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

Requirements:

- Design a 4-bit UP-counter which counts from 0 through n and follows the sequence [0 → n^0+a → n^1+a → n^2+a → n^3+a → ….]. The start/reset state of the FSM is 0 and numbers n & a such that n, a > 0 and n > a always.
- Using behavioral modeling design the 4-bit UP-counter using different states of the FSM. Do not use any Multipliers or Adders to design the counter.
- Design a test bench which can show all the FSM states possible. Capture the waveforms and verify the results.
- UP-counter clock should be connected to switch (SW0), input number n(2-bit) should be connected to switches SW2(MSB) and SW1(LSB). Input number a(2-bit) should be connected to switches SW4(MSB) and SW3(LSB) on the Nexys 4 board. Connect reset signal to the CPU Reset push button on the Nexys 4 board. Outputs (n) should be connected to LED3(MSB) through LED0(LSB).

Learning Objectives:
The general learning objectives of this lab were to design an FSM. The FSM would be for a 4-bit UP counter that would be implemented on the Nexys 4 FPGA board

General Steps:

1. Design an UP counter that has two inputs, one output, a clock, and a reset
2. Design a testbench for all possible results of the FSM states
3. Update the constraint files with the correct switches
    a. Connect clock to the first switch SW[0]
    b. Connect reset to CPU reset button
    c. Connect inputs to switches SW[1] – SW[4]
4. Synthesis the program so that it is useable on Nexys board
5. Test the program on Nexys board

Detailed Steps:
Some detailed steps to complete this lab were

1. Design an UP-counter ports

```
entity lab4_src is
   Port ( clk: in std_logic;   -- clock input
          reset: in std_logic; --reset input
          n: in std_logic_vector(1 downto 0);
          a: in std_logic_vector(1 downto 0);
          counter: out std_logic_vector(3 downto 0) -- 4bit counter
          );
end lab4_src;
```

    a.
2. Design the logic for the up counter

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

```
elsif(rising_edge(clk)) then
  if n = "10" and a = "01" then
     if count_up = "0000" then
         count_up <= "0010";
     elsif count_up = "0010" then
         count_up <= "0011";
     elsif count_up = "0011" then
         count_up <= "0101";
     elsif count_up = "0101" then
         count_up <= "1001";
     elsif count_up = "1001" then
         count_up <= "0000";
     else
         count_up <= "0000";
     end if;
  elsif n = "11" and a = "01" then
     if count_up = "0000" then
         count_up <= "0010";
     elsif count_up = "0010" then
         count_up <= "0100";
     elsif count_up = "0100" then
         count_up <= "1010";
     elsif count_up = "1010" then
         count_up <= "0000";
     else
         count_up <= "0000";
```

a.

3. Afterwards design the testbench
   a. Make sure that the ports match in the component

```
component lab4_src
  PORT(   clk: in std_logic;    -- clock input
          reset: in std_logic; --reset input
          n: in std_logic_vector(1 downto 0);
          a: in std_logic_vector(1 downto 0);
          counter: out std_logic_vector(3 downto 0) -- 4bit counter
          );
end component;
```

   b.
4. Afterwards run the simulation
5. When the simulation runs correctly attach the constraints given
6. Make sure to uncomment the lines needed for this lab
7. Make sure the ports, clock, and reset are attach to the correct switches and buttons

```
14 set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
15 set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { n[0] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
16 set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { n[1] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
17 set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { a[0] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
18 set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports { a[1] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
19 #set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports { SW[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
```

a.

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

b.

```
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { counter[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { counter[1] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { counter[2] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { counter[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { LED[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
```

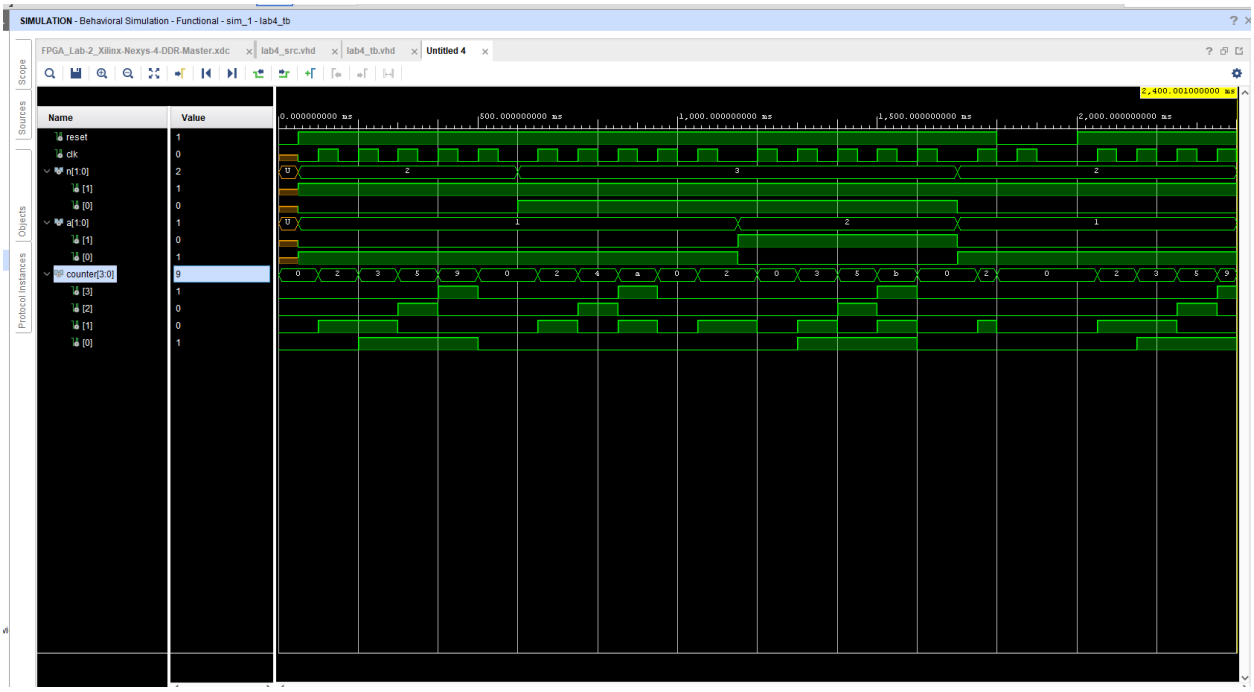
c.

```
set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { reset }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
```

