Ryerson University

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

Course Title:	COE
Course Number:	608
Semester/Year (e.g.F2016)	w2024

Instructor:	Dr. Hafeez

Assignment/Lab Number:	3
Assignment/Lab Title:	b

Submission Date:	13/2/2024
Due Date:	13/2/2024

Student LAST Name	Student FIRST Name	Student Number	Section	Signature*
Mohamed	Lama	501042394	042	L A M A

*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: http://www.ryerson.ca/senate/current/pol60.pdf

1. Lab Objective

In this lab, we need to implement and test an 8-bit version of the ALU that we designed in the first part of the ALU designed in the first part of the lab.

I first modified the ALU from 32-bit, to 8-bit and also created an 8-bit adder to use in the 8-bit ALU to replace the 32-bit adder we used previously.

2. Experiment Details:

8-bit ALU:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity alu8 is
port(
   a : in std_logic_vector(7 downto 0);
   b : in std_logic_vector(7 downto 0);
   op : in std_logic_vector(2 downto 0);
   result : out std_logic_vector(7 downto 0);
   zero : out std_logic;
   cout : out std_logic );
end alu8;

architecture Behavior of alu8 is
   component adder8
      port(
         Cin : in std_logic;
         X,Y : in std_logic_vector(7 downto 0);
         S : out std_logic_vector(7 downto 0);
         Cout : out std_logic
         );
      end component;
      signal result_s: std_logic_vector(7 downto 0):=(others=>'0');
      signal result_add: std_logic_vector(7 downto 0):=(others=>'0');
      signal result_sub: std_logic_vector(7 downto 0):=(others=>'0');
      signal cout_s: std_logic := '0';
      signal cout_add: std_logic := '0';
      signal cout_sub: std_logic := '0';
      signal zero_s: std_logic := '0';

begin
         add0 : adder8 port map(op(2), a, b, result_add, cout_add);
         sub0 : adder8 port map(op(2), a, not b, result_sub, cout_sub);

      process (a,b,op)
      begin
         case(op) is
               when "000" => -- "000" a and b
                  result_s <= a and b;
                  cout_s <= '0';
               when "001" => -- "001" a or b
                  result_s <= a or b;
                  cout_s <= '0';
               when "010" => -- "010" a + b
                  result_s <= result_add;
                  cout_s <= cout_add;
```

