Student Name: Ricardo Garza
Student UNT email ID: ricardogarza3@my.unt.edu 10967208

Requirements:

1. Using **Behavioral Modeling** design an ALU to perform the operations: Add, Subtract, multiply, Divide, less than, greater than, Logical AND, Logical OR. _____

2. Using testbench test the designed ALU using **at least two testcases for each operation**. Verify the results from the waveforms. _____

3. First operand should be connected to Switches SW0 (LSB) - SW3 (MSB) on Nexys 4 DDR board. Second operand should be connected to Switches SW4 (LSB) – SW7(MSB) on Nexys 4 DDR board. Type of operation should be selected based on select lines SW8 (LSB) – SW10 (MSB) on Nexys 4 DDR board. The output of Adder and subtractor are connected LED0 (LSB) – LED4 (MSB). Output of Multiplier will be 8 bits and should be connected to LED0 (LSB) – LED7 (MSB). Divider output should be connected to LED0 (LSB) – LED3 (MSB). The output of less than and greater than should be connected to LED0 and LED1, Respectively. Logical AND and Logical OR outputs should be connected to LED0 (LSB) – LED3 (MSB). **Follow the Switch and LED assignments** as given. _____

Learning Objectives:
The general learning objectives of this lab to design an implement a 4-bit ALU that can perform addition, subtraction, multiplication, division, and other logical operations.

General Steps:
1. Design a ALU using a 2 4-bit inputs and an 8-bit output with a clock using behavioral modelling.
2. Using case statements design 8 different operations
    a. The different operations will include addition, subtraction, division, multiplication, greater than, less than, AND gate and OR gate
3. Make sure operations work based on clock input.
4. Addition, Subtraction, greater than, less than, AND, OR will use LED0 – LED3
5. Multiplication and division will use LED0 – LED8
6. Write a testbench to verify the design and two test cases for testing
7. Simulate and capture the waveforms
8. Synthesize the design and then run on Nexys 4 FPGA board

Student Name: Ricardo Garza
Student UNT email ID: ricardogarza3@my.unt.edu 10967208

Detailed Steps:
Some detailed steps to complete this lab were

1. Create a source file to include all ports needed and clock.

```
Port ( clock: in STD_LOGIC;                          --system clock
       --res: in STD_LOGIC;                            --synchronous reset
       a : in STD_LOGIC_VECTOR (3 downto 0);         --a input
       b : in STD_LOGIC_VECTOR (3 downto 0);         --b input
       alu_select: in STD_LOGIC_VECTOR(2 downto 0);  --logic unit opcode
       y : out STD_LOGIC_VECTOR (7 downto 0));       --output from logic unit
```

a.

2. After making sure behavioral modelling is being used using a case statements design the logic

```
case(alu_select) is
  when "000" =>
    y <= std_logic_vector(to_unsigned((to_integer(unsigned(a)) + to_integer(unsigned(b))),8)); --addition
  when "001" =>
    y <= std_logic_vector(to_unsigned((to_integer(unsigned(b)) - to_integer(unsigned(a))),8)); --subtraction
  when "010" => -- multilication
    y <= std_logic_vector(to_unsigned((to_integer(unsigned(a)) * to_integer(unsigned(b))),8));
  when "011" => --division
    y <= std_logic_vector(to_unsigned((to_integer(unsigned(a)) / to_integer(unsigned(b))),8));
  when "100" =>
    if(a>b) then
      y <= x"01";
    else
      y <= x"00";
    end if;
  when "101" =>
    if (a<b) then
      y <= x"01";
    else
      y <= x"00";
    end if;
  when "110" =>
    y(3 downto 0) <= a(3 downto 0) and b(3 downto 0);
    y(7 downto 4) <= "0000";
  when "111" =>
    y(3 downto 0) <= a(3 downto 0) or b(3 downto 0);
    y(7 downto 4) <= "0000";
```

a.

3. Finishing up the source file start writing the testbench.
   a. The testbench should include nothing in the entity
   b. The component must include the ALU and all the correct ports.

```
COMPONENT lab_3source
PORT( clock: in std_logic;
      --res: in std_logic;
      a: in std_logic_vector (3 downto 0);
      b: in std_logic_vector (3 downto 0);
      alu_select: in std_logic_vector (2 downto 0);
      y: out std_logic_vector(7 downto 0));
END COMPONENT;
```

   c.
4. Afterwards you run the simulation.
5. Attach the given constraints given on canvas

Student Name: Ricardo Garza
Student UNT email ID: ricardogarza3@my.unt.edu 10967208

a. Make sure to uncomment the lines needed for this lab
b. Make sure to also include the clock in the clock signal.
c. A should only include the first 4 switches
d. B should only include the second 4 switches but be different bits
e. Same with selector.

```
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { a[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { a[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { a[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { a[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { b[0] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { b[1] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { b[2] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports { b[3] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports { alu_select[0] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports { alu_select[1] }]; #IO_25_34 Sch=sw[9]
```