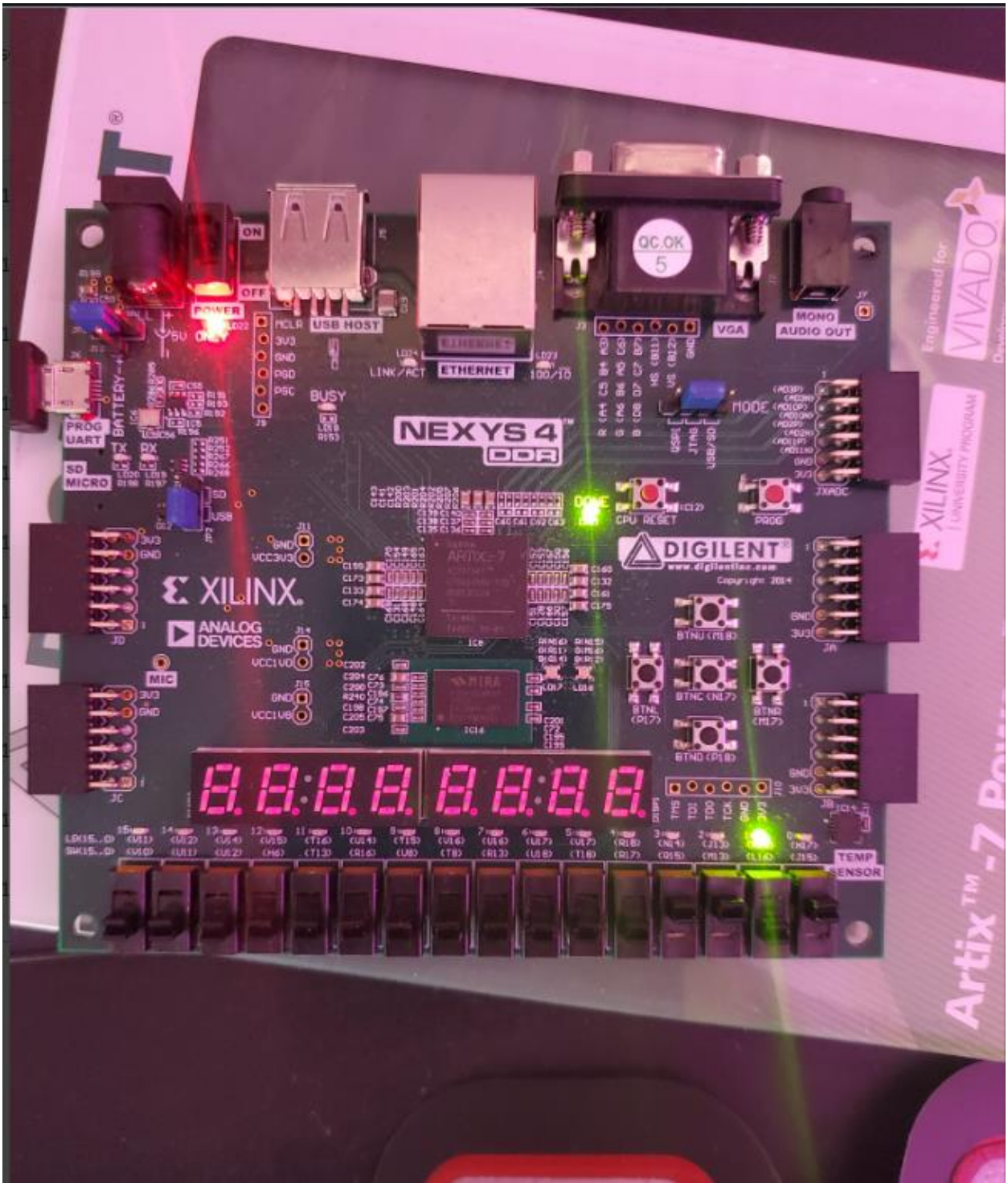
8. Run the synthesis, implementation and generate bitstream a.
    a. Make sure to fix any errors generated along the way
9. Run on the Nexys 4 FPGA board

Observations:

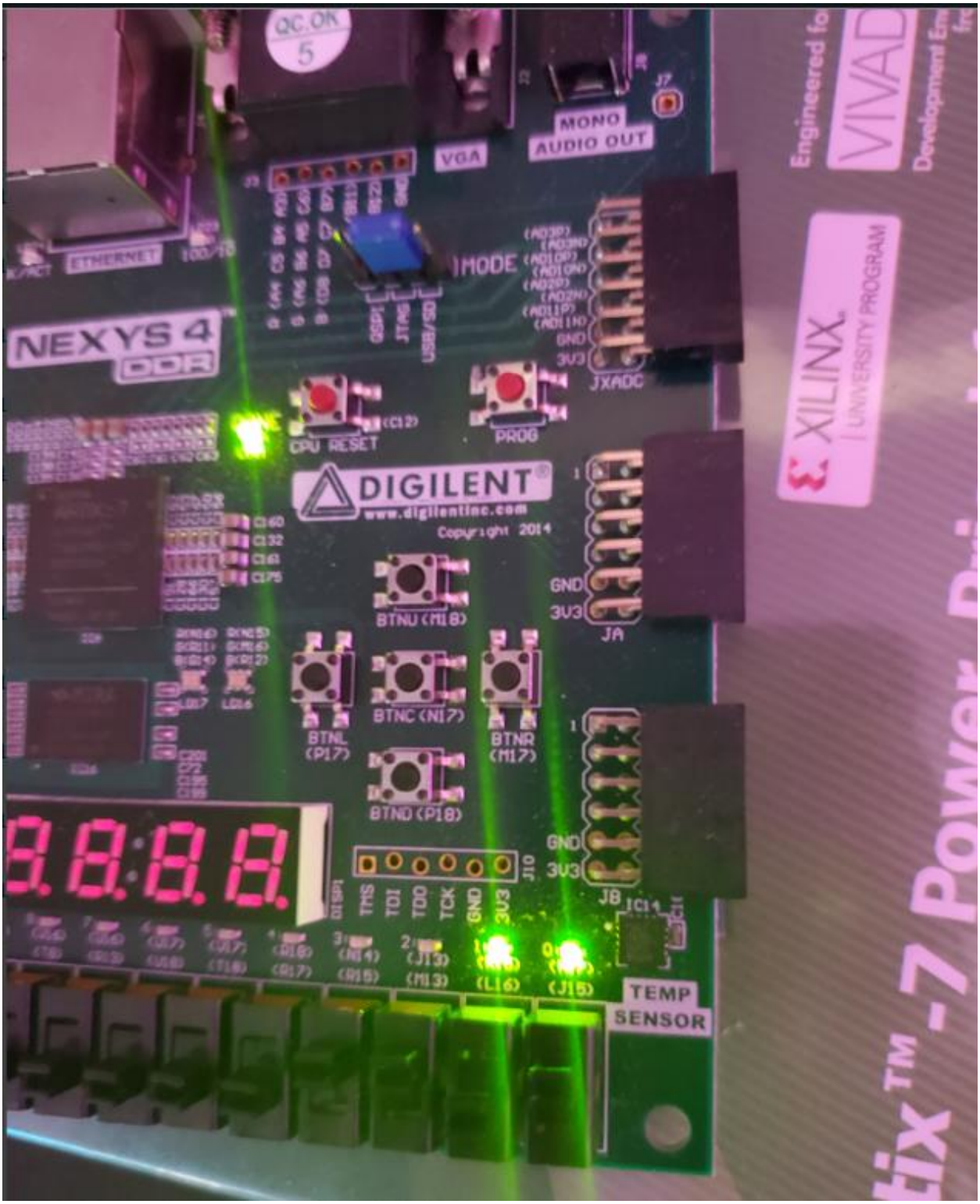
Compared to the other labs this one was more challenging because of what was needed. I thought just counting up would be enough, but it was not. There is a specific pattern that was needed and not just a counter going from 0 to 15.



