
OAKLAND
UNIVERSITY

Traffic Light Controller

ECE 2700 Digital Logic Design
Instructor: Dr. Daniel Llamocca

List of Authors:

Joseph Hill

Aldo Gega

John Brooks

Nashwan Marcos

Electrical and Computer Engineering Department

School of Engineering and Computer Science

Oakland University, Rochester, MI

Content

- Introduction
- Design and Implementation
- Vivado Schematic
- VHDL Code Samples
- I/O Circuit Implementation
- Issues Encountered
- Demo
- Improvement
- Q&A

Introduction

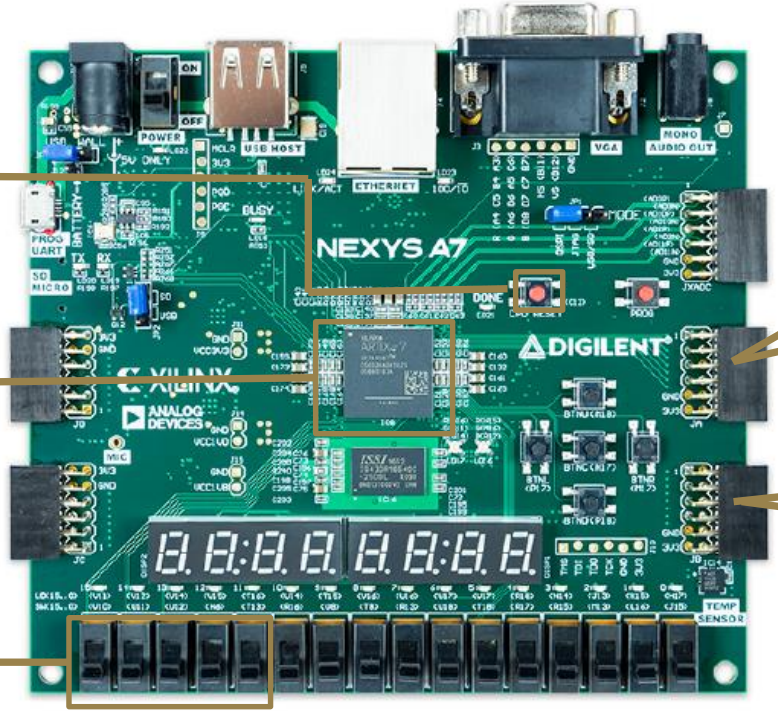
The intention of most school related project is to challenge students with real world applications intended to further their knowledge and enhance their skills. The intentions for this project are to explore the design and implementation of a 4-way traffic light controller. This involved the decision of a project idea, planning and capturing the thought processes from what we have learned and what we were able to bring to the table from our individual experience. The most challenging part in the design process is the control of the different states using counters, LUTs and the logic behind it. Our design should be a basic prototype of a working system with room for improvements. We can't avoid traffic jams and busy intersections, but we can always add smarter logic systems that will aid in traffic flow making it smarter, saver and more adaptable to growing infrastructure.

