100100
4	0101	1000	5	8	0100100	0000000
5	0010	1001	2	9	0010010	0000100
6	0010	0000	2	0	0010010	0000001
7	1101	1000	D	8	1000010	0000000
8	0010	1011	2	B	0010010	1100000

Handwritten Work for ALU2 Operations



Function 4: Swap upper 4 bits of A with lower 4 bits of B

A = 00101000 B = 01100101

= 01011000

5 8

Function 5: Increment A by 1 :

```
  00101000
+ 00000001
----------
  00101001
```

2 9

Function 6: A · B => 00101000 · 01100101 = 00100000

2 0

Function 7: Invert upper 4 bits of A : $\overline{0010}1000$ = 11011000

D 8

Function 8: Rotate B left 3 bits: 01100101

1 rotation

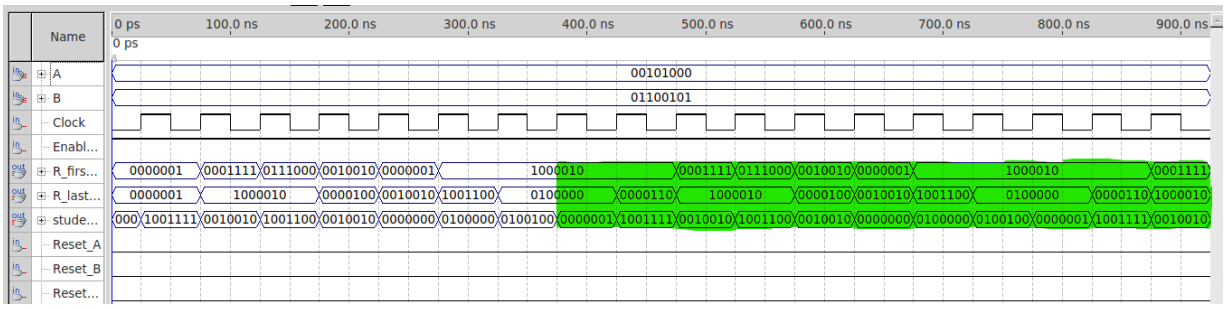
11001010

10010101

00101011

2 B

Screenshot of Complete Waveform for ALU2



Note: The output is correct in the highlighted region. The highlighted region includes cycled states and their outputs as well. All the outputs are in terms of the 7-bit binary for the SSEG display. '0' means the segment is on and '1' means the segment is off.