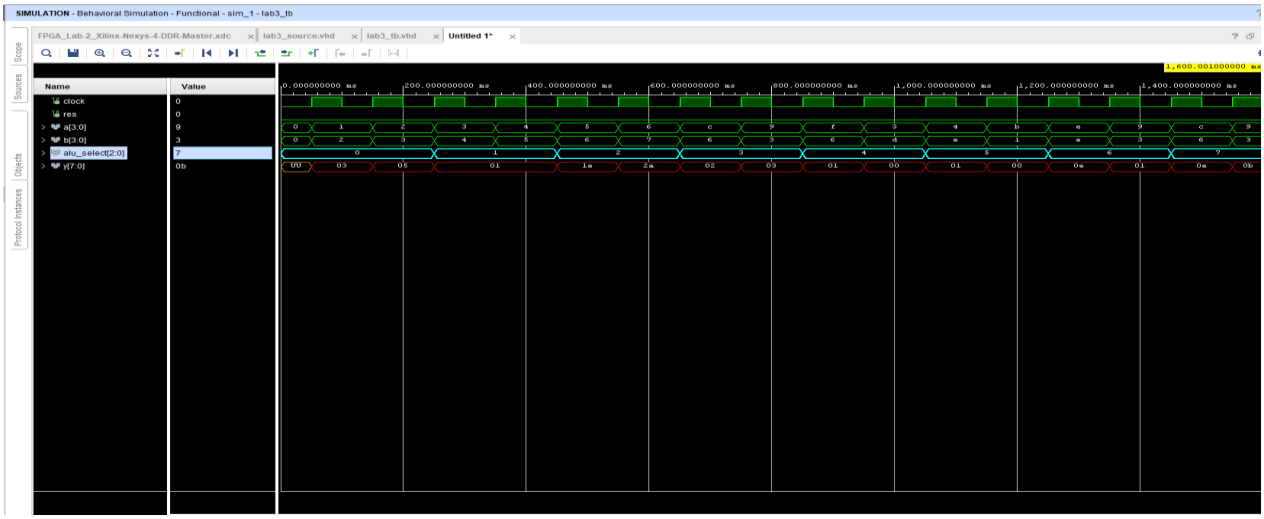
f. `set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports { alu_select[2] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]`
g. Make sure you have 8 led pins assigned to you output

```
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { y[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { y[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { y[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { y[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { y[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { y[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { y[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
```

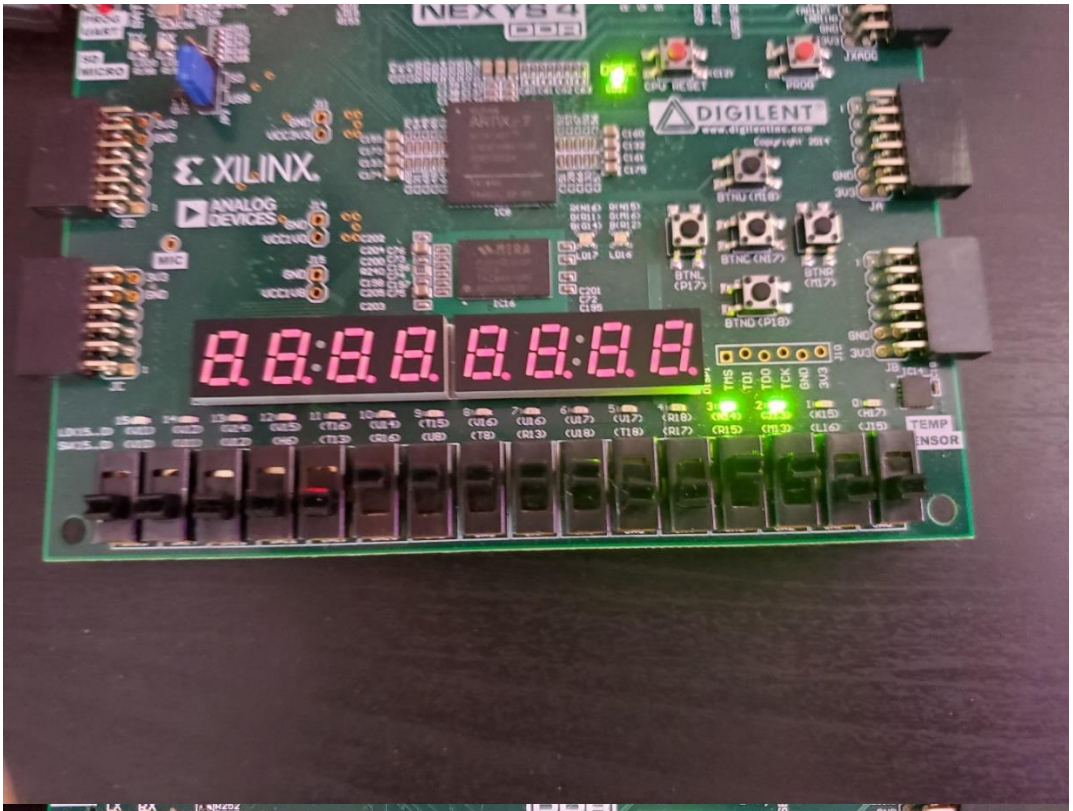
h. `set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { y[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]`