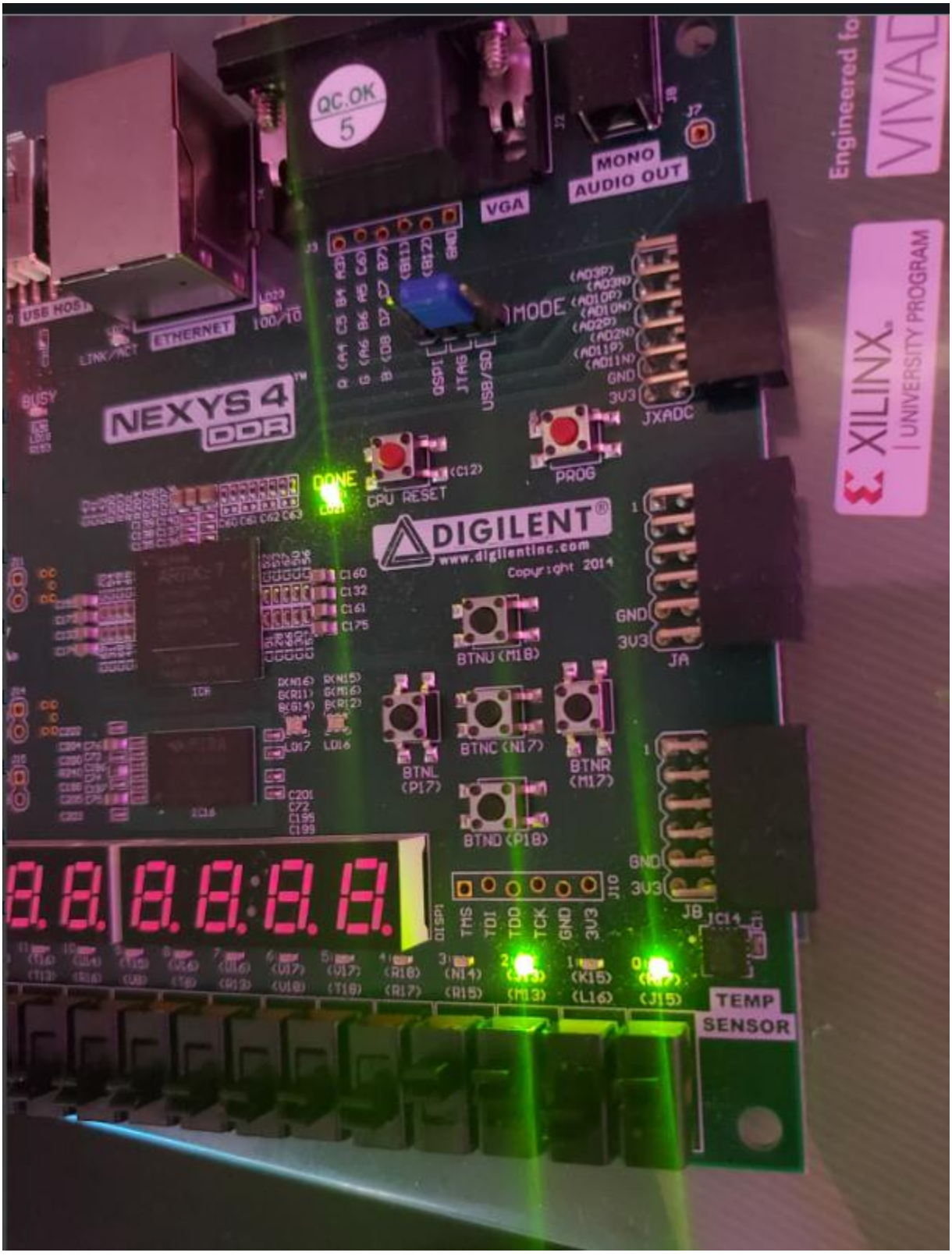
Student Name: Ricardo Garza
Student UNT email ID: 10967208 ricardogarza3@my.unt.edu