ALU2 VHDL Code

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.std_logic_unsigned.all;
4   USE ieee.numeric_std.all;
5
6   ENTITY ALU2 IS
7   PORT(           clk : in  STD_LOGIC;
8                   A,B : in  UNSIGNED(7 DOWNTO 0); -- 8-bit inputs A & B from Reg. 1 & Reg. 2
9              opcode : in  UNSIGNED(15 DOWNTO 0); -- 8-bit opcode from Decoder
10             R1, R2 : out UNSIGNED(3 DOWNTO 0); -- two 4-bit results
11          negative : out STD_LOGIC);
12  END ALU2;
13
14  ARCHITECTURE calculation OF ALU2 IS
15  SIGNAL Reg1, Reg2, Result : UNSIGNED(7 DOWNTO 0) := (OTHERS => '0');
16
17  BEGIN
18  Reg1 <= A;
19  Reg2 <= B;
20
21  PROCESS(clk,opcode)
22  BEGIN
23        case opcode is
24
25        WHEN "0000000000000001" =>
26        Result <= Reg1 srl 2; -- Shift A right 2 bits
27        Result(6) <= '1'; --Input bit 1
28        Result(7) <= '1';
29
30
31        WHEN "0000000000000010" =>  -- Subtraction of both numbers and increment by 3
32        Result <= (Reg1 - Reg2) + "00000011";
33
34        WHEN "0000000000000100" =>  -- Output the maximum of the 2 numbers;
```

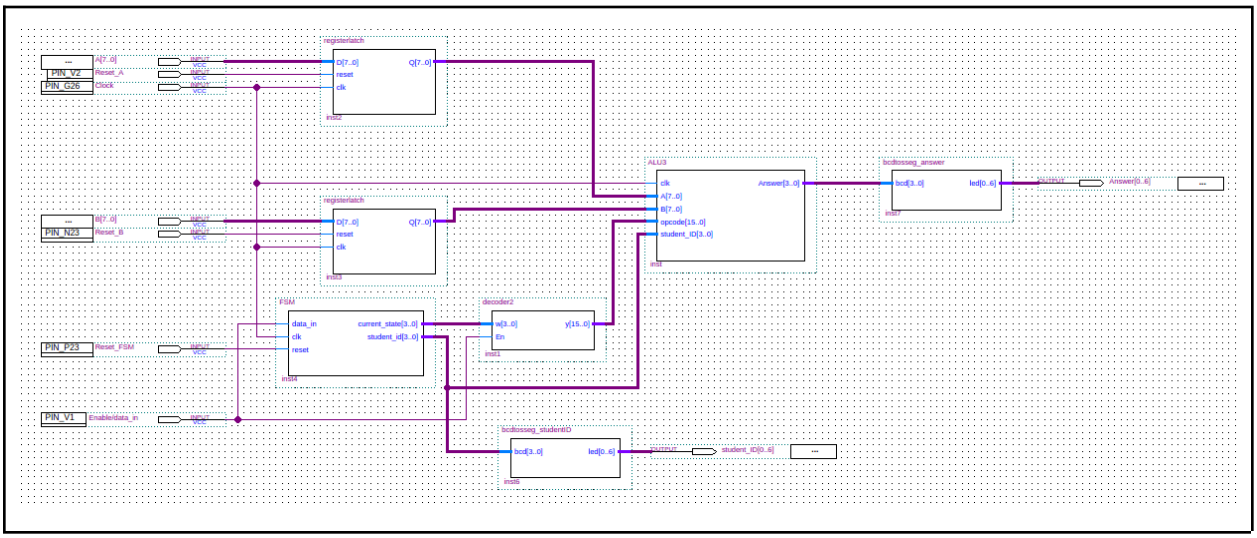
```
35          if(Reg1 > Reg2) then
36             Result <= Reg1;
37          else
38             Result <= Reg2;
39          end if;
40
41          WHEN "0000000000001000" =>  -- Swap upper 4 bits of A with lower 4 bits of B
42          Result(0) <= Reg1(0);
43          Result(1) <= Reg1(1);
44          Result(2) <= Reg1(2);
45          Result(3) <= Reg1(3);
46          Result(4) <= Reg2(0);
47          Result(5) <= Reg2(1);
48          Result(6) <= Reg2(2);
49          Result(7) <= Reg2(3);
50
51          WHEN "0000000000010000" =>  -- Increment A by 1
52          Result <= Reg1 + "00000001";
53
54          WHEN "0000000000100000" =>  -- A AND B operation