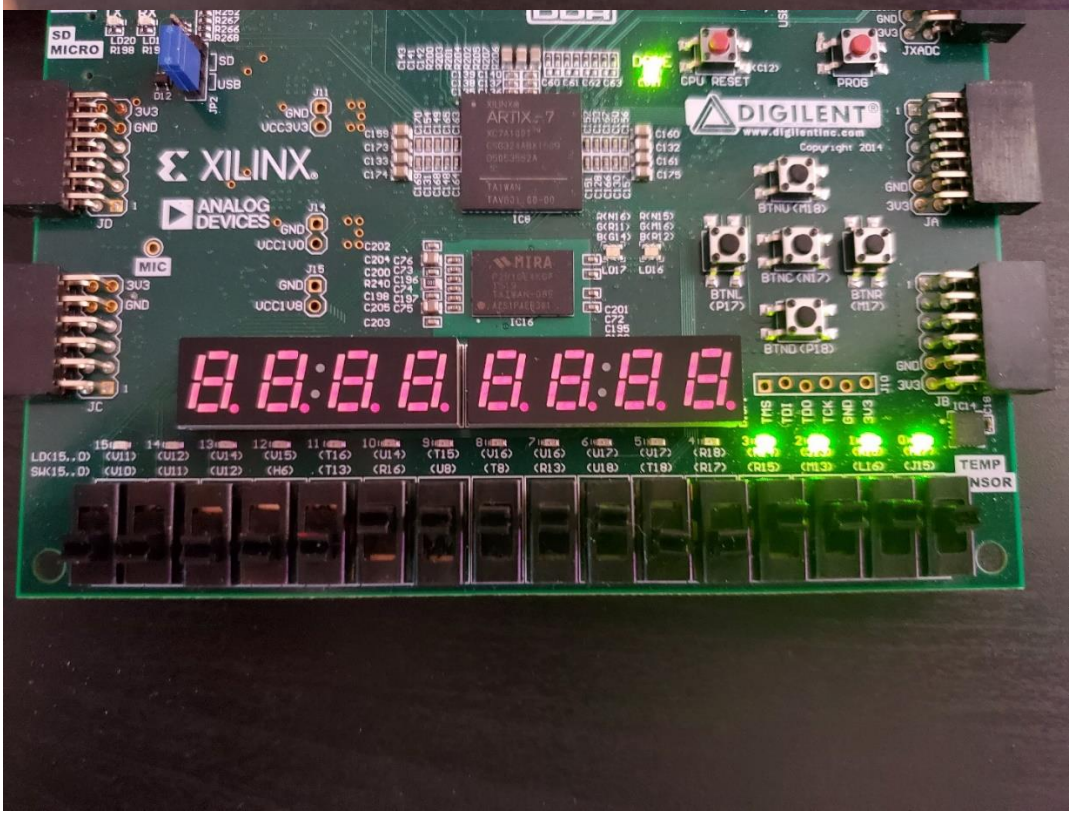
6. Run the synthesis, implementation and generate bitstream
    a. Make sure to fix any errors generated along the way
7. Run on the Nexys 4 FPGA board

Observations:
One thing I noticed was you must have all your test case signal in correctly or you get a weird waveform. I also noticed that I had to double check my logic in the source file. I would get random waveform outputs or it would just crash. Luckly it would have an arrow to were it would crash so it was hard to find the where the error was. Fixing them was another issue. Had to find out how to covert to integer then back.