```
50                when "110" => -- "110" a - b
51                   result_s <= result_sub;
52                   cout_s <= cout_sub;
53                when "100" => -- "100" a << 1 "ROL" "sll
54                   result_s <= a(6 downto 0) & '0';
55                   cout_s <= a(7);
56                when "101" => -- "101" a >> 1 "ROR"
57                   result_s <= '0' & a(7 downto 1);
58                   cout_s <= '0';
59                when others => -- any other value of opcode defaults to passing a through as the result
60                   result_s <= a;
61                   cout_s <= '0';
62             end case;
63
64          case (result_s) is --case statement is used to determine the value of the zero flag based on the result_s value--
65             when(others => '0') =>
66                zero_s <= '1';
67             when others =>
68                zero_s <= '0';
69          end case;
70       end process;
71
72       result <= result_s;
73       cout <= cout_s;
74       zero <= zero_s;
75       end Behavior;
```

Lab3_b:

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

ENTITY lab3_b IS
   PORT
   (
   --Input Switches
      SW             : IN  STD_LOGIC_VECTOR(17 DOWNTO 0);
   --Seven-Segment Display Outputs
      HEX0, HEX1         : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX2, HEX3         : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX4, HEX5         : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX6, HEX7         : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
   --Green Led Outputs
      LEDG           : OUT STD_LOGIC_VECTOR(8 DOWNTO 0);
   --Red Led Outputs
      LEDR           : OUT STD_LOGIC_VECTOR(17 DOWNTO 0)
   );
END lab3_b;

ARCHITECTURE behavior OF lab3_b IS

   --Use these signals to connect inputs to your ALU. A and B
   --are the operand inputs and op is the Op-Code input.
   SIGNAL A_to_ALU : STD_LOGIC_VECTOR(7 DOWNTO 0);
   SIGNAL B_to_ALU : STD_LOGIC_VECTOR(7 DOWNTO 0);
   SIGNAL op_to_ALU : STD_LOGIC_VECTOR(2 DOWNTO 0);

   --Use these signals to connect ALU outputs to the seven-segment displays
   --and leds. Result is the 8-bit output of the ALU. Zero and Cout should
   --be self-evident.
   SIGNAL result_from_ALU : STD_LOGIC_VECTOR(7 DOWNTO 0);
   SIGNAL zero_from_ALU : STD_LOGIC;
   SIGNAL cout_from_ALU : STD_LOGIC;

   SIGNAL tmpInv : STD_LOGIC;

   COMPONENT sevenSeg_8bit
   PORT
   (
   -- 8-bit signed number input.
      dataIn       : IN  STD_LOGIC_VECTOR(7 downto 0);
   -- 7-bit segment control signals.
      segVal1      : OUT STD_LOGIC_VECTOR(6 downto 0);
      segVal2      : OUT STD_LOGIC_VECTOR(6 downto 0);
   -- 1-bit sign control signal.
      sign      : OUT STD_LOGIC
   );
   END COMPONENT;
```

```
   --=============================================================
   -- This is where your ALU component declarations should go.
   --=============================================================
   COMPONENT alu8
   PORT
   (
      a, b   : IN  std_logic_vector(7 DOWNTO 0);
      op     : IN  std_logic_vector(2 DOWNTO 0);
      result    : OUT    std_logic_vector(7 DOWNTO 0);
      zero   : OUT    std_logic;
      cout   : OUT    std_logic
   );
   END COMPONENT;

EGIN

   --Signal routing from the input/output pins to the ALU and other
   --utilities.
   A_to_ALU <= SW(17 DOWNTO 10);
   B_to_ALU <= SW(10 DOWNTO 3);
   op_to_ALU <= SW(2 DOWNTO 0);

   LEDG(0) <= zero_from_ALU;
   LEDG(1) <= cout_from_ALU;

   HEX2(6) <= NOT tmpInv;

   --Constants
   HEX2(5 DOWNTO 0) <= "111111";
   HEX3(6 DOWNTO 0) <= "1111111";

   sevSeg_driver_A : sevenSeg_8bit
   PORT MAP
   (
      dataIn => A_to_ALU,
      segVal1 => HEX6,
      segVal2 => HEX7,
      sign => LEDR(16)
   );

   sevSeg_driver_B : sevenSeg_8bit
   PORT MAP
   (
      dataIn => B_to_ALU,
      segVal1 => HEX4,
      segVal2 => HEX5,
      sign => LEDR(13)
   );
```