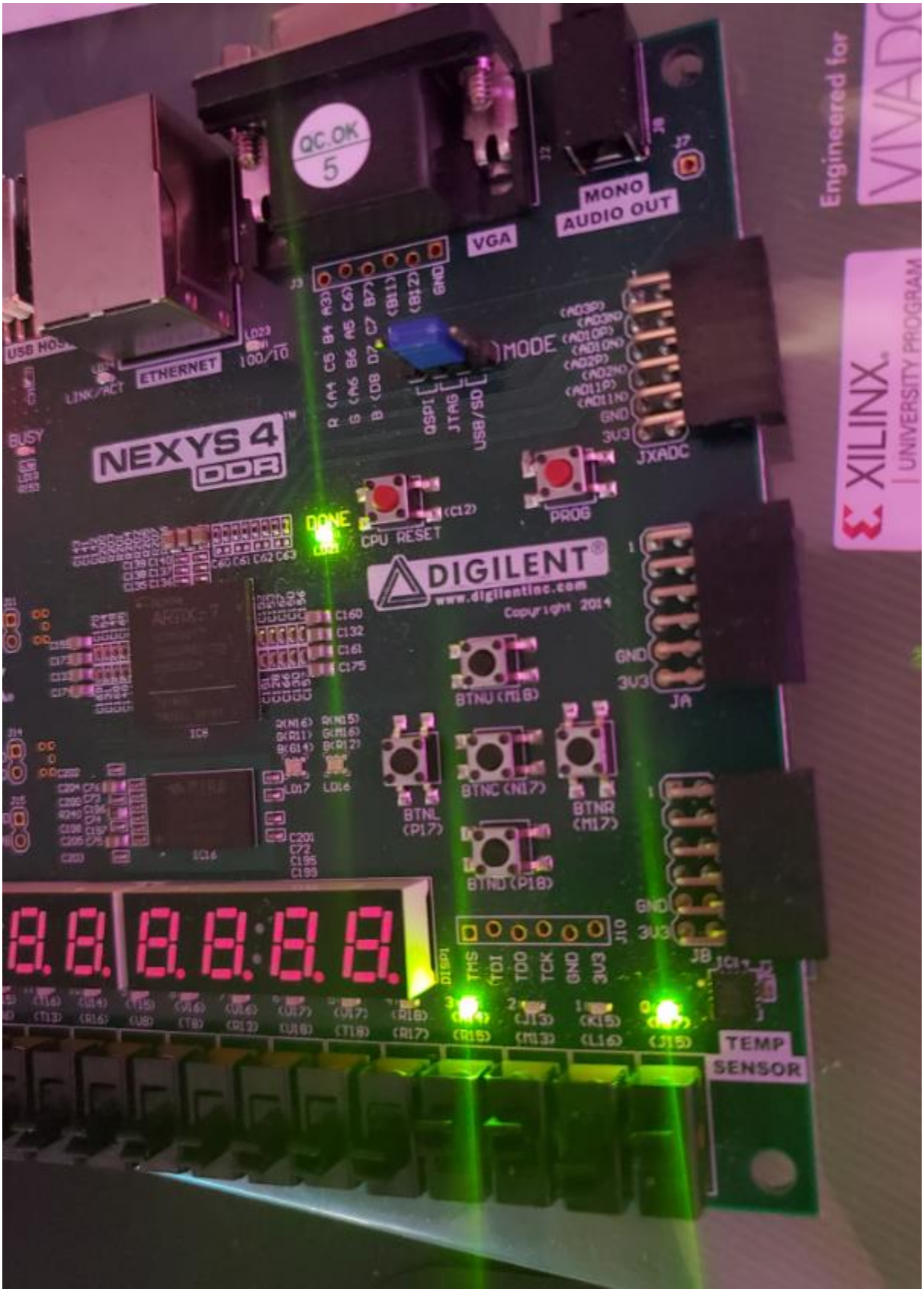
Student Name: Ricardo Garza
Student UNT email ID: 10967208 ricardogarza3@my.unt.edu



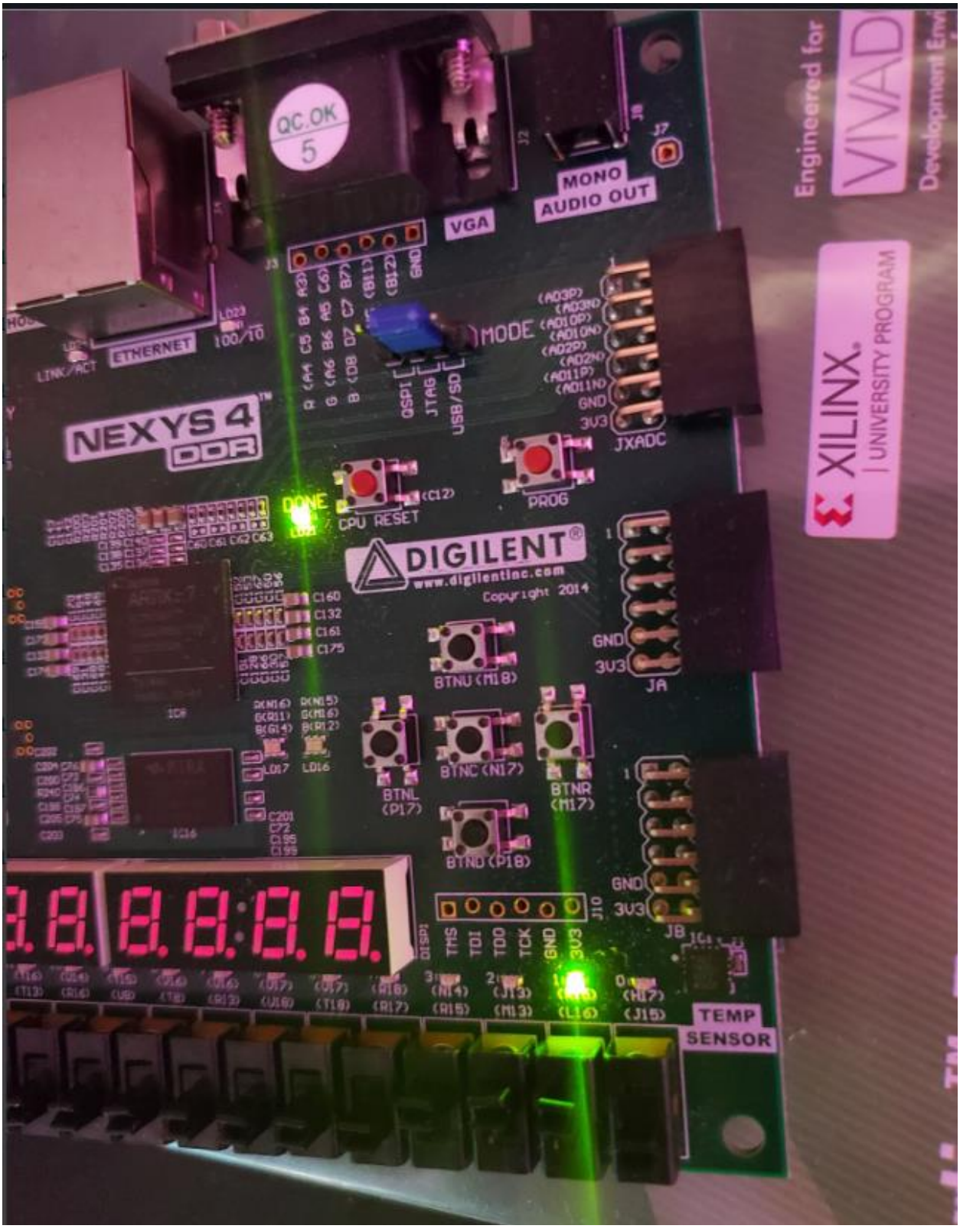
Student Name: Ricardo Garza
Student UNT email ID: 10967208 ricardogarza3@my.unt.edu