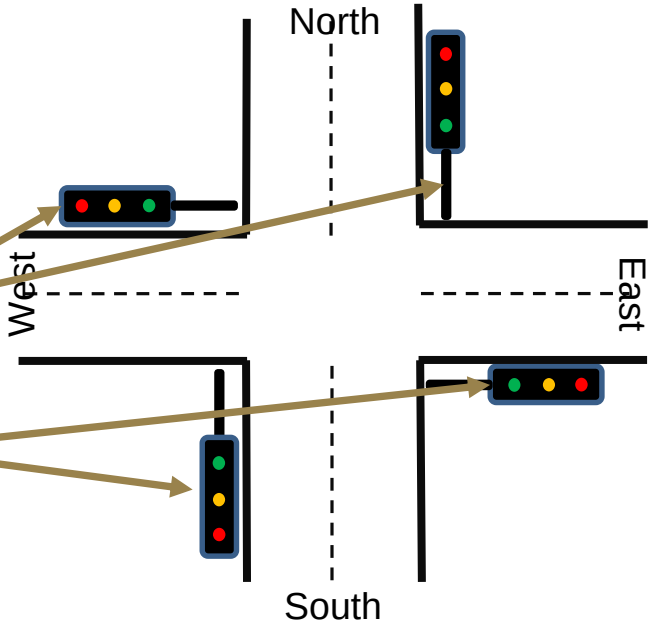
Design & Implementation Overview

Board reset button

VHDL , Modulus Counters, LUTs programmed to FPGA

User timing control for green timing phase

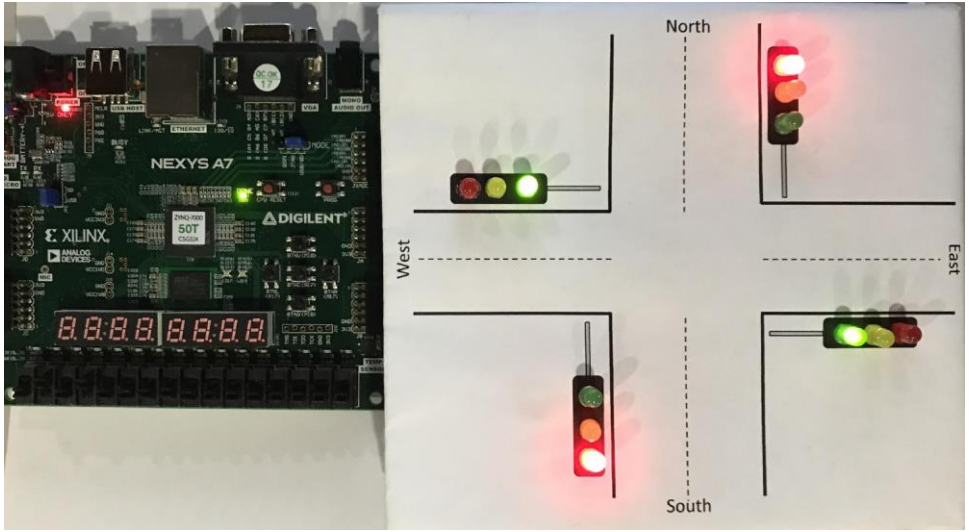




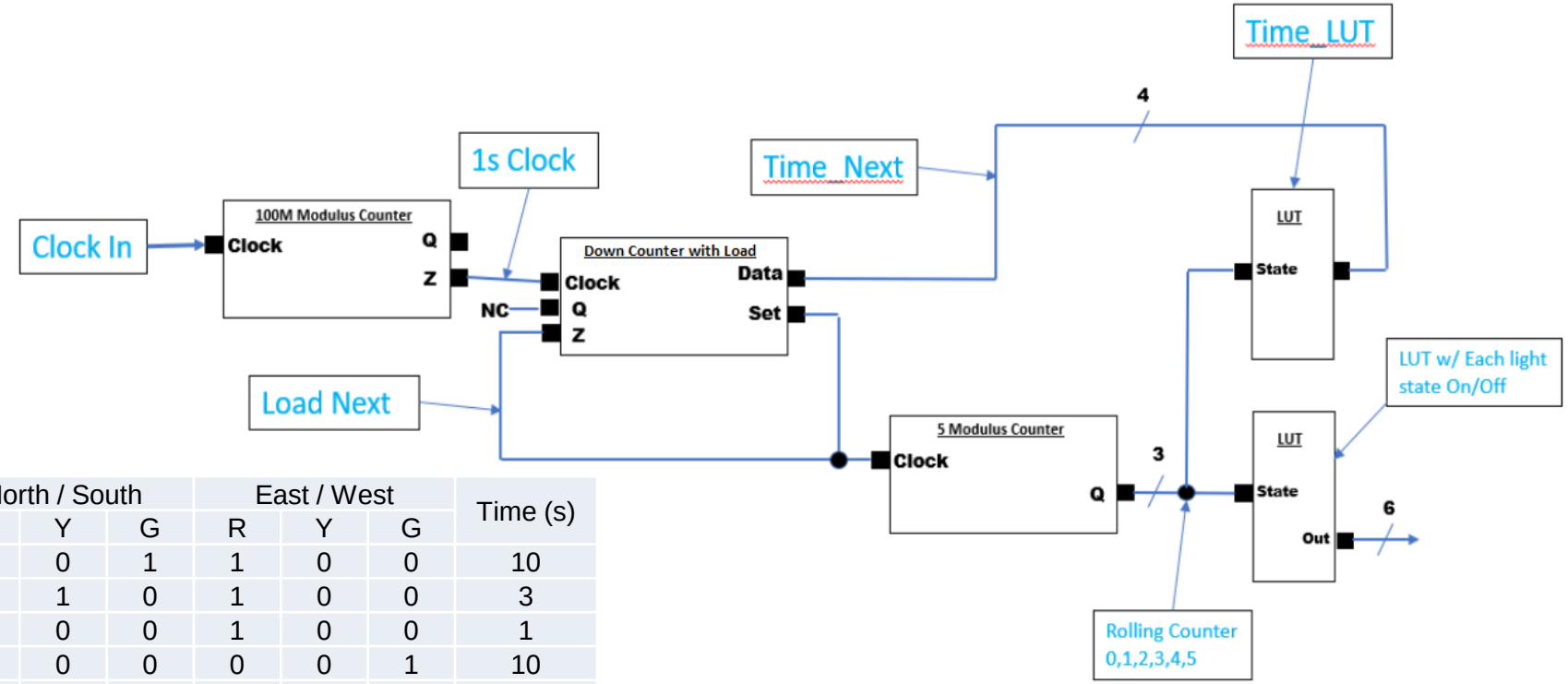
Implementation -Top Level

The design was implemented on a Nexys 7 FPGA and tested on both the *XC7A100T* and *XC7A50T* models. An external circuit with LEDs soldered to it was used to demonstrate the behavior of the design and provide a visual representation of the intended concept.