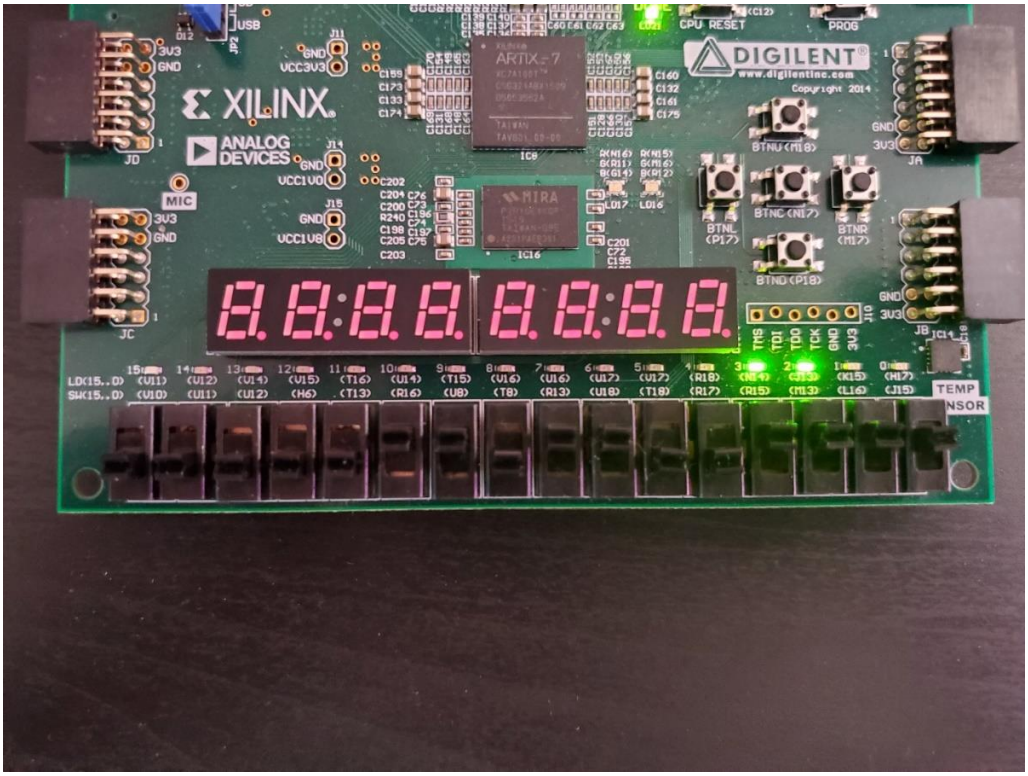


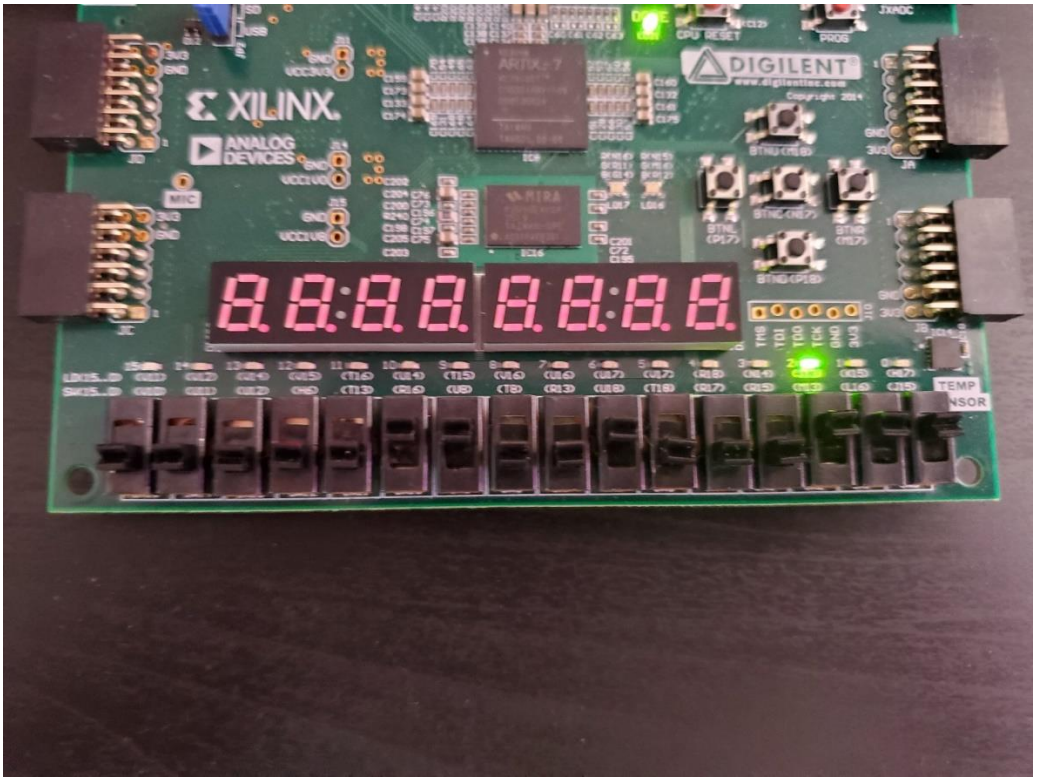
Student Name: Ricardo Garza
Student UNT email ID: ricardogarza3@my.unt.edu 10967208



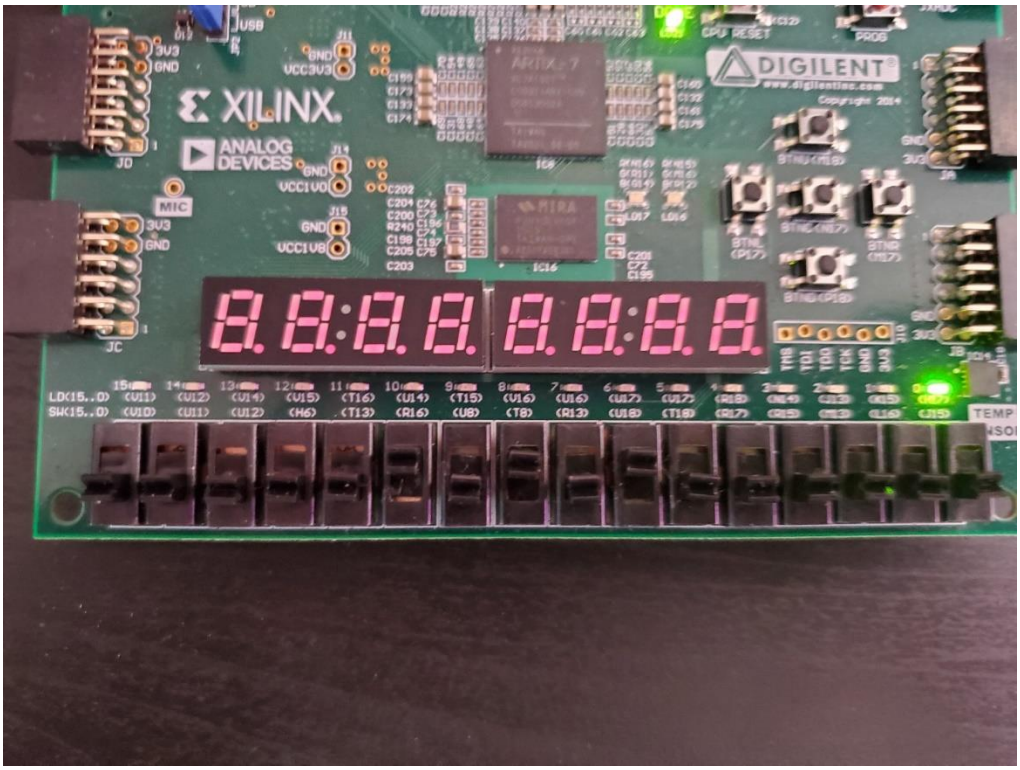


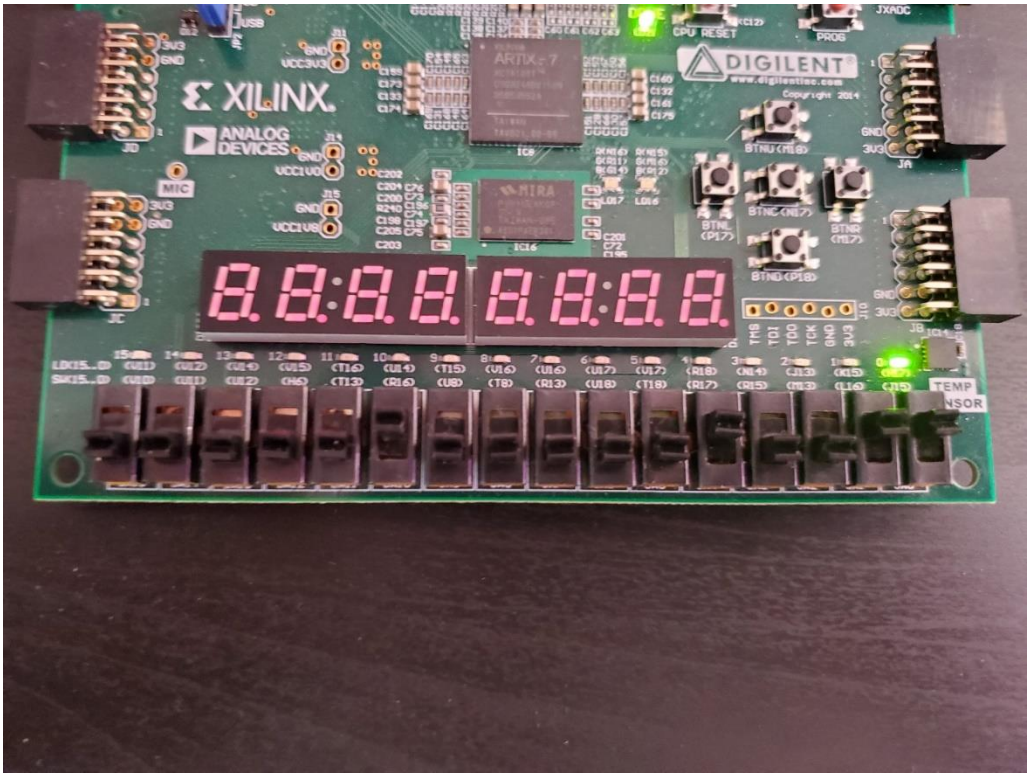
Student Name: Ricardo Garza
Student UNT email ID: ricardogarza3@my.unt.edu 10967208