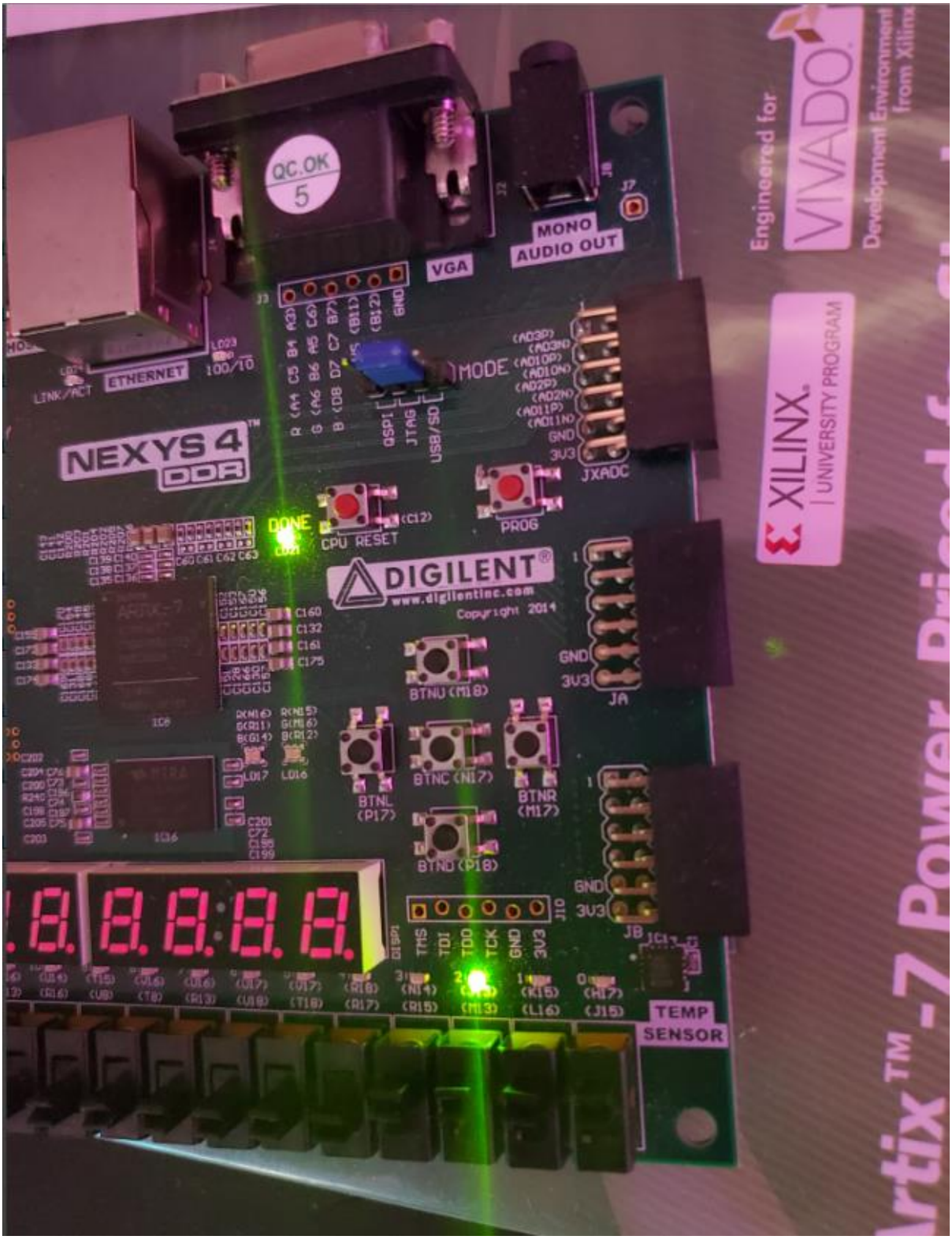
Student Name: Ricardo Garza
Student UNT email ID: 10967208 ricardogarza3@my.unt.edu