55          Result <= Reg1 AND Reg2;
56          negative <= '0';
57
58          WHEN "0000000001000000" =>  -- Invert upper four bits of A
59          Result(0) <= Reg1(0);
60          Result(1) <= Reg1(1);
61          Result(2) <= Reg1(2);
62          Result(3) <= Reg1(3);
63          Result(4) <= NOT Reg1(4);
64          Result(5) <= NOT Reg1(5);
65          Result(6) <= NOT Reg1(6);
66          Result(7) <= NOT Reg1(7);
67
68          WHEN "0000000010000000" =>  -- Rotate B left 3 bits
69          Result <= Reg2 ROL 3;
70
71          WHEN OTHERS => --don't care, do nothing
72
73          end case;
74    END PROCESS;
75    R1 <= Result(3 DOWNTO 0); --Result split into two 4-bit outputs and displayed on 2 pairs of seven segment displays
76    R2 <= Result(7 DOWNTO 4);
77    END calculation;
```

ALU3 Problem Set 3

Description

ALU3 will work slightly differently to the other ALUs. In this case, the only useful inputs to the ALU3 is the opcode from the control unit and the student ID digits from the FSM. The ALU3 core will receive the opcode instruction of what operation to do at each rising edge of the clock signal. According to Problem Set (c), the ALU will check the current student ID digit for the current state. If the digit has odd parity in 4-bit binary, the SSEG will display the letter ‘y’ for yes. If the digit has even parity, the SSEG will display the letter ‘n’ for no. Since there is just the need for one letter output at a time, there will only be one SSEG for ALU3 which will be modified appropriately to display ‘y’ or ‘n’. Every component of the CPU is connected to the same clock. As the FSM up-counts through the different states and corresponding student ID digits, the ALU3 will cycle through performing the different functions supplied by the opcode. Once all the functions have been performed and the FSM cycles back to loop the sequence, the ALU will also loop from the beginning to do the operations again.