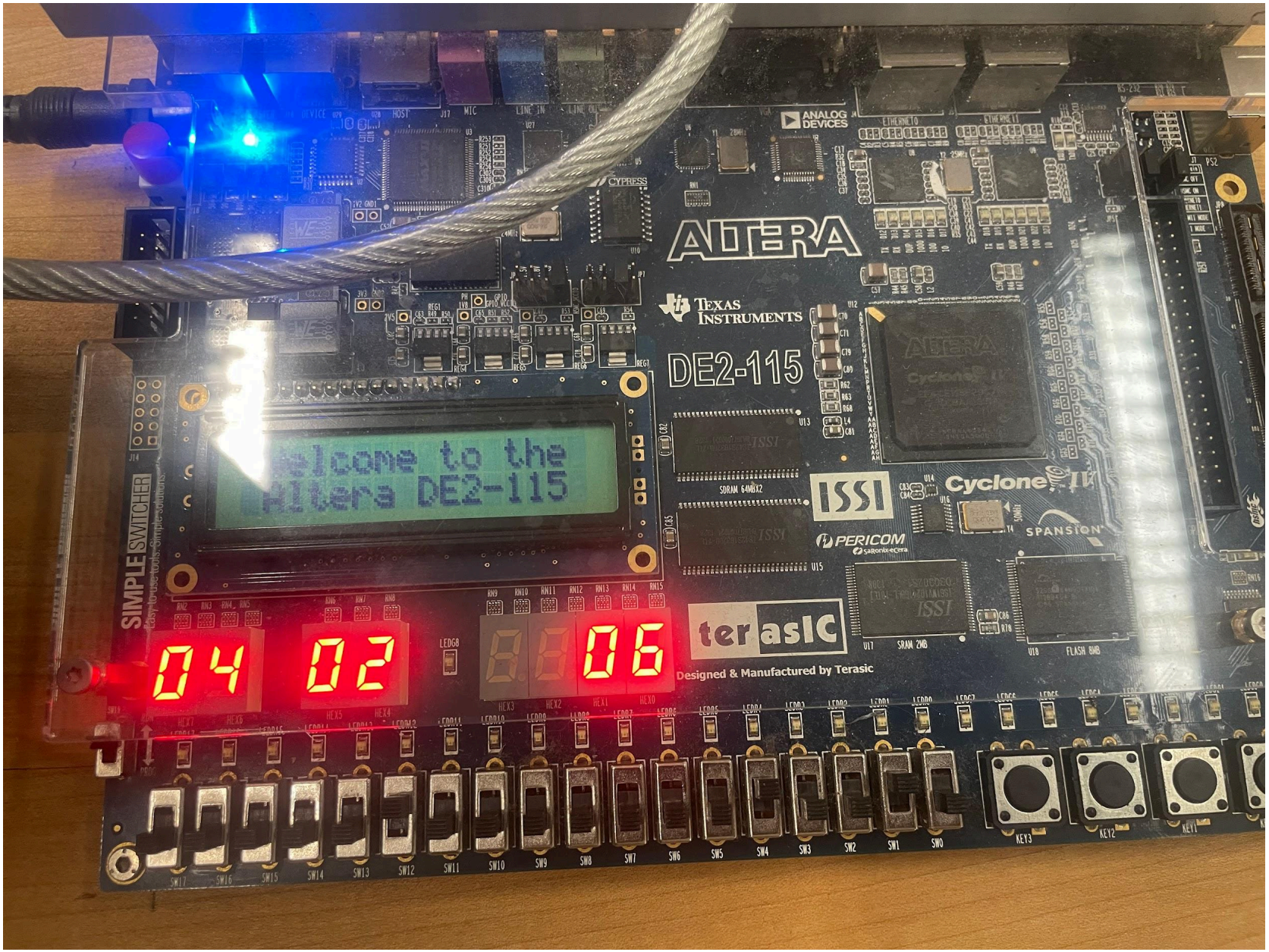
```
    sevSeg_driver_ALU : sevenSeg_8bit
    PORT MAP
    (
       dataIn => result_from_ALU,
       segVal1 => HEX0,
       segVal2 => HEX1,
       sign => tmpInv
    );

    --============================================================
    --This is where the ALU must be connected. The signal names
    --should be changed to match your ALU.
    --============================================================

    the_ALU : alu8
    PORT MAP
    (
    --Operand A
       a => A_to_ALU,
    --Operand B
       b => B_to_ALU,
    --Operation Code
       op => op_to_ALU,
    --ALU Result
       result => result_from_ALU,
    --Zero Signal
       zero => zero_from_ALU,
    --Cout Signal
       cout => cout_from_ALU
    );

END behavior;
```

Adder8:

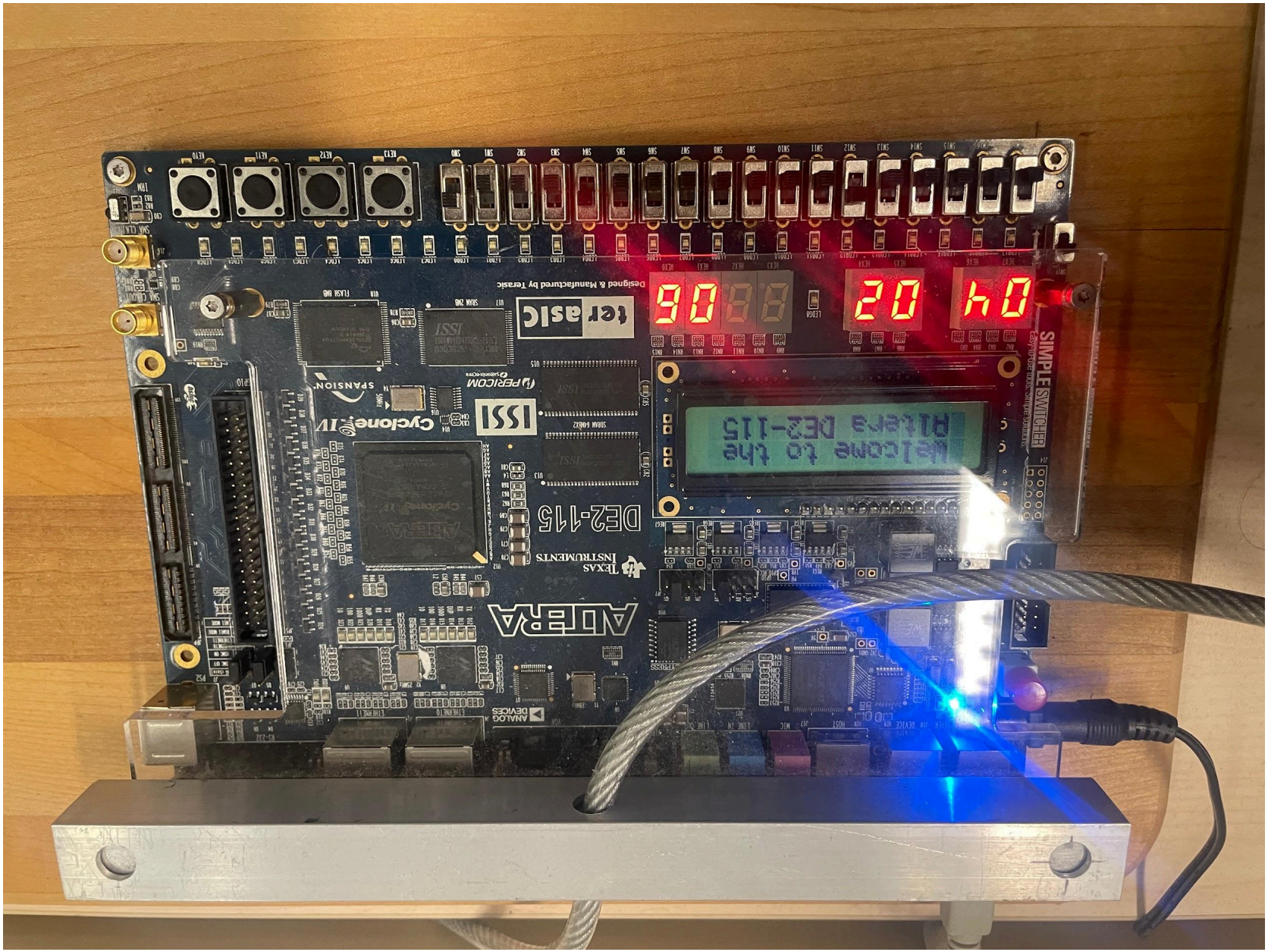
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity adder8 is
    port (
        Cin : in  std_logic;
        X,Y : in  std_logic_vector(7 downto 0);
        S   : out std_logic_vector(7 downto 0);
        Cout: out std_logic
    );
end entity adder8;