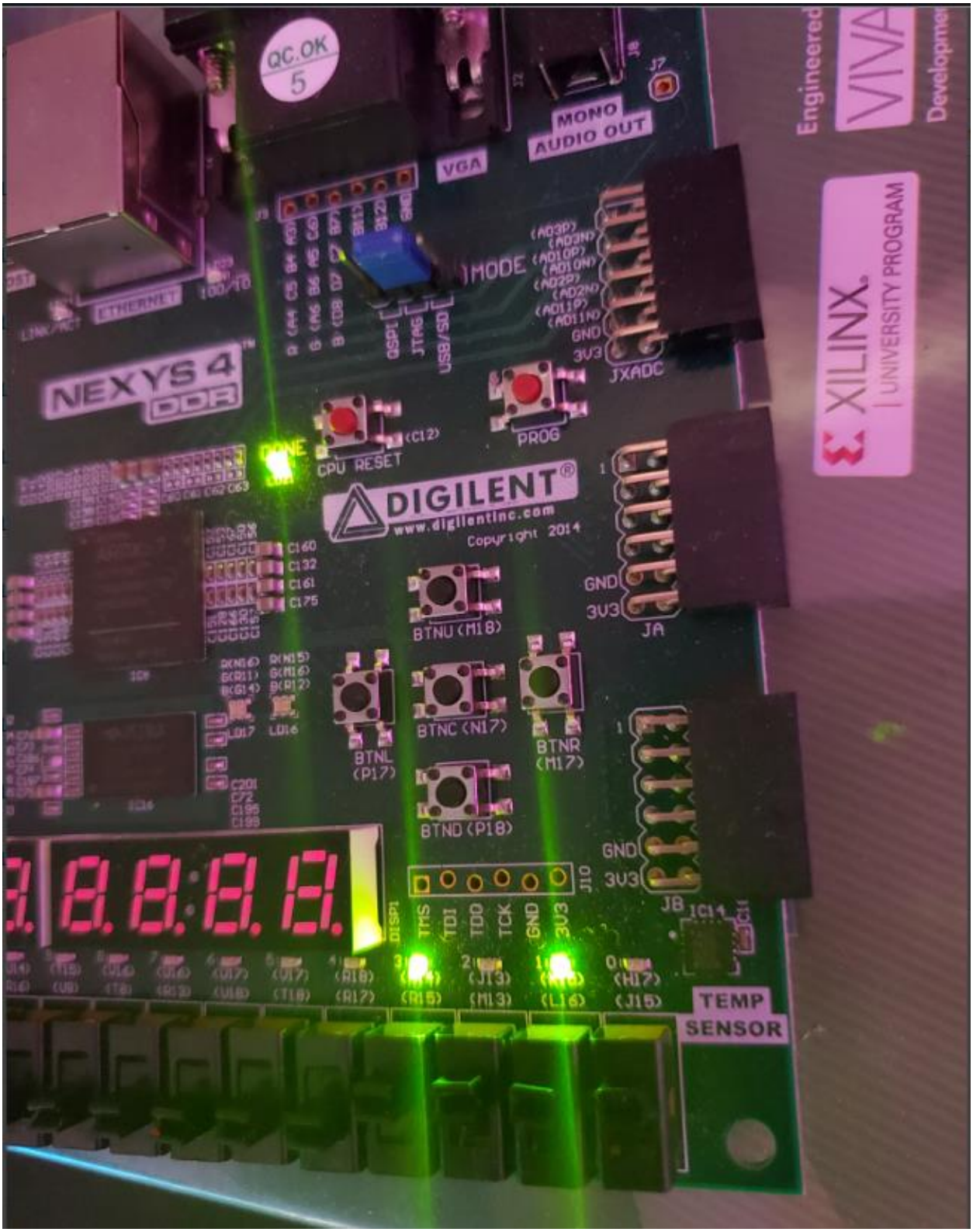
Student Name: Ricardo Garza
Student UNT email ID: 10967208 ricardogarza3@my.unt.edu



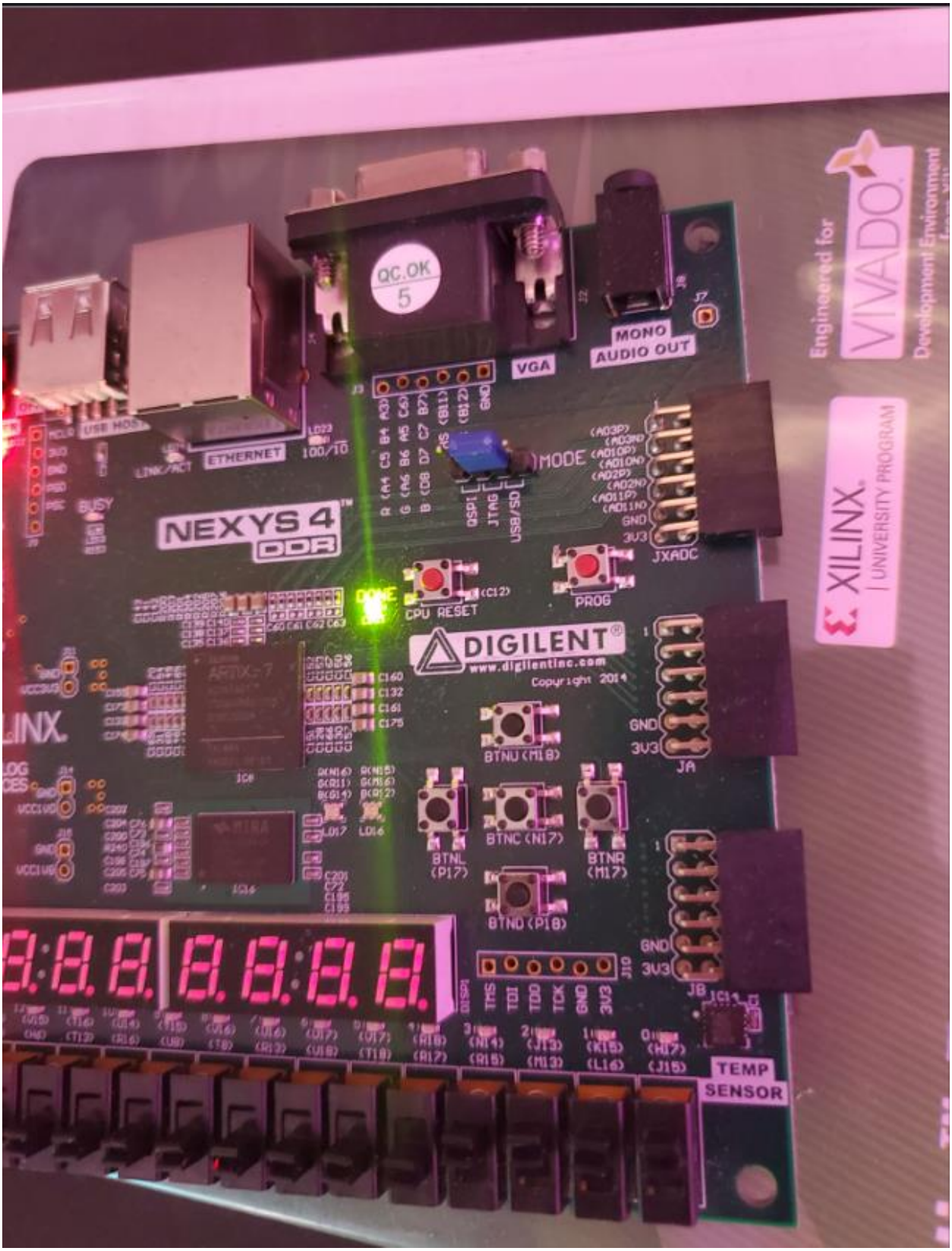
Student Name: Ricardo Garza
Student UNT email ID: 10967208 ricardogarza3@my.unt.edu