Screenshot of ALU3 Block Schematic File (BDF)



Purpose of Inputs and Outputs for ALU3

The ALU3 takes the student ID input from the FSM. The clock's purpose is to let the FSM know when to perform the given operation. The opcode input will inform the ALU of the current operation to be completed. Regarding the output, a binary code will be sent to the SSEG and depending on the ALU's operation, the letter 'y' or the letter 'n' will be displayed by the SSEG.

Table of Microcodes Generated by the Decoder for ALU3

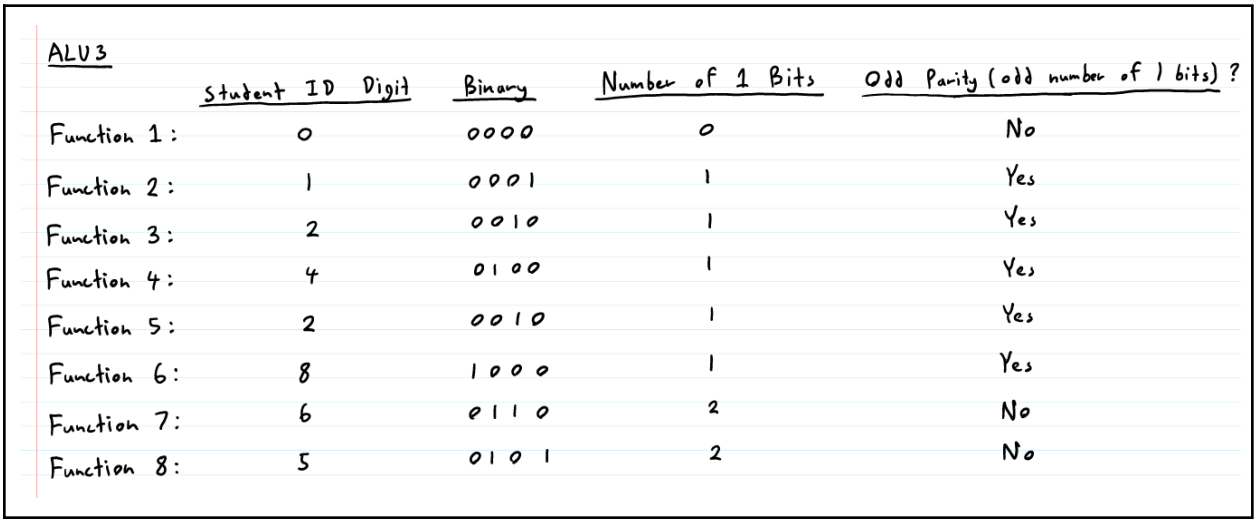
Problem Set: c): *For each opcode submitted to the ALU, display 'y' if the student_id signal has an odd parity and 'n' otherwise.*

Function #	Microcode (from 4-to-16 Decoder)	Operation
1	0000000000000001	Checks if the student ID's 2nd digit has odd parity.
2	0000000000000010	Checks if the student ID's 3rd digit has odd parity.
3	0000000000000100	Checks if the student ID's 4th digit has odd parity.
4	0000000000001010	Checks if the student ID's 5th digit has odd parity.
5	0000000000010000	Checks if the student ID's 6th digit has odd parity.
6	0000000000100000	Checks if the student ID's 7th digit has odd parity.
7	0000000001000000	Checks if the student ID's 8th digit has odd parity.
8	0000000010000000	Checks if the student ID's 9th digit has odd parity.

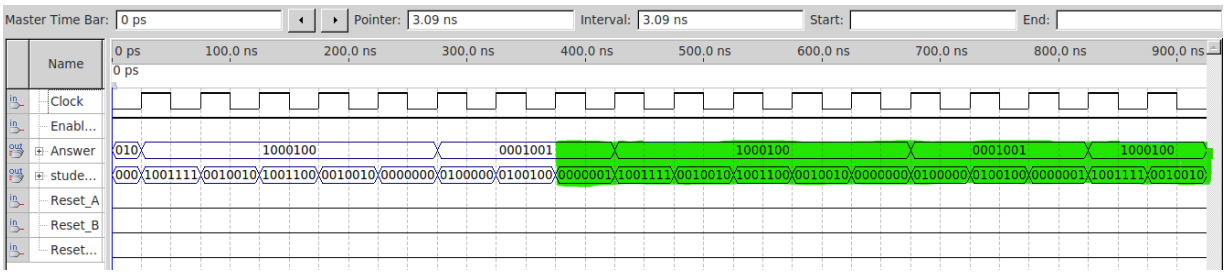
ALU3 Complete Truth Table for the Various Operations

ALU3 Truth Table					
Student ID (BCD Format)	**Function #**	**Student ID # (Decimal)**	**Student ID # (4-bit Binary)**	**Odd Parity**	
				(Yes or No)	**SSEG**
0000001	1	0	0000	No	0001001
1001111	2	1	0001	Yes	1000100