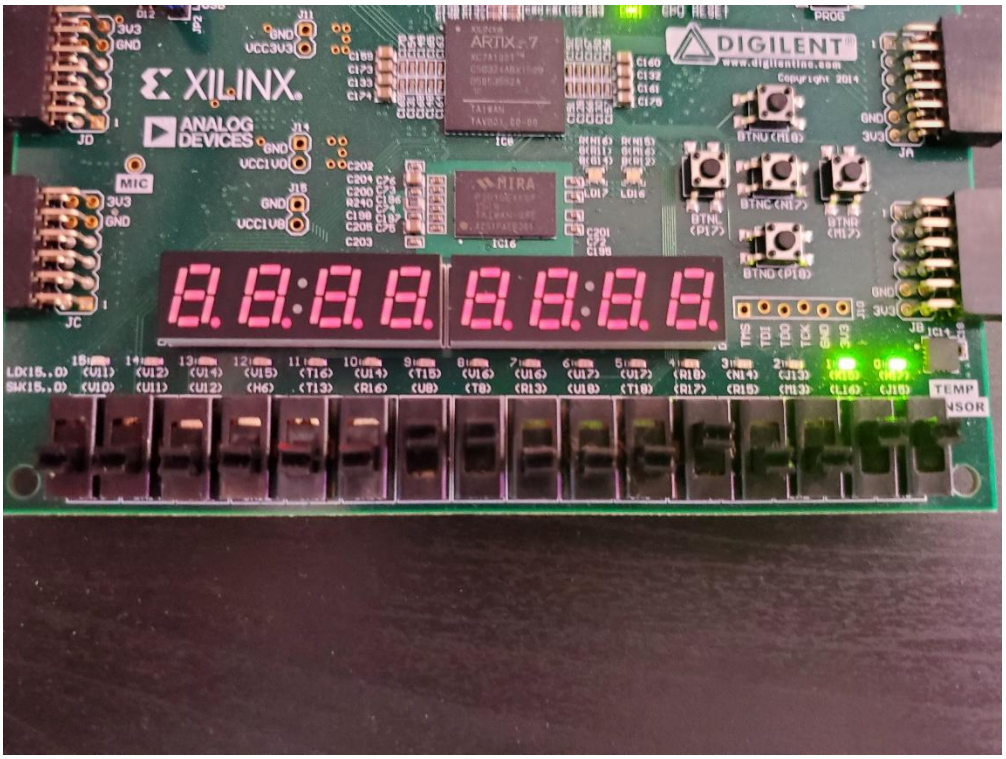


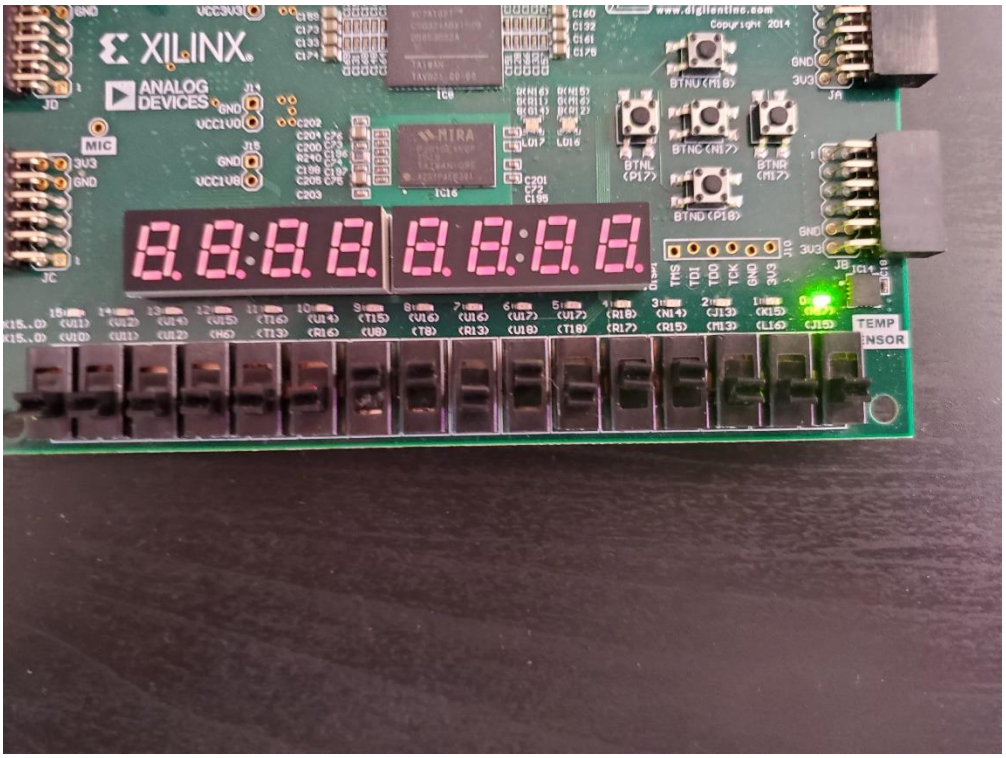
Student Name: Ricardo Garza
Student UNT email ID: ricardogarza3@my.unt.edu 10967208