architecture Behavioral of adder8 is
begin
    process(X, Y, Cin)
        variable sum : unsigned(8 downto 0);
    begin
        sum := ('0' & unsigned(X)) + ('0' & unsigned(Y)) + (Cin & "00000000");
        S <= std_logic_vector(sum(7 downto 0));
        Cout <= sum(8);
    end process;
end architecture Behavioral;
```

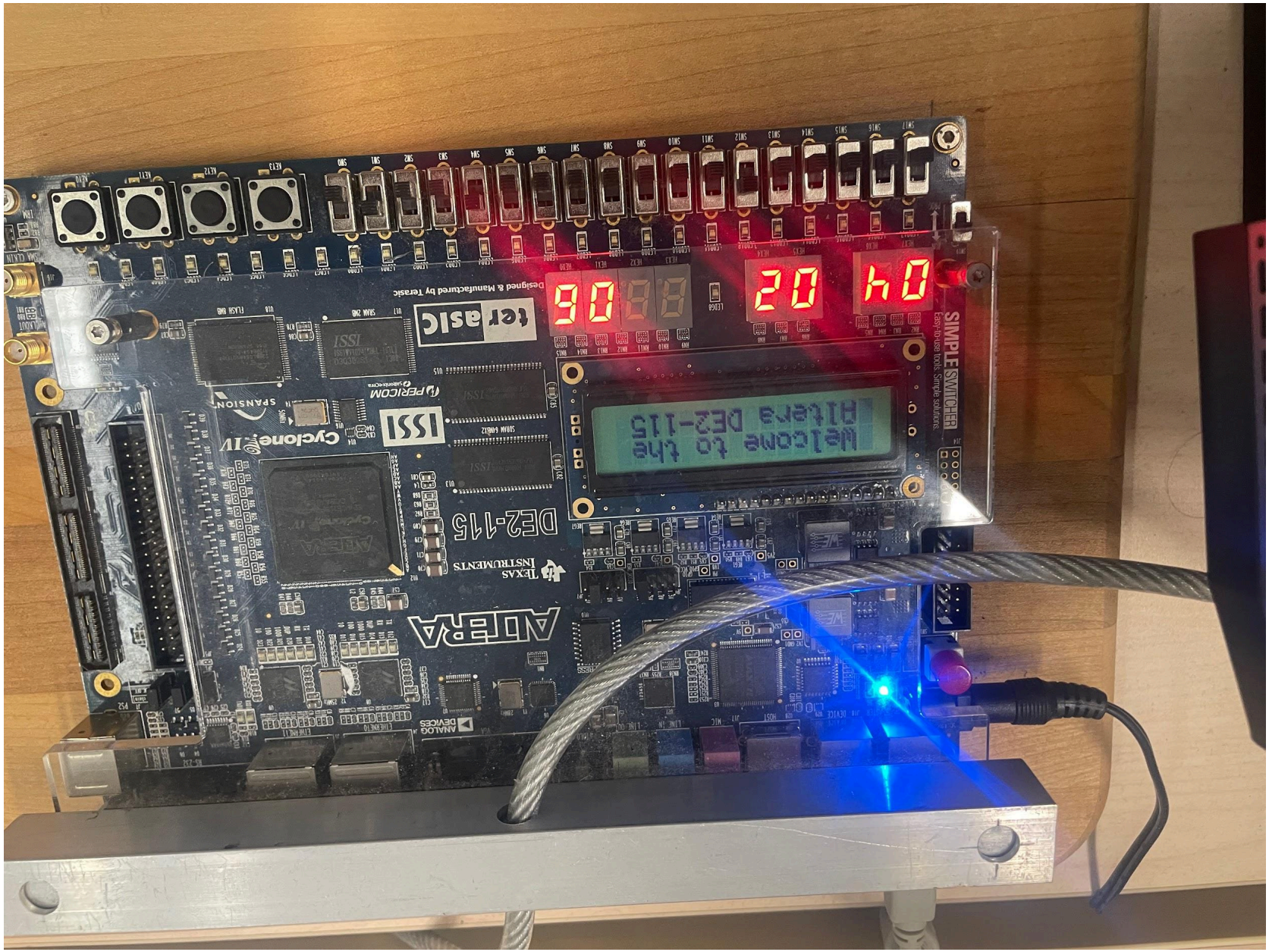
3. Results:

Welcome to the
Altera DE2-115
DE2-115
ALTERA
Designed & Manufactured by Terasic

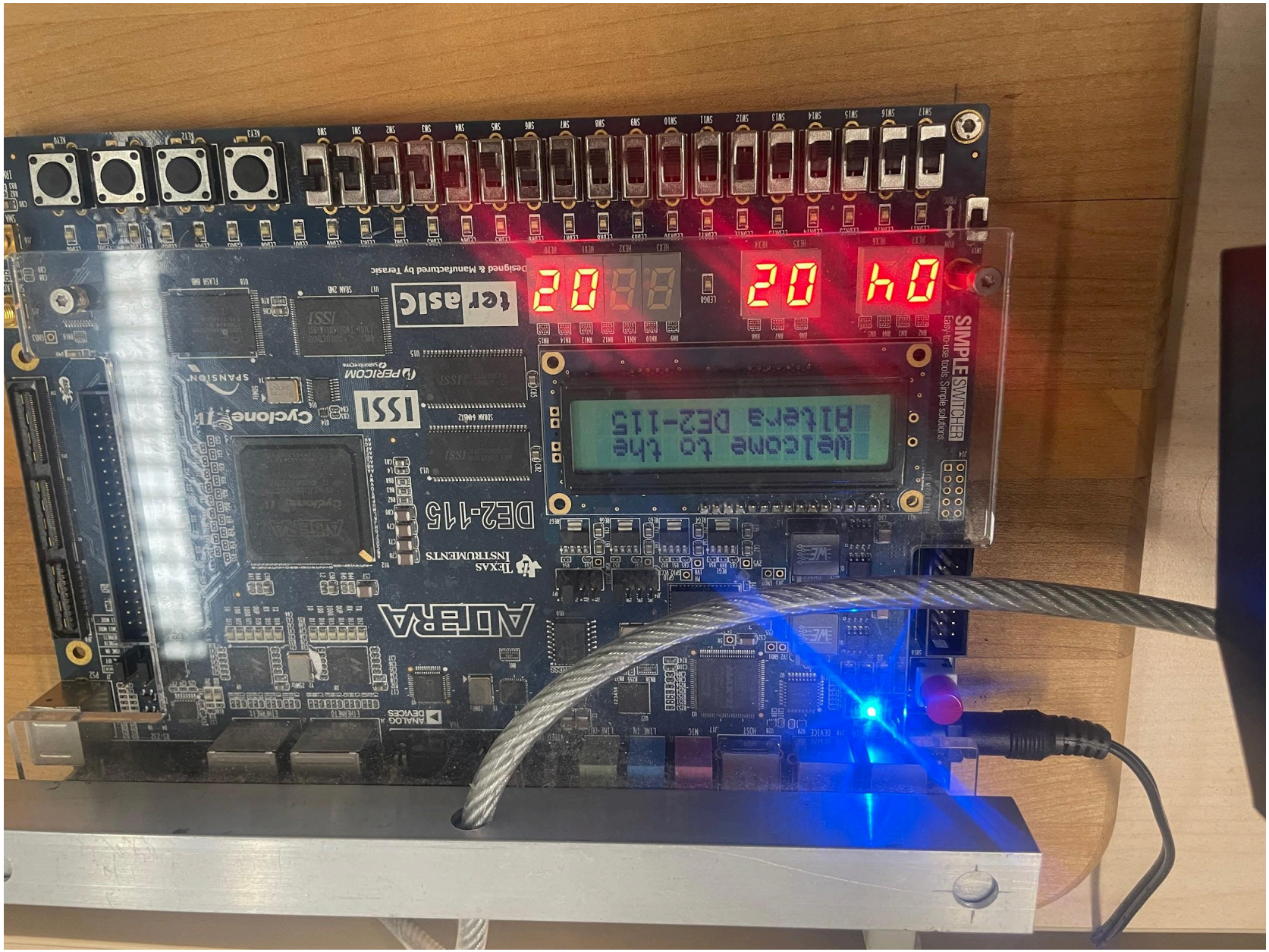

elcome to the
Altera DE2-115
ALTERA
DE2-115
Cyclone II
ISSI
PERICOM
SPANSION
terasIC