0010010	3	2	0010	Yes	1000100
1001100	4	4	0100	Yes	1000100
0010010	5	2	0010	Yes	1000100
0000000	6	8	1000	Yes	1000100
0100000	7	6	0110	No	0001001
0100100	8	5	0101	No	0001001

Handwritten Work for ALU3 Operations



Screenshot of Complete Waveform for ALU3



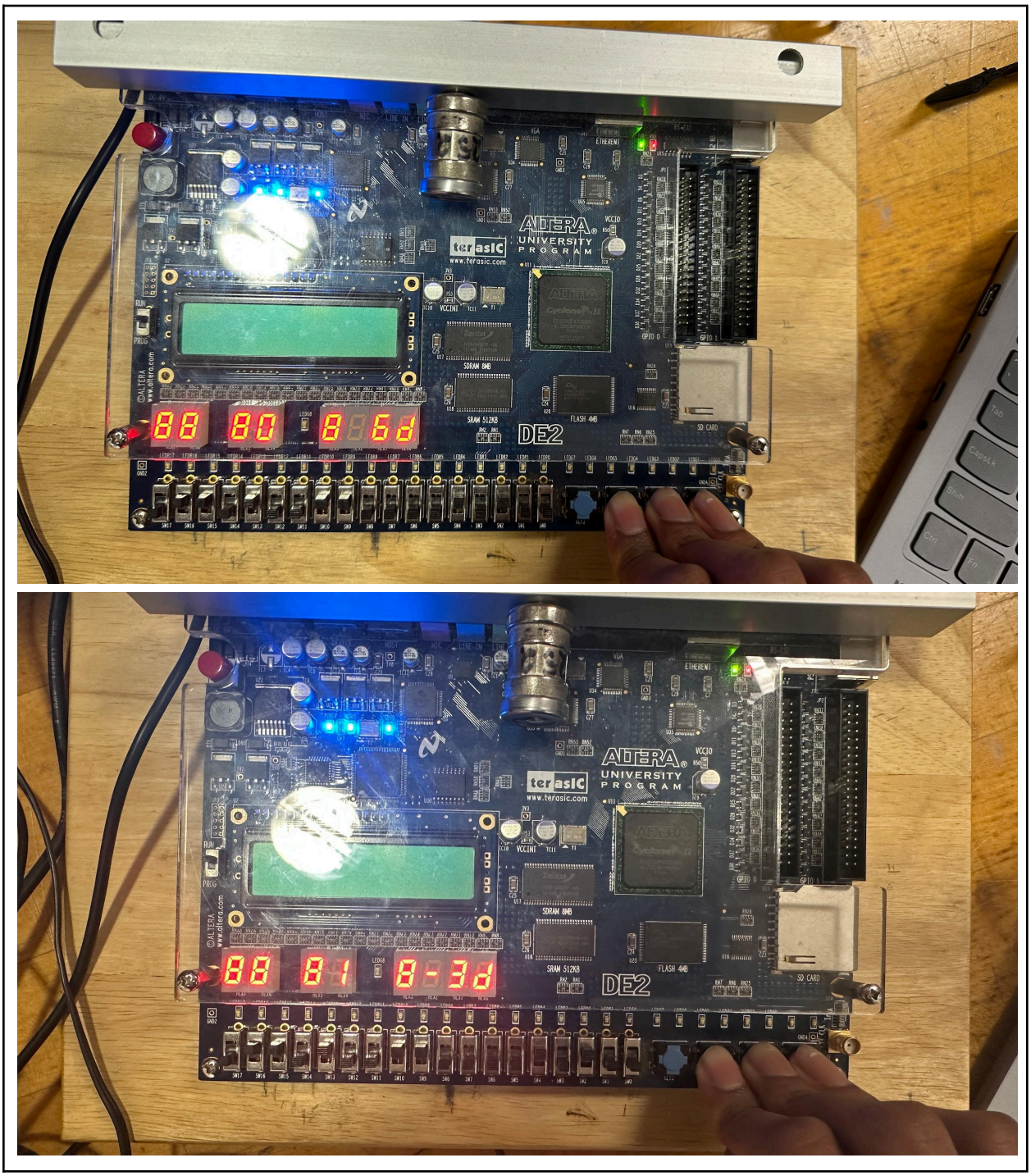
Note: The output is correct in the highlighted region. The highlighted region includes cycled states and their outputs as well. All the outputs are in terms of the 7-bit binary for the SSEG display. '0' means the segment is on and '1' means the segment is off.

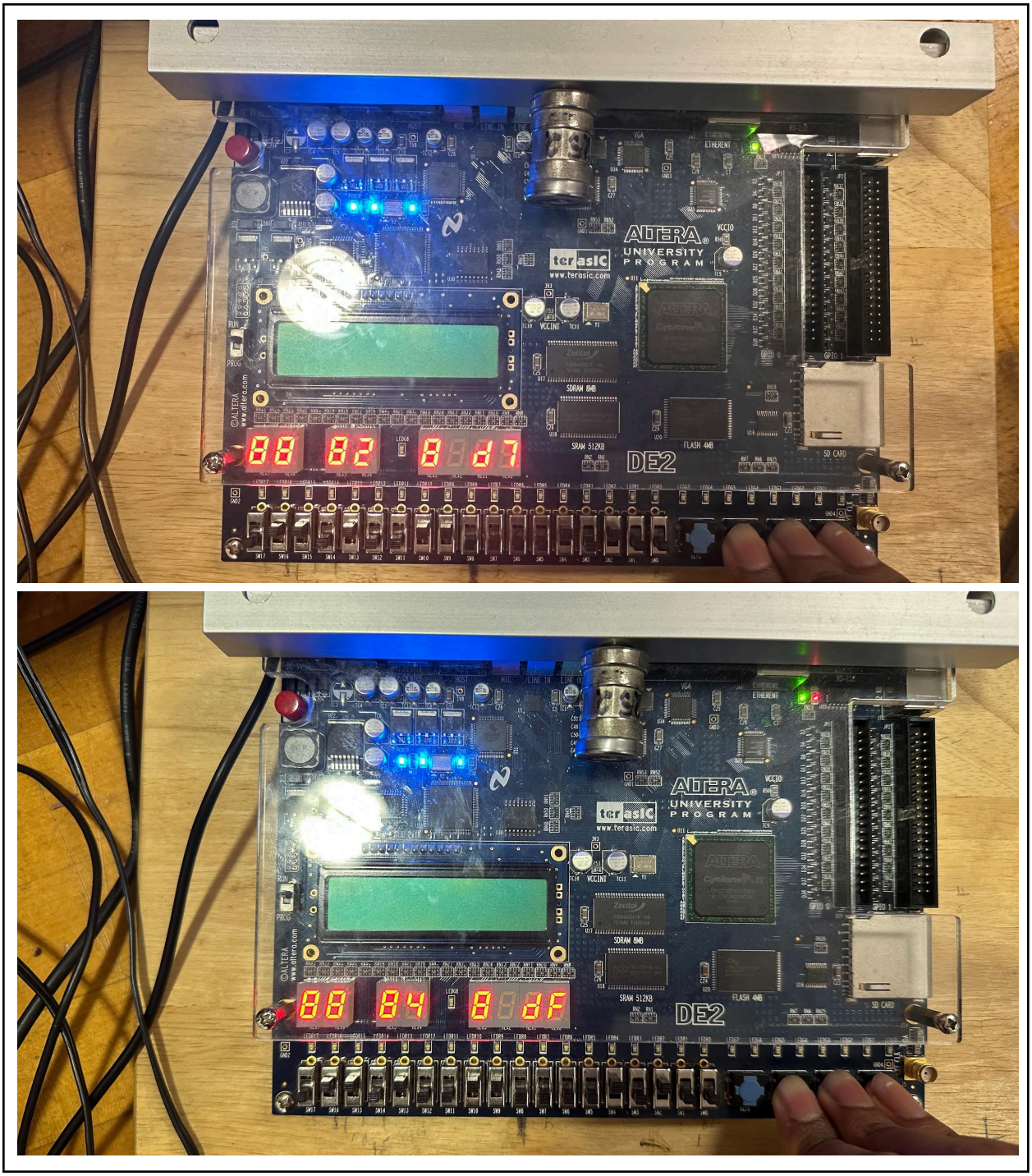
ALU3 VHDL Code

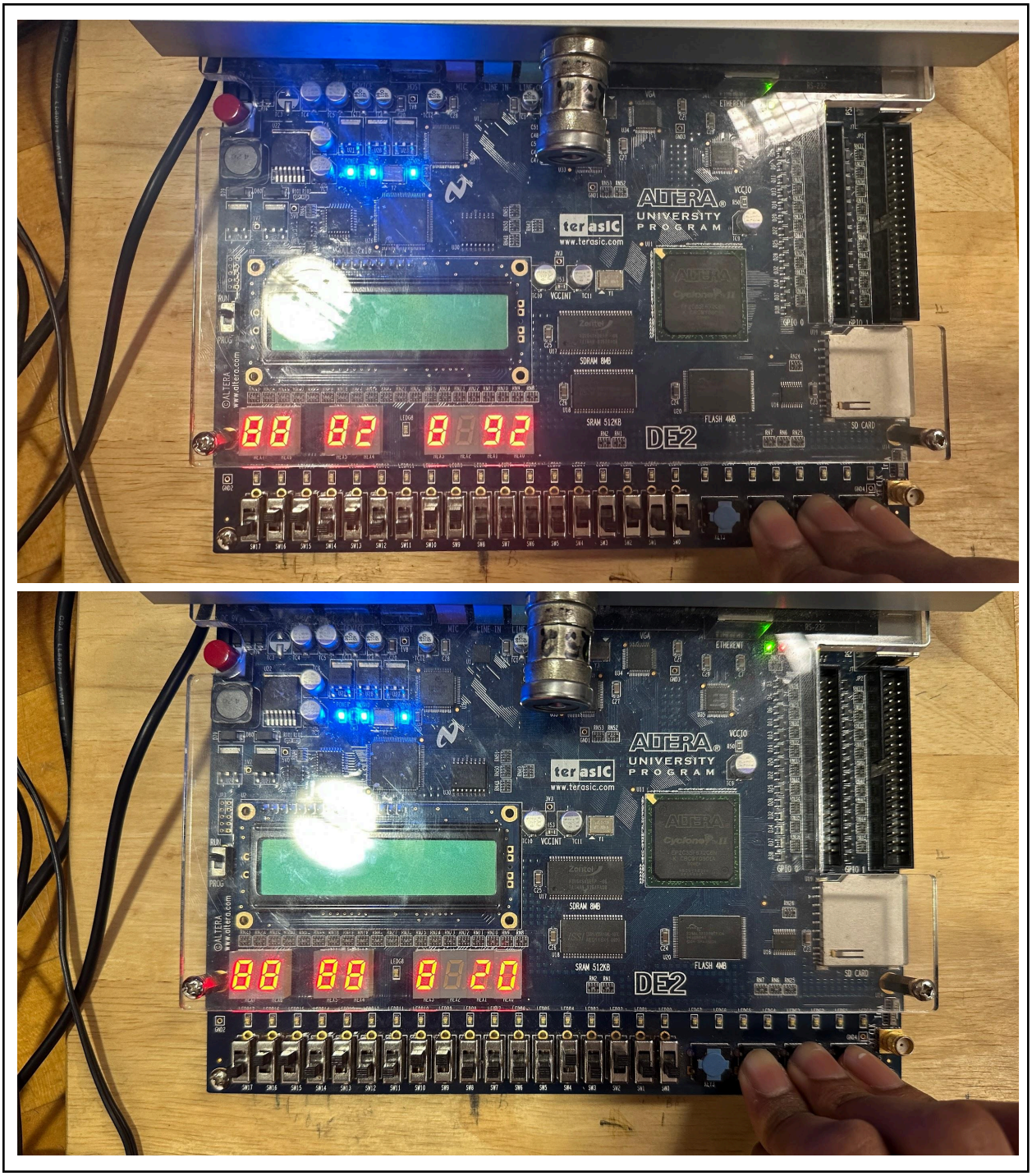
```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.numeric_std.all;
4   USE ieee.std_logic_unsigned.all;
5
6   ENTITY ALU3 IS
7   PORT(          clk : in STD_LOGIC;
8                  A,B : in STD_LOGIC_VECTOR(7 DOWNTO 0);
9              opcode : in  STD_LOGIC_VECTOR(15 DOWNTO 0); -- 8-bit opcode from Decoder
10         student_ID : in STD_LOGIC_VECTOR(3 DOWNTO 0);
11             Answer : out STD_LOGIC_VECTOR(3 DOWNTO 0));
12  END ALU3;
13
14  ARCHITECTURE calculation OF ALU3 IS
15
16  BEGIN
17
18  PROCESS(opcode)
19  BEGIN
20        case opcode is
21        -- Checking for odd parity; if odd parity, output 'y'; if even parity, output 'n'
22        WHEN "0000000000000001" =>
23        if (NOT(student_ID(3) XOR student_ID(2) XOR student_ID(1) XOR student_ID(0)) = '0') then
24           Answer <= "0001";
25        else
26           Answer <= "0000";
27        end if;
28
29        WHEN "0000000000000010" =>
30        if (NOT(student_ID(3) XOR student_ID(2) XOR student_ID(1) XOR student_ID(0)) = '0') then
31           Answer <= "0001";
```

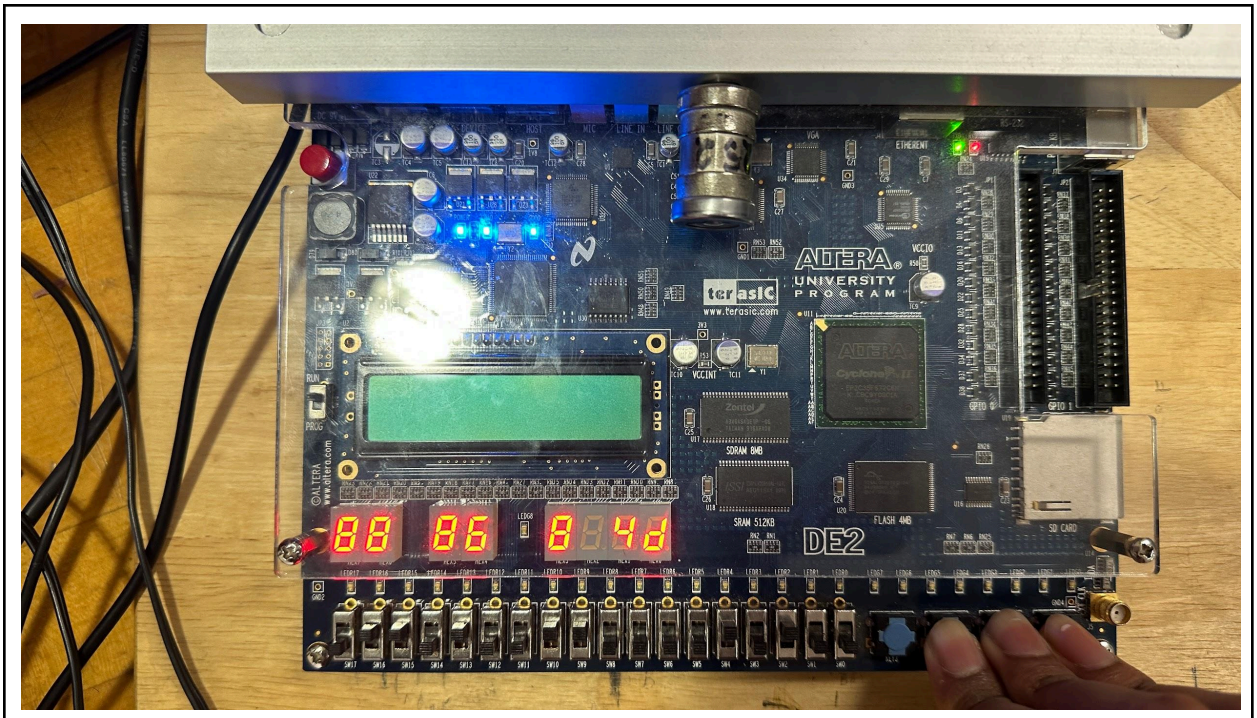
```
32          else
33             Answer <= "0000";