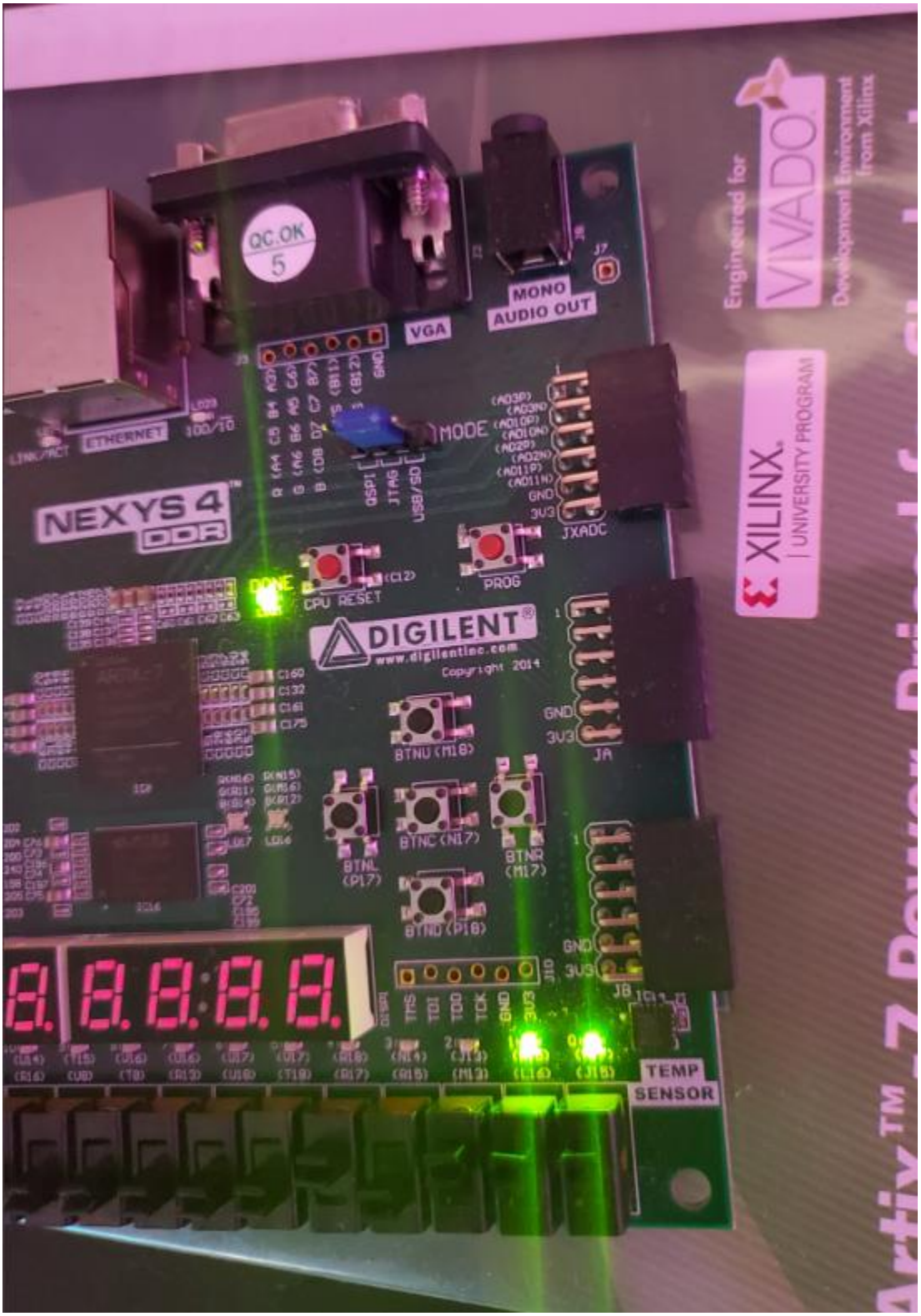
Student Name: Ricardo Garza
Student UNT email ID: 10967208 ricardogarza3@my.unt.edu



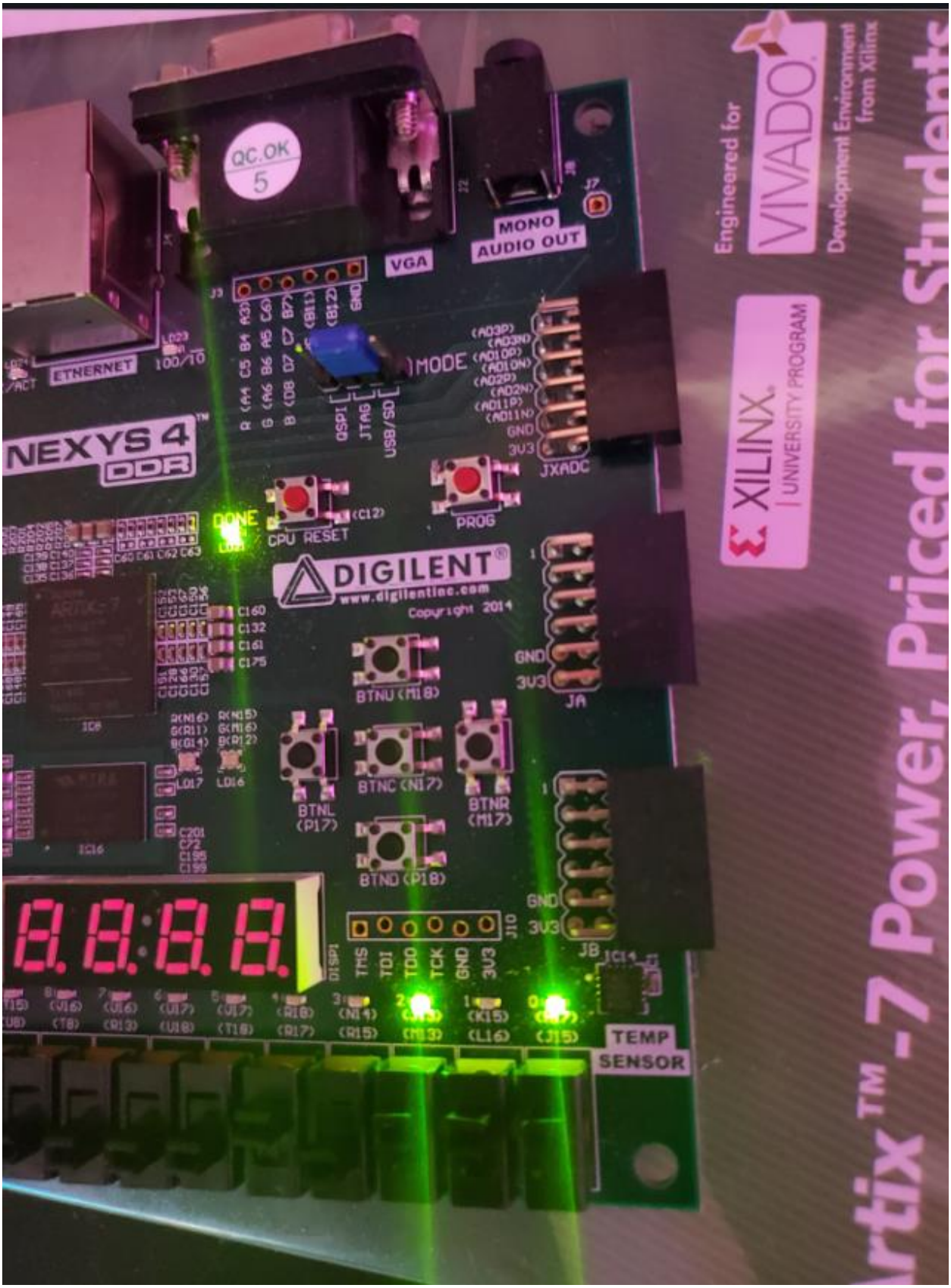
Student Name: Ricardo Garza
Student UNT email ID: 10967208 ricardogarza3@my.unt.edu