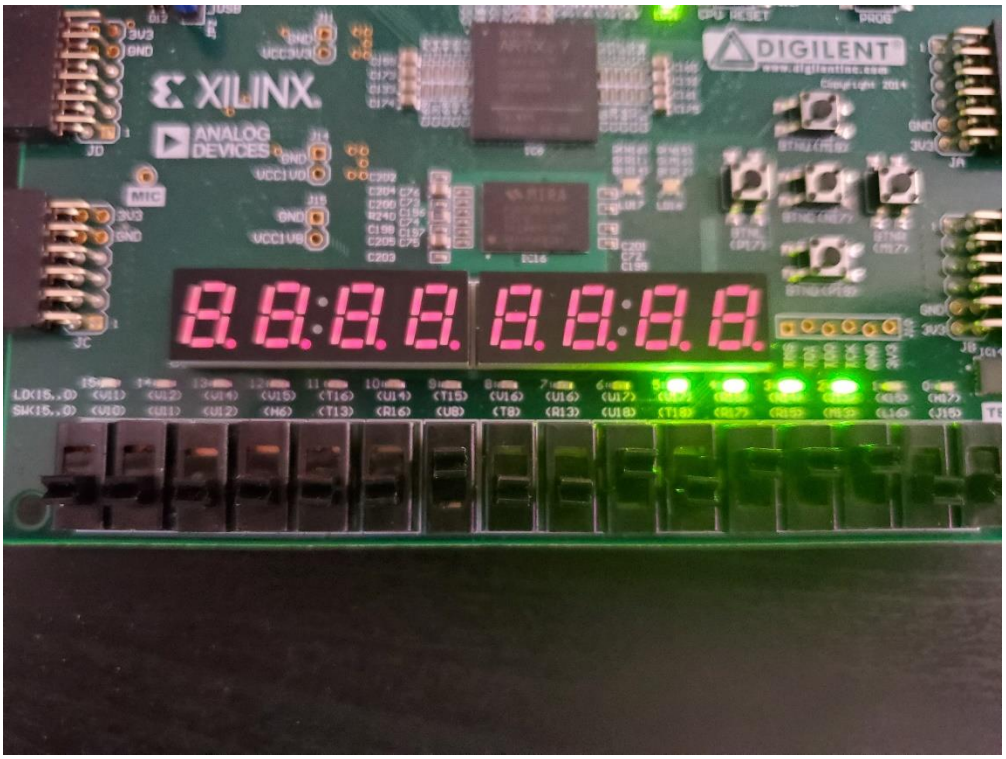


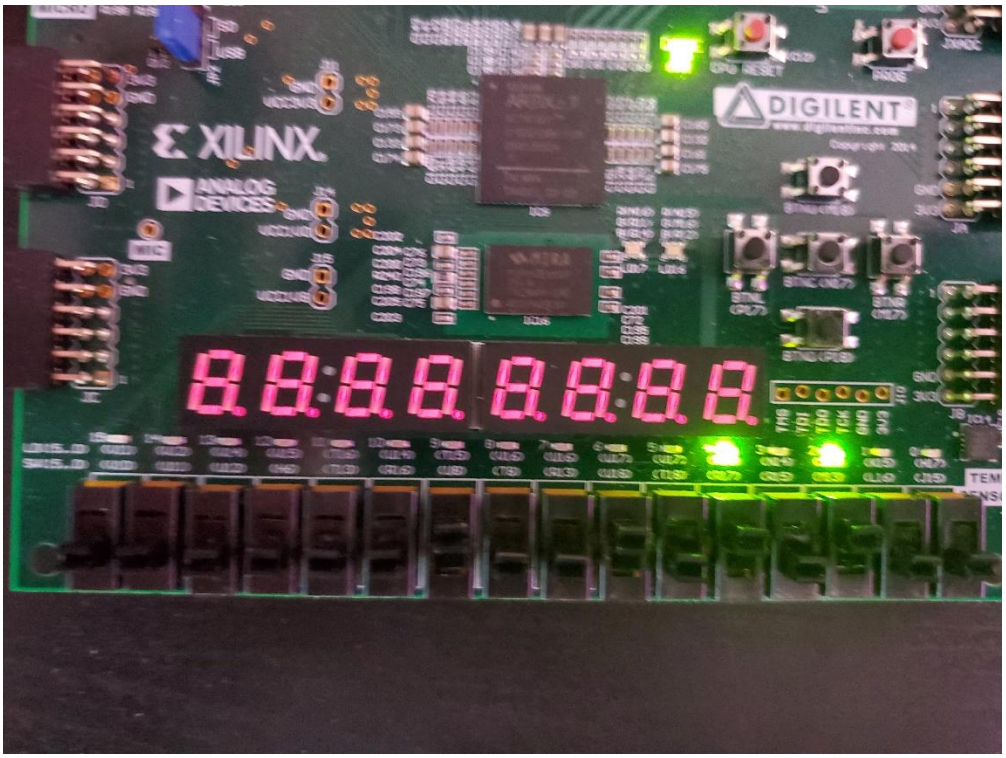
Student Name: Ricardo Garza
Student UNT email ID: ricardogarza3@my.unt.edu 10967208