34          end if;
35
36          WHEN "0000000000000100" =>
37          if (NOT(student_ID(3) XOR student_ID(2) XOR student_ID(1) XOR student_ID(0)) = '0') then
38             Answer <= "0001";
39          else
40             Answer <= "0000";
41          end if;
42
43          WHEN "0000000000001000" =>
44          if (NOT(student_ID(3) XOR student_ID(2) XOR student_ID(1) XOR student_ID(0)) = '0') then
45             Answer <= "0001";
46          else
47             Answer <= "0000";
48          end if;
49
50          WHEN "0000000000010000" =>
51          if (NOT(student_ID(3) XOR student_ID(2) XOR student_ID(1) XOR student_ID(0)) = '0') then
52             Answer <= "0001";
53          else
54             Answer <= "0000";
55          end if;
56
57          WHEN "0000000000100000" =>
58          if (NOT(student_ID(3) XOR student_ID(2) XOR student_ID(1) XOR student_ID(0)) = '0') then
59             Answer <= "0001";
60          else
61             Answer <= "0000";
62          end if;
63
64          WHEN "0000000001000000" =>
65          if (NOT(student_ID(3) XOR student_ID(2) XOR student_ID(1) XOR student_ID(0)) = '0') then
66             Answer <= "0001";
67          else
68             Answer <= "0000";
69          end if;
70
71          WHEN "0000000010000000" =>
72          if (NOT(student_ID(3) XOR student_ID(2) XOR student_ID(1) XOR student_ID(0)) = '0') then
73             Answer <= "0001";
74          else
75             Answer <= "0000";
76          end if;
77
78          WHEN OTHERS => --don't care, do nothing
79             Answer <= "----";
80
81          end case;
82    END PROCESS;
83    END calculation;
```

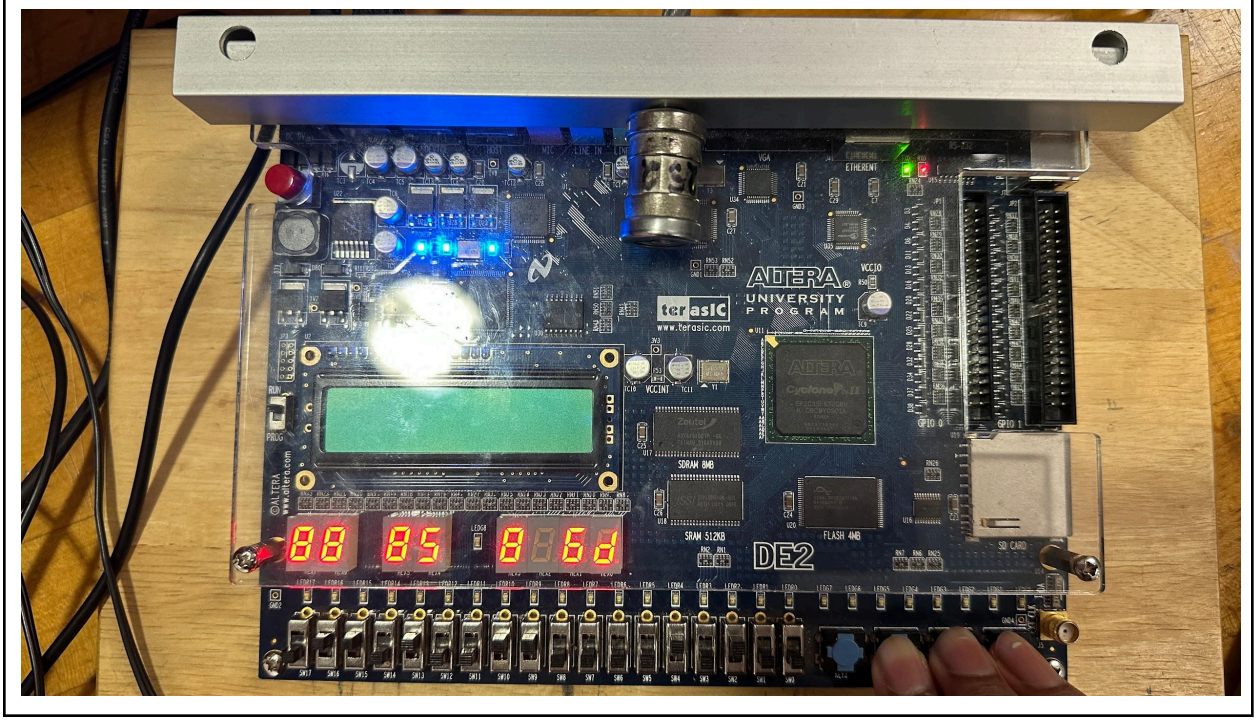
ALU1 Operation Pictures (One Full Sequence In Order)



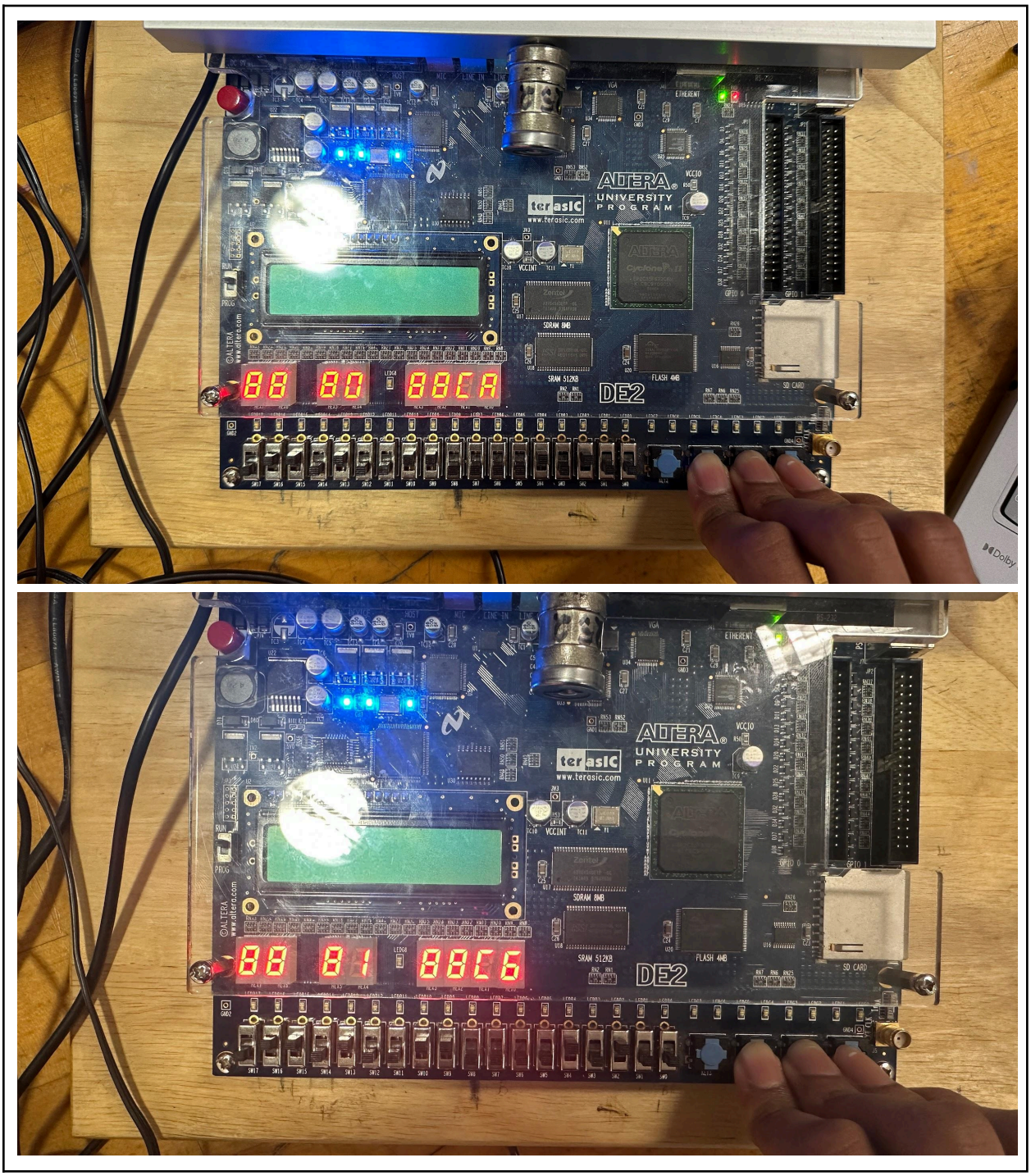