Designed & Manufactured by Terasic
04 02 06

SIMPLE SWITCHER
Welcome to the
Altera DE2-115
04
02
00
DE2-115
ALTERA
Cyclone IV
Designed & Manufactured by Terasic

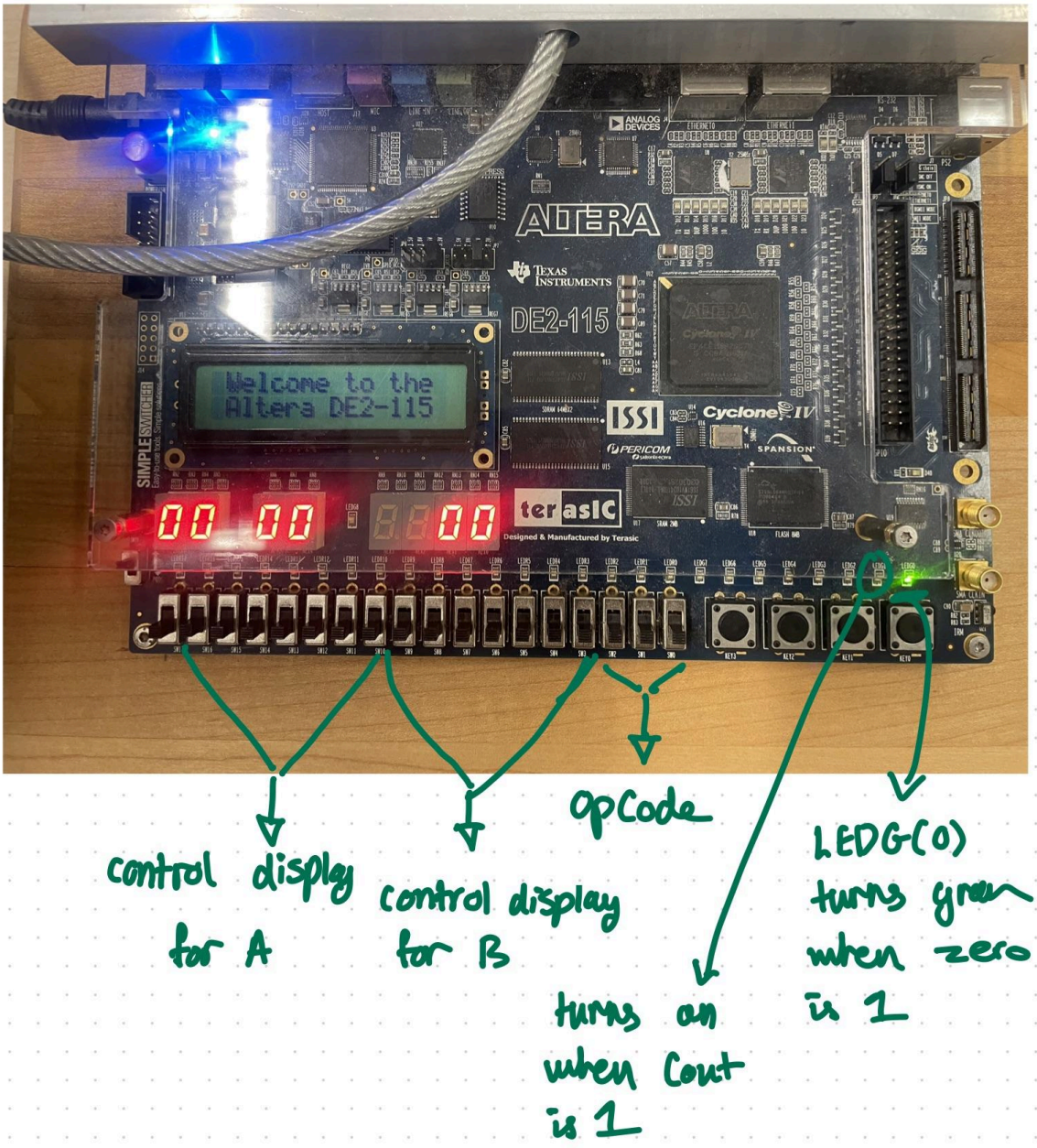

DE2-115
terasic
ALTERA
ISSI
Cyclone IV
PERICOM
TEXAS INSTRUMENTS
ANALOG DEVICES
SIMPLE SWITCHER
Welcome to the
Altera DE2-115


DE2-115
ALTERA
Cyclone IV
ISSI
PERICOM
TEXAS INSTRUMENTS
terasic
SIMPLE SWITCHER
Easy-to-use tools. Simple solutions.
Designed & Manufactured by Terasic
Altera DE2-115
welcome to the

SIMPLE SWITCHER
Easy-to-use tools. Simple solutions.
Welcome to the
Altera DE2-115
terasic
Designed & Manufactured by Terasic
DE2-115
ALTERA
Cyclone IV
ISSI
PERICOM
SPANSION
TEXAS INSTRUMENTS
ANALOG DEVICES


DE2-115
ALTERA
Cyclone II
ISSI
PERICOM
TEXAS INSTRUMENTS
ANALOG DEVICES
terasic
Designed & Manufactured by Terasic
SIMPLE SWITCHER
Easy-to-use tools. Simple solutions.
Welcome to the
Altera DE2-115

4. Discussion



The results that were observed from the board were as expected. The leds and switches were all responsive and performed as expected. When setting a = 4 and b = 2, all the operation codes were accurate, as well as the zero flag and count flag.