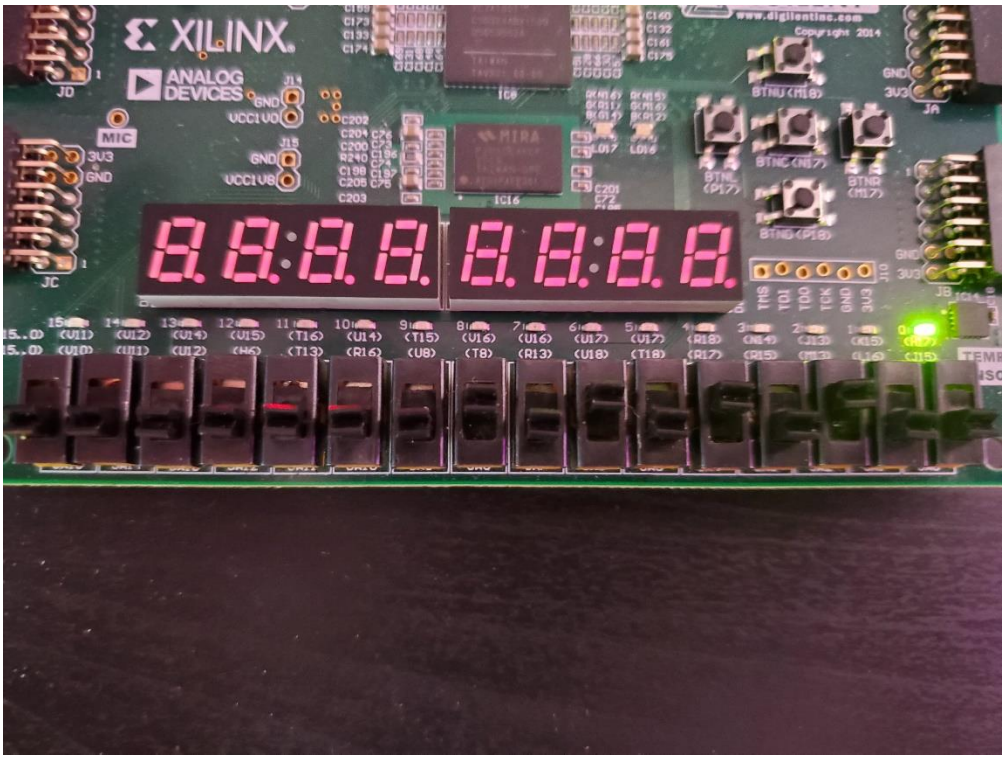


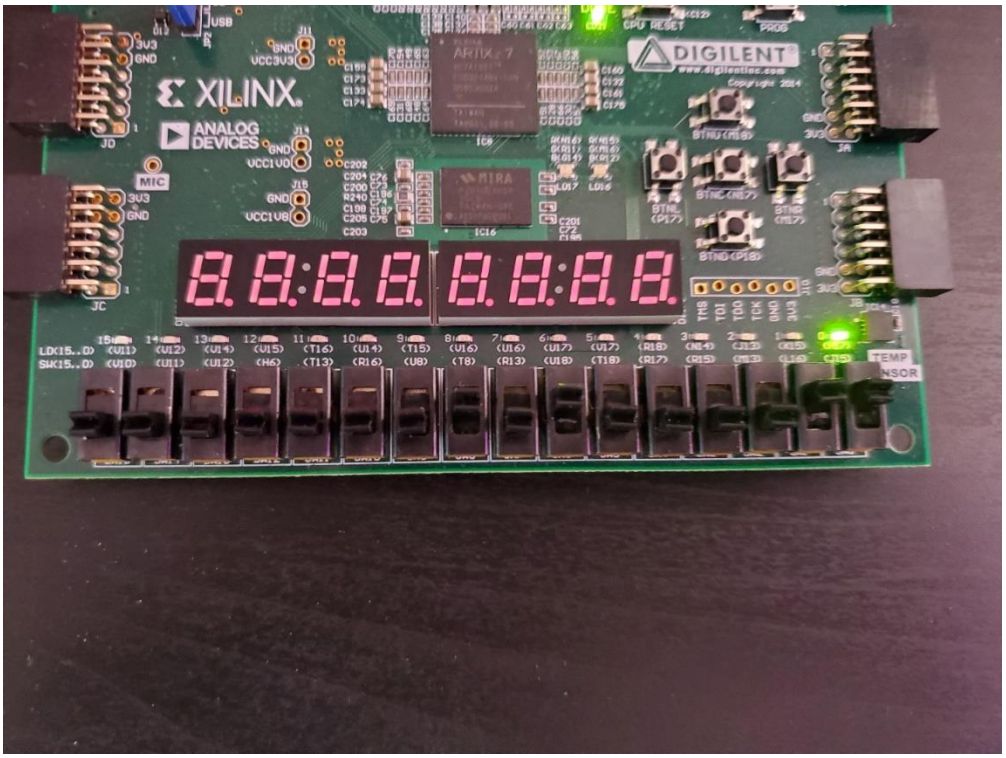
Student Name: Ricardo Garza
Student UNT email ID: ricardogarza3@my.unt.edu 10967208