ALTERA
UNIVERSITY
PROGRAM
terasIC
www.terasic.com
DE2
SDRAM 8MB
SRAM 512KB
FLASH 4MB
SD CARD
ETHERNET
VCCIO
VCCINT
00 02 8 d7
00 04 8 dF

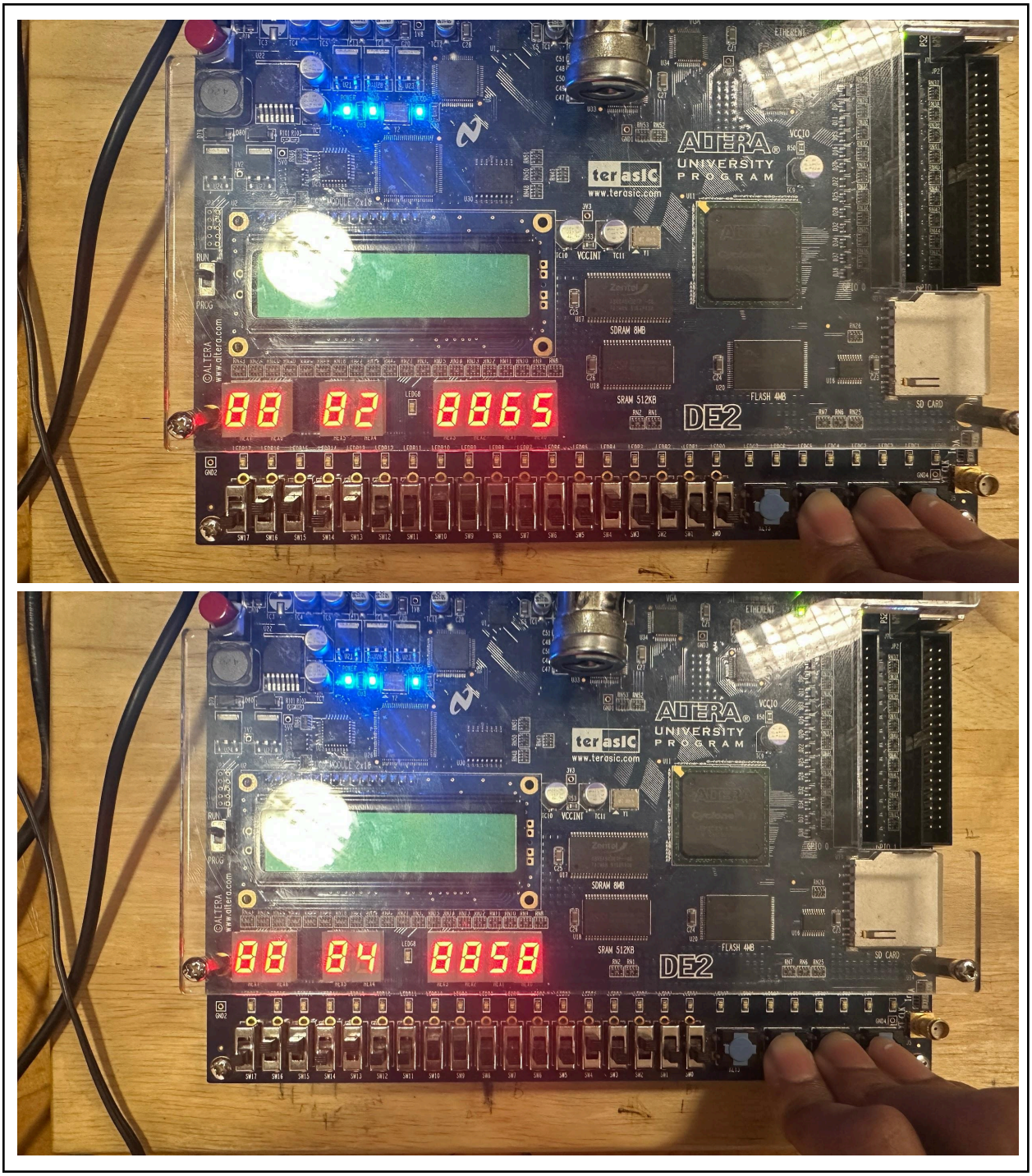

ALTERA
UNIVERSITY
PROGRAM
terasIC
www.terasic.com
DE2
SRAM 512KB
SDRAM 8MB
FLASH 4MB
SD CARD
ETHERNET
VGA
LINE IN
MIC

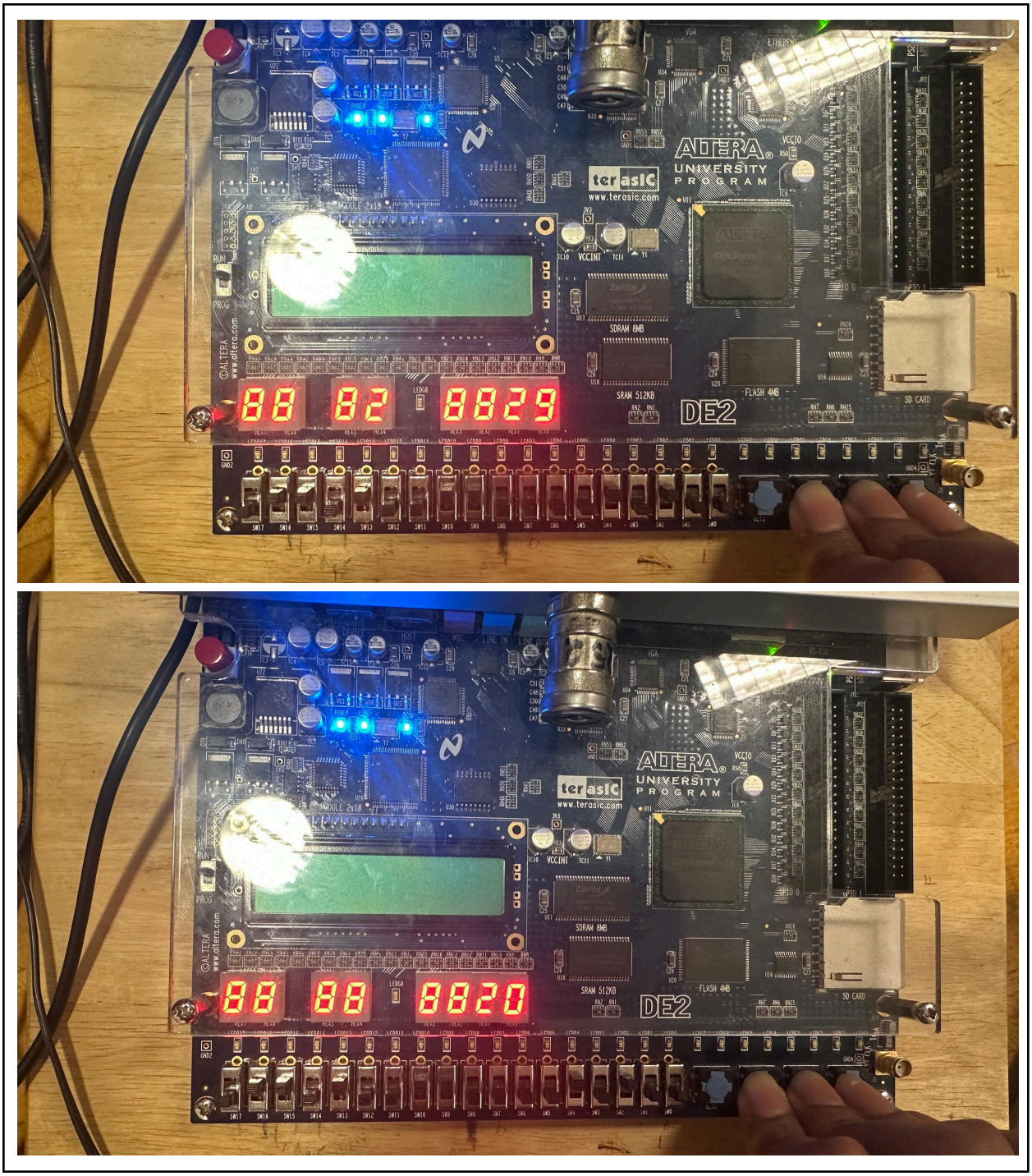

ALTERA
UNIVERSITY
PROGRAM
terasIC
www.terasic.com
DE2
SDRAM 8MB
SRAM 512KB
FLASH 4MB
SD CARD
ETHERNET
VGA
VCCINT
VCCIO

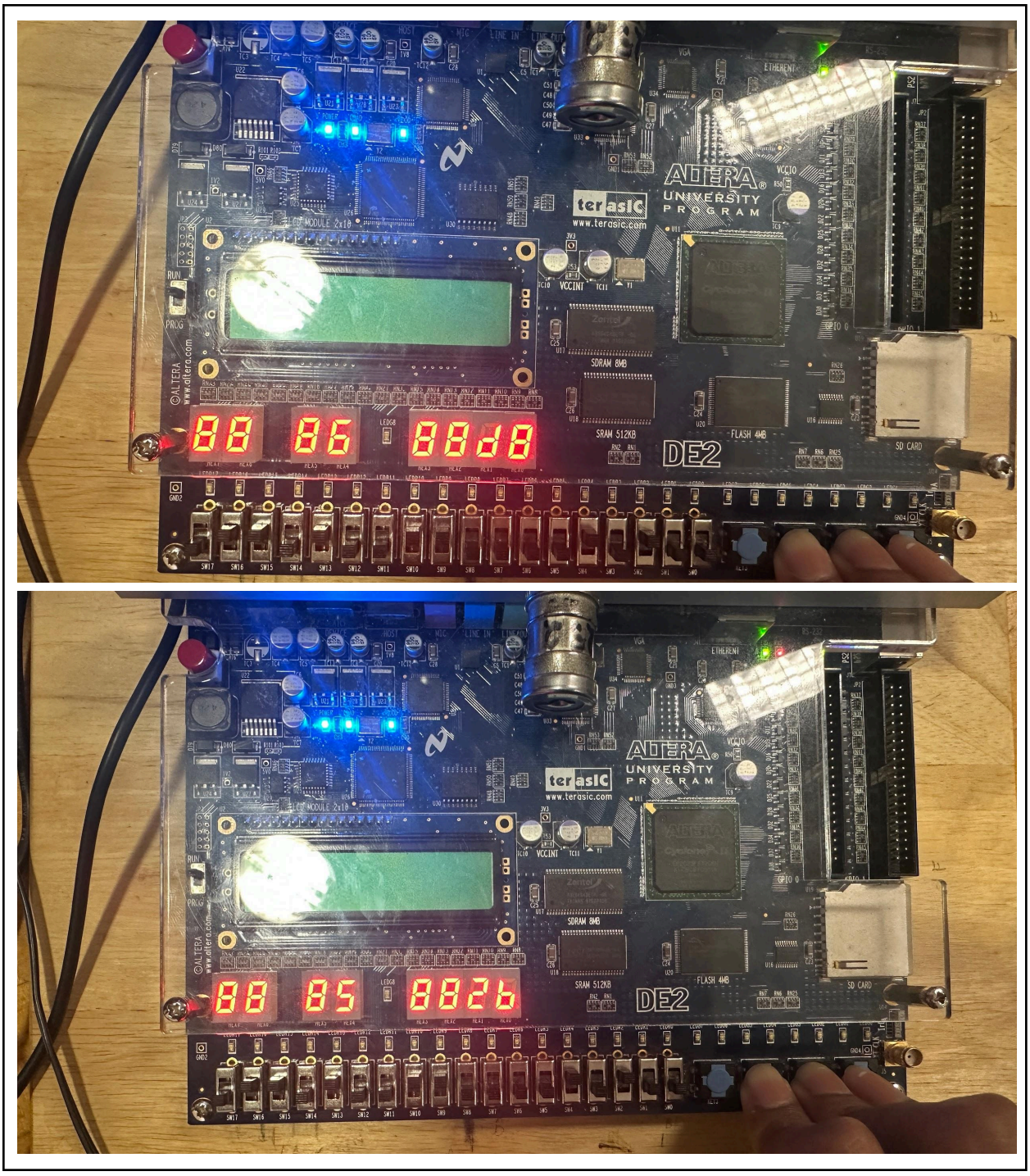

ALTERA
UNIVERSITY
PROGRAM
terasIC
www.terasic.com
DE2
SDRAM 8MB
SRAM 512KB
FLASH 4MB
SD CARD
ETHERNET
VGA
VCCINT
VCCIO

ALU2 Operation Pictures (One Full Sequence In Order)

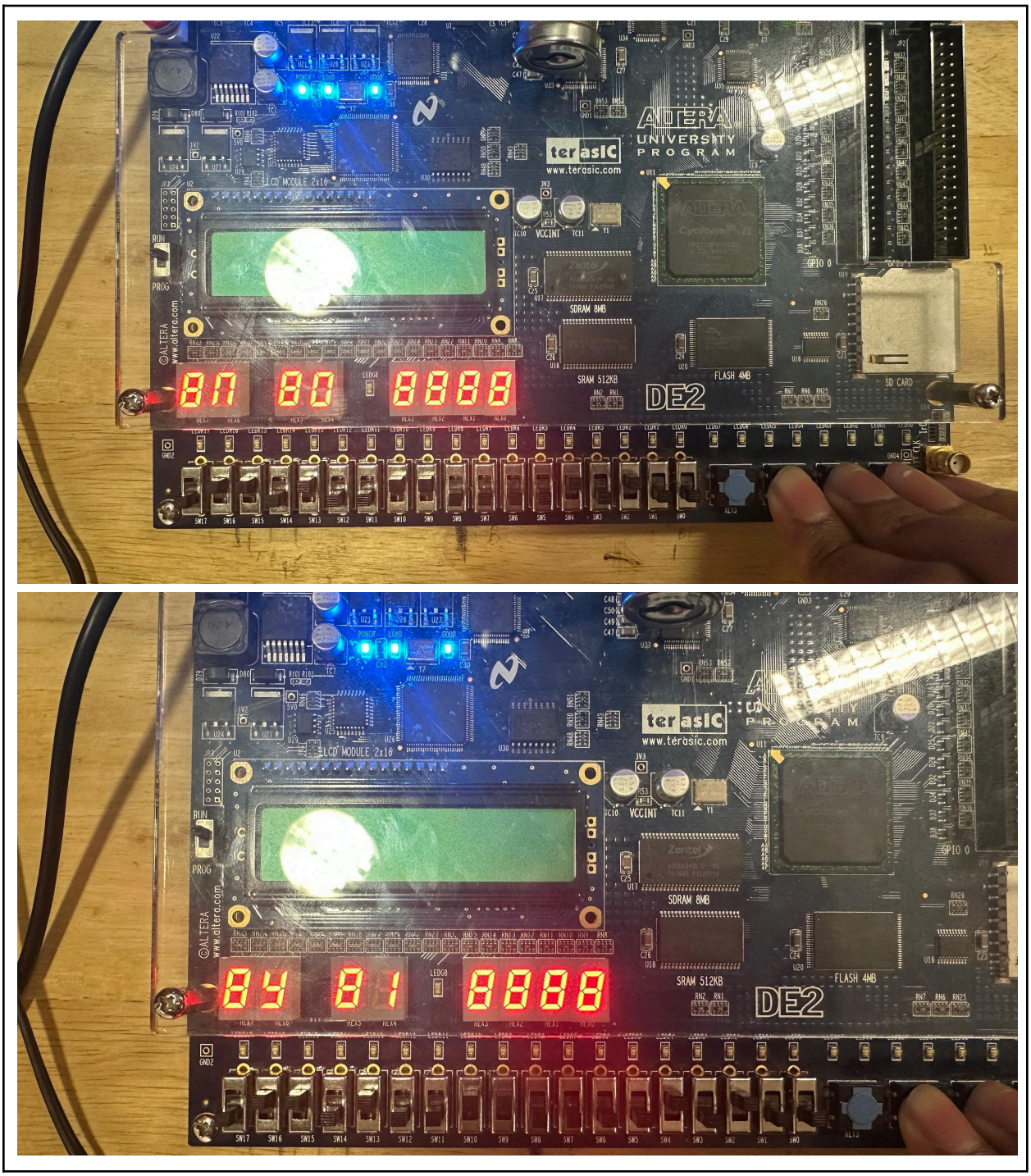


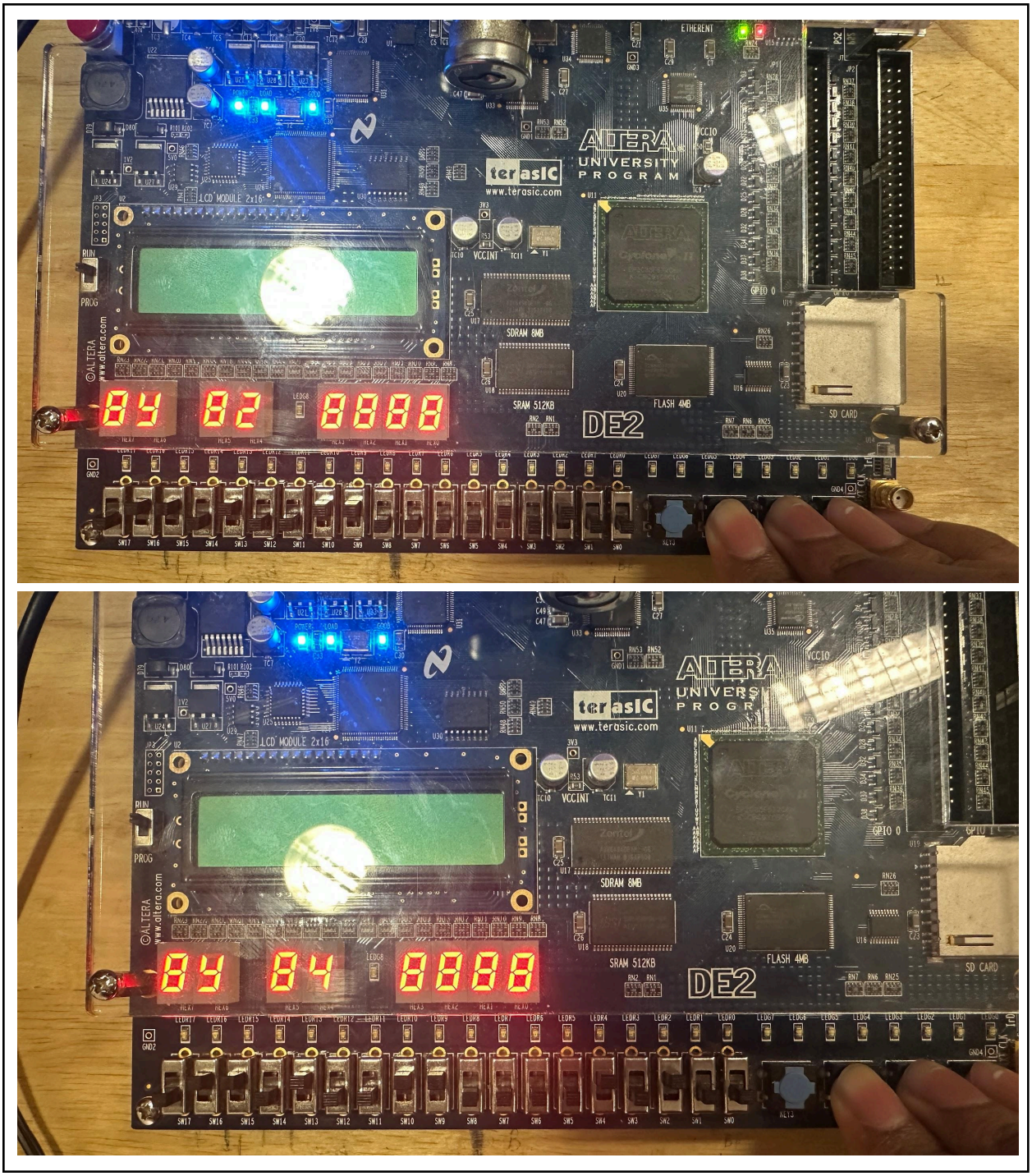

ALTERA
UNIVERSITY
PROGRAM
terasIC
www.terasic.com
DE2
SDRAM 8MB
SRAM 512KB
FLASH 4MB
SD CARD
ETHERNET
VCCIO
VCCINT
LEDG8
00 02 0865
00 04 0850

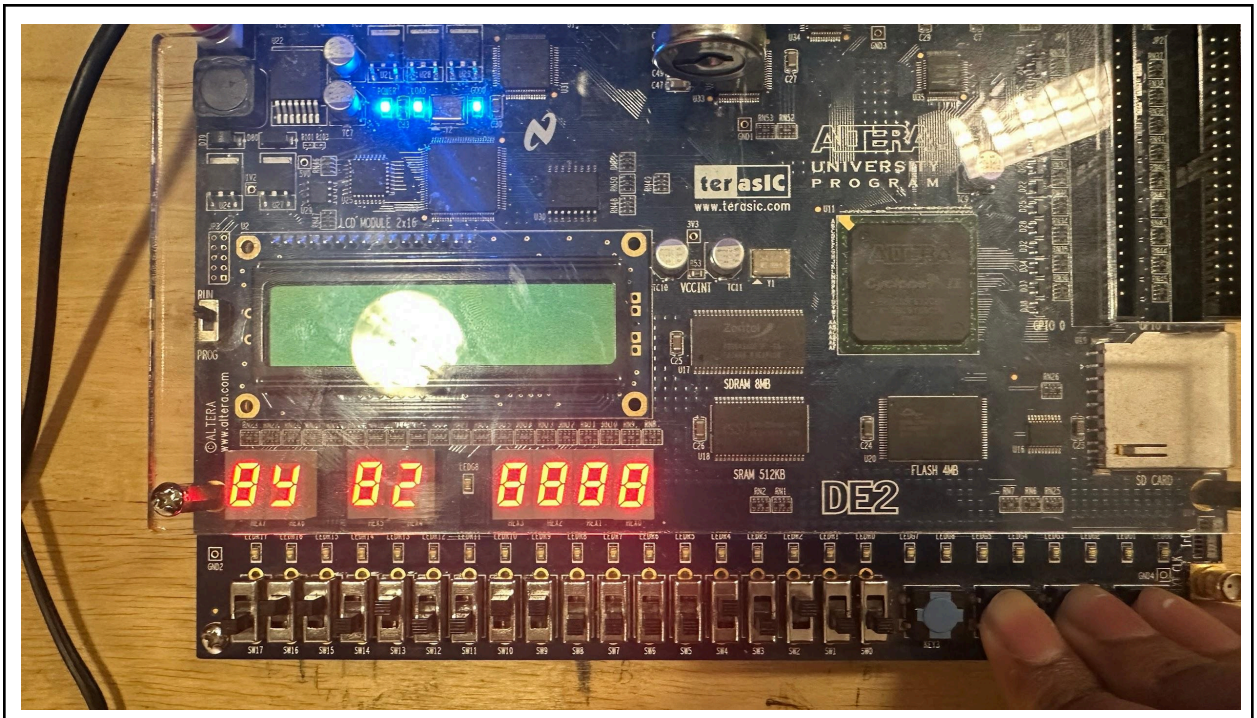

ALTERA
UNIVERSITY
PROGRAM
terasic
www.terasic.com
SDRAM 8MB
SRAM 512KB
FLASH 4MB
SD CARD
DE2
00 02 0029
ALTERA
www.altera.com
00 00 0020

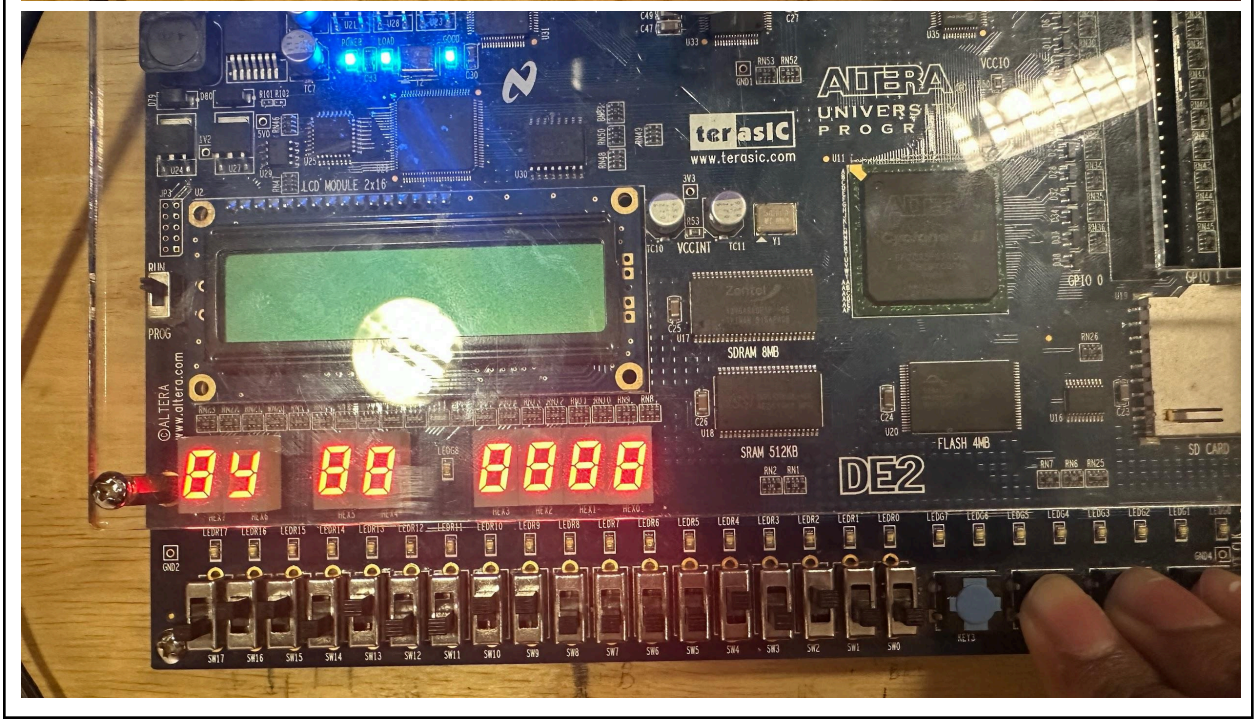

ALTERA
UNIVERSITY
PROGRAM
terasIC
www.terasic.com
DE2
SDRAM 8MB
SRAM 512KB
FLASH 4MB
SD CARD
ETHERNET
VGA
LCD MODULE 2x16
00 86 00d8
00 85 002b

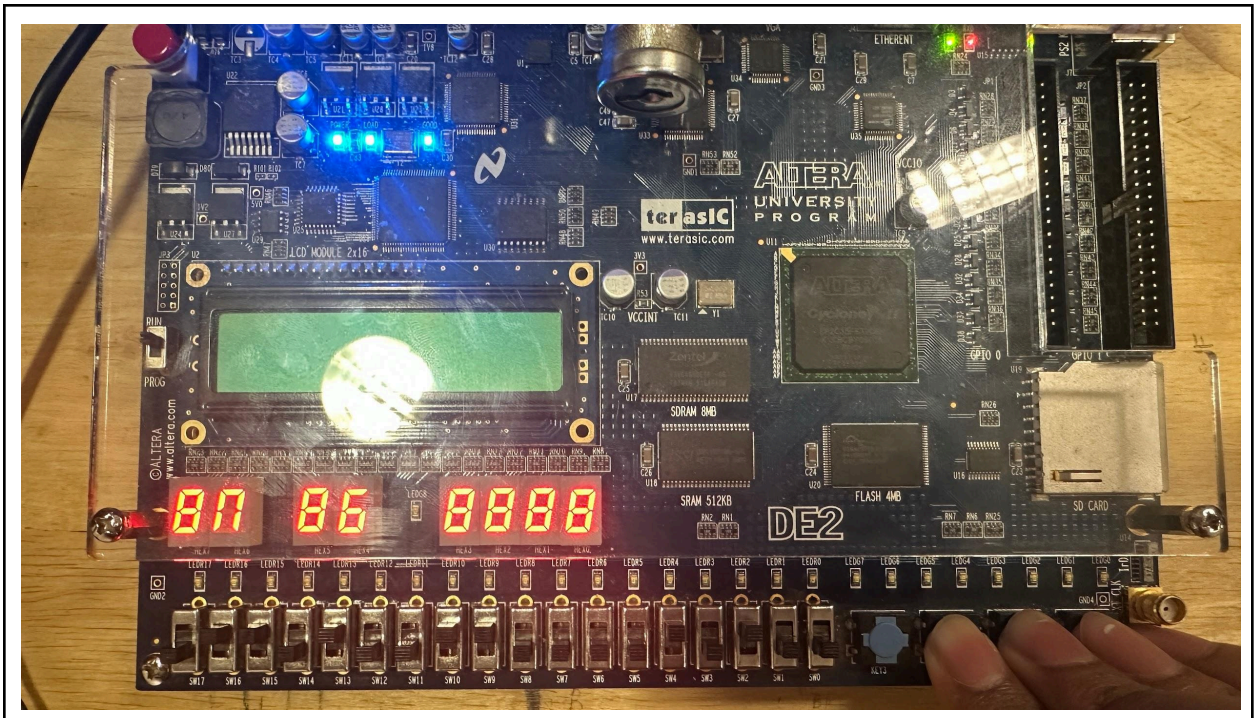
ALU3 Operation Pictures (One Full Sequence In Order)

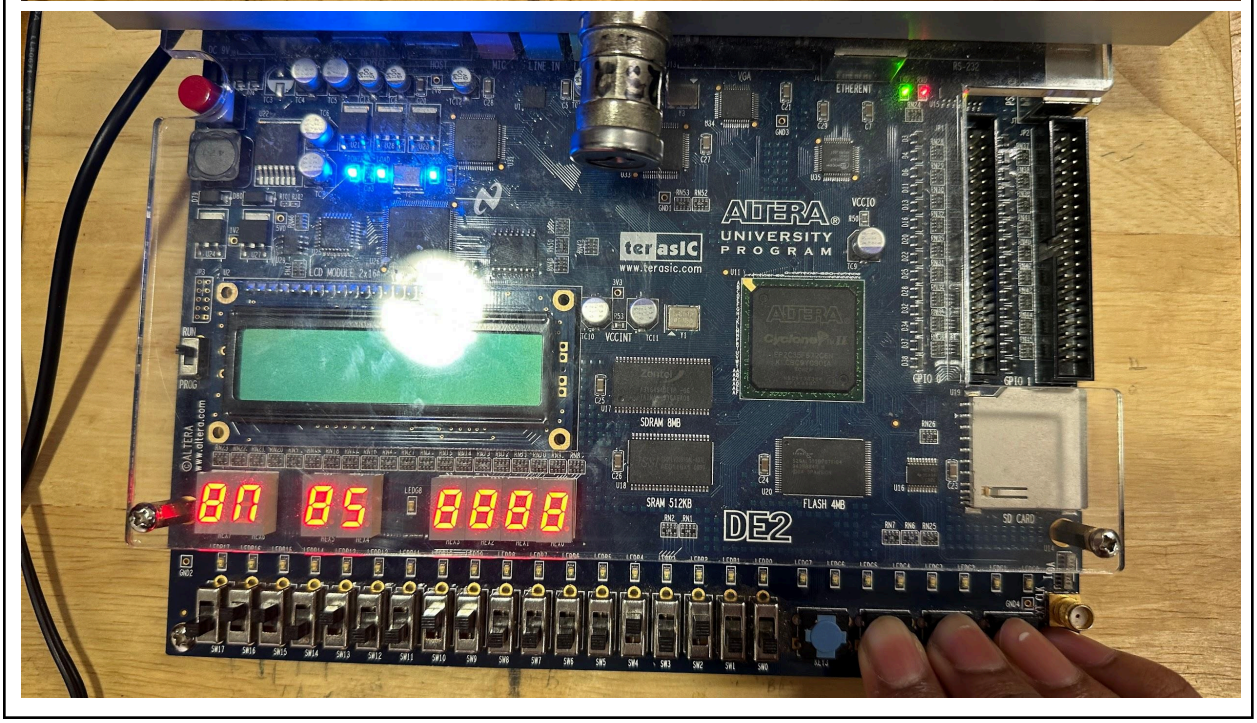



ALTERA
UNIVERSITY
PROGRAM
terasIC
www.terasic.com
LCD MODULE 2x16
SDRAM 8MB
SRAM 512KB
FLASH 4MB
DE2
SD CARD
GPIO 0
ETHERNET
VCCINT
VCC10
RUN
PROG
©ALTERA
www.altera.com
LEDG8
SW17 SW16 SW15 SW14 SW13 SW12 SW11 SW10 SW9 SW8 SW7 SW6 SW5 SW4 SW3 SW2 SW1 SW0


terasIC
www.terasic.com
ALTERA
UNIVERSITY
PROGRAM
LCD MODULE 2x16
SDRAM 8MB
SRAM 512KB
FLASH 4MB
SD CARD
DE2
©ALTERA
www.altera.com