Our VHDL approach was to create one main top file with three sub modules. The first module brings the clock down from 100 MHz to 1 Hz. The second module counts the time for each cycle. The third module has a rolling counter to loop through the different stages. LUTs are used to control the outputs and the time for next stage

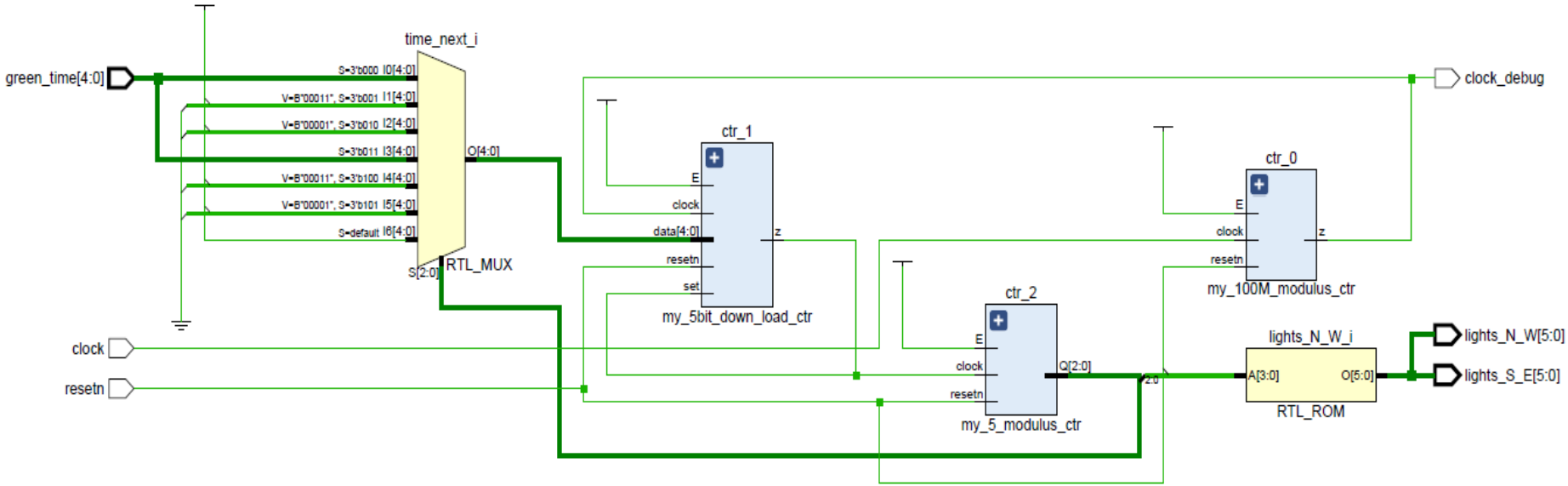


Logic Design – Top Level



North / South			East / West			Time (s)
R	Y	G	R	Y	G	
0	0	1	1	0	0	10
0	1	0	1	0	0	3
1	0	0	1	0	0	1
1	0	0	0	0	1	10
1	0	0	0	1	0	3
1	0	0	1	0	0	1

Vivado Schematics



VHDL Code (Samples)