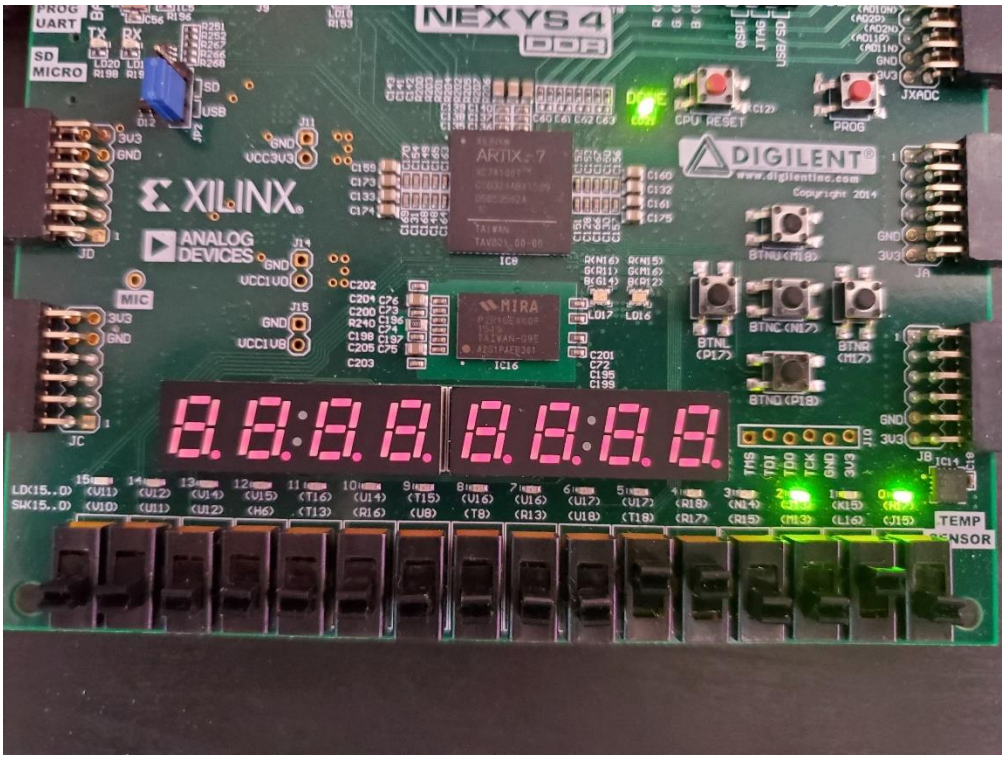


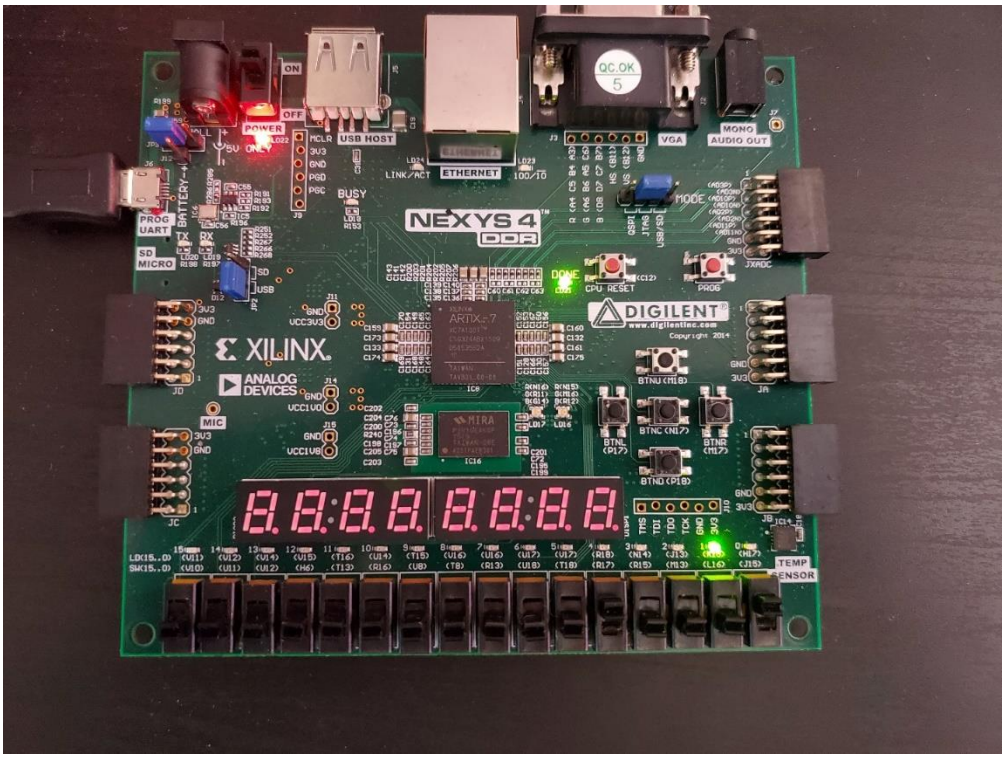
Student Name: Ricardo Garza
Student UNT email ID: ricardogarza3@my.unt.edu 10967208





Student Name: Ricardo Garza
Student UNT email ID: ricardogarza3@my.unt.edu 10967208





Student Name: Ricardo Garza
Student UNT email ID: ricardogarza3@my.unt.edu 10967208

Summary:
In this lab we learned how to implement an ALU with a clock. The ALU could do different operations and light up depending on the operations. The operations implemented are addition, division, subtraction, multiplication, greater than, less than, AND gate and OR gate. Each one had to take 4-bits and output an 8-bit number. Overall, the lab was more difficult than the first two labs. It was knowing how to implement a clock which was nothing something I was familiar with.

References:

1. https://stackoverflow.com/questions/17904514/vhdl-how-should-i-create-a-clock-in-a-testbench
2. https://allaboutfpga.com/vhdl-code-for-4-bit-alu/
3. https://www.fpga4student.com/2017/06/vhdl-code-for-arithmetic-logic-unit-alu.html
4. https://stackoverflow.com/questions/17904514/vhdl-how-should-i-create-a-clock-in-a-testbench
5. https://electronics.stackexchange.com/questions/148320/proper-clock-generation-for-vhdl-testbenches
6. https://vhdlguru.blogspot.com/2010/03/how-to-write-testbench.html
7. https://vhdlwhiz.com/clocked-process/
8. https://electronics.stackexchange.com/questions/337565/vhdl-alu-8-bit-register/337601