PROG


terasIC
www.terasic.com
ALTERA
LCD MODULE 2x16
SDRAM 8MB
SRAM 512KB
FLASH 4MB
SD CARD
DE2
©ALTERA
www.altera.com
PROG


ALTERA
UNIVERSITY
PROGRAM
terasIC
www.terasic.com
LCD MODULE 2x16
SDRAM 8MB
SRAM 512KB
FLASH 4MB
DE2
SD CARD
ETHERENT
VCCINT
VCCIO
GPIO 0
RUN
PROG
©ALTERA
www.altera.com


ALTERA
UNIVERSITY
PROGRAM
terasIC
www.terasic.com
LCD MODULE 2x16
SDRAM 8MB
SRAM 512KB
FLASH 4MB
DE2
SD CARD
ETHERENT
VCCINT
VCCIO
GPIO 1
RUN
PROG
©ALTERA
www.altera.com

Conclusion

In Lab 6, I was able to successfully design and implement a simple working CPU that could perform basic arithmetic and logical operations. Throughout the semester, I learned about various components and how their logic worked. Through working on the labs prior to Lab 6, I was able to gain a deeper understanding on how various digital circuits work. Things like storage units are crucial to keep memory of the input data passed on to the system. Without a decoder, it would be difficult to pass on instructions in a concise and neat manner to the ALU. Most importantly, the clock signal is an extremely important aspect. Without a clock to time everything synchronously, no calculation would occur properly; matter of fact, the system would be completely dysfunctional. Using all of this knowledge I gained through my time studying the course, I was able to strive really hard to achieve the accomplishment of successfully completing Lab 6. It was definitely challenging to get the entire system to work and there were several hurdles along the way. But with each obstacle, I was patient and analysed the situation to be able to move past it and finish the project. The knowledge I gained from this course, its content and all of the labs will most definitely help me in my future endeavours and I am grateful for that!