OAKLAND UNIVERSITY

```
entity my_100M_modulus_ctr is
    generic (COUNT: INTEGER:= (10**8)); -- (10**8)/2 cycles of T = 10 ns --> 0.5 s
    port (clock, resetn, E: in std_logic;
          Q: out std_logic_vector ( integer(ceil(log2(real(COUNT)))) - 1 downto 0);
          z: out std_logic);
end my_100M_modulus_ctr;

architecture Behavioral of my_100M_modulus_ctr is
    constant nbits: INTEGER:= integer(ceil(log2(real(COUNT))));
    signal Qt: std_logic_vector (nbits -1 downto 0);
begin

    process (resetn, clock)
    begin
        if resetn = '0' then
            Qt <= (others => '0');
        elsif (clock'event and clock = '1') then
            if E = '1' then
                if Qt = conv_std_logic_vector (COUNT-1,nbits) then
                    Qt <= (others => '0');
                    z <= '1';
```

```
--Down clocks the 100MHz clock to 1Hz
ctr_0: my_100M_modulus_ctr port map( clock => clock,
                                     resetn => resetn,
                                     E => ctr_0_enable,
                                     z => one_sec_clock); --For Test bench unmap this signal


--Counts down the time for each cycle
ctr_1: my_5bit_down_load_ctr port map(   clock => one_sec_clock,
                                         z => load_next,
                                         set => load_next,
                                         data => time_next,
                                         resetn => resetn,
                                         E => ctr_1_enable);

--Loops through the light cycle "stages"
ctr_2: my_5_modulus_ctr port map(    clock => load_next,
                                     E => ctr_2_enable,
                                     resetn => resetn,
                                     Q => stage);
```

I/O Circuit Implementation

Pmod ports JA & JB were used as VCC and logic signals to the LEDs. Each 12-pin Pmod port provides two 3.3V VCC signals (pins 6 and 12), two Ground signals (pins 5 and 11), and eight logic signals, as shown in Figure 1. The VCC and Ground pins can deliver up to 1A of current. Pmod logic signals were used to provide a logic low to the LEDs as they have a built-in pull down resistor on the Nexys A7 board as shown in figure 2 below. The current draw on each LED was 6.4 mA.