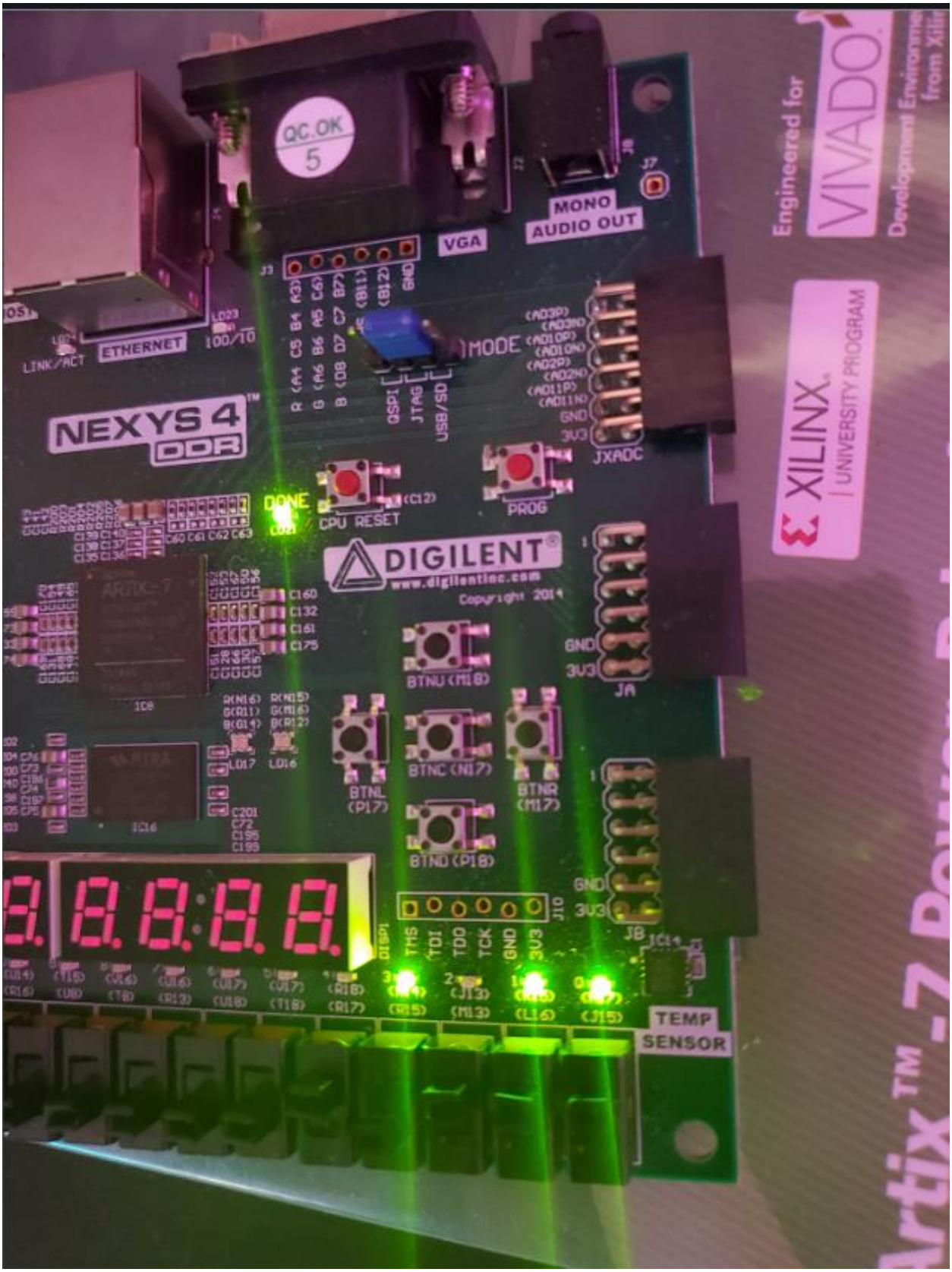
Student Name: Ricardo Garza
Student UNT email ID: 10967208 ricardogarza3@my.unt.edu



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



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



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

Summary:
In this lab we learned how to design a sequence on VHDL. The sequence would count using two 2-bit numbers to output a 4-bit number. We had to learn how to properly design a reset that would work on an FPGA board properly along with a clock on a switch. I thought it would be more difficult to pass it over to the FPGA, but it turned out to be simple. I did notice that you had to delete it to the zero assigned to the first switch or else the switch doesn't do anything, and it doesn't compile.

References:

- https://www.ics.uci.edu/~jmoorkan/vhdlref/ifs.html
- https://www.fpga4student.com/2017/06/vhdl-code-for-counters-with-testbench.html
- https://www.ics.uci.edu/~jmoorkan/vhdlref/sig_dec.html
- https://allaboutfpga.com/vhdl-code-for-4-bit-binary-counter/