Figure 1: Pmod Connectors; Front View, as Loaded on PCB

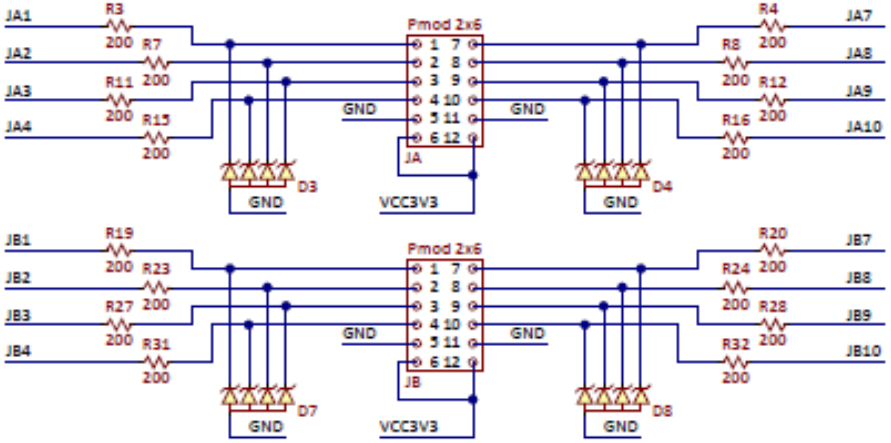


Figure 2: Pmod Connectors; Front View, as Loaded on PCB

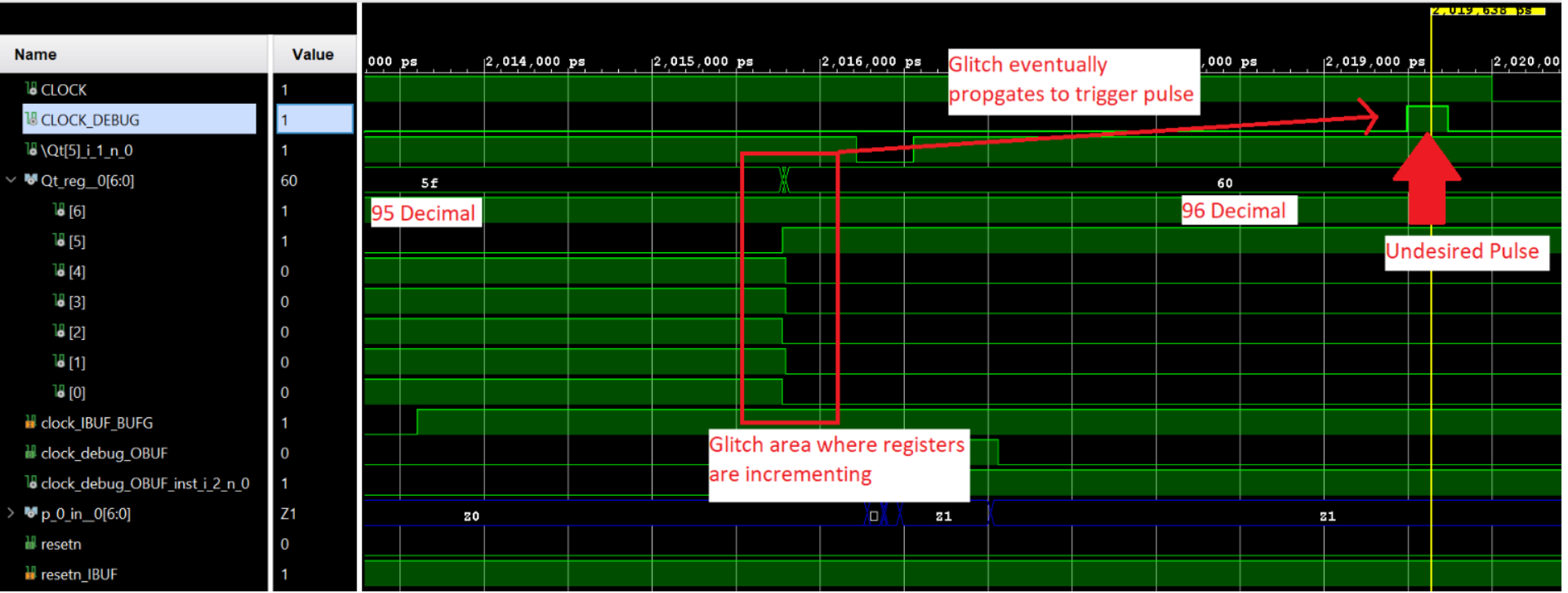
Issues Encountered

Below are some of the issues we encountered:

1. Pmod PINs are assigned differently from the original XDC file. According to the manual, Pmod data signals are not matched pairs, and they are routed using best available tracks without impedance control or delay matching.

2. Major Issue: 100 MHz to 1 Hz modulus counter bug...

Issues Encountered



Issues Encountered

Why does this bug occur?

The output of the counter (z) is directly wired to the registers that store the count Qt. When Qt gets incremented there is a small amount of time in which the value of Qt is in an undefined state as the registers update at very slightly different times. Similar to the issues that Gray code is used to solve. When this occurs it's possible for the registers to trigger the output z before the modulus value.

This is resolved by moving the assignment of output z into the synchronous logic within the clock event process. As seen in the comments here.

```
architecture Behavioral of my_100M_modulus_ctr is
    constant nbits: INTEGER:= integer(ceil(log2(real(COUNT))));
    signal Qt: std_logic_vector (nbits -1 downto 0);
begin

    process (resetn, clock)
    begin
        if resetn = '0' then
            Qt <= (others => '0');
        elsif (clock'event and clock = '1') then
            if E = '1' then
                if Qt = conv_std_logic_vector (COUNT-1,nbits) then
                    Qt <= (others => '0');
                    --z <= '1';
                else
                    Qt <= Qt + conv_std_logic_vector (1,nbits);
                    --z <= '0';
                end if;
            end if;
        end if;
    end process;

    z <= '1' when Qt = conv_std_logic_vector (COUNT-1,nbits) else '0';
    Q <= Qt;

end Behavioral;
```

Improvements

Below are some of the suggested improvements:

1. Smart detection of traffic flow by means of machine vision. This will require the addition of smarter logics to aid in traffic flow.

1. Closed form feedback to check on functionality in case the traffic lights failed due to external conditions (Weather, Power issues…etc).

1. Fail safe mode by flashing red on all lights in case there is an issue within the system.

1. Application specific implementation, for example, night mode to flash yellow on all four traffic lights.


OAKLAND
UNIVERSITY™