Traffic Light Circuit

Marko Gjorgjievski Adam Algallaf

Maximilian Schaefer

28th June 2020

[4]

Mentor: Christopher Bonenberger Hochschule Ravensburg-Weingarten

Semester: Summer 2020

Contents

1 Introduction

Traffic lights are signaling devices that we encounter all the time every day that are used to regulate traffic flow, as given in [3], it is almost impossible to go anywhere without having to cross a traffic signal.

Without these signaling devices, everyday traffic would not be able to operate as efficiently as it does right now, and it would result in utter chaos on the roads and accidents. We encounter traffic lights daily, yet we rarely stop and question ourselves how do traffic lights function and work. As we almost always take this box with 3 distinct light bulbs tied with a simple circuit for granted, forgetting the crucial yet subtle role it plays in our transportation.

That is why it sparked the curiosity of my team and that is the reason why we choose to explore and research the traffic light circuit with the intent of making one.

2 Traffic Light: Type

As we all know there are numerous types of traffic lights that are used in specific situations on the roads today. Each type is programmed to act according to the road or intersection that it is placed on. As we aimed primarily towards simplicity, we choose to code a traffic light controller circuit which is independent of the placement and stops or allows traffic or pedestrians crossing and with that included, a crossing button which when pressed forces the traffic light to change states and prepare to stop traffic and allowing pedestrians to cross. As a result, we have coded (in VHDL) and simulated a single traffic light controller circuit in ModelSim. The expected behavior of the traffic light is as follows:

- To simultaneously stop traffic flow and pedestrians crossing for 30 seconds. (Two red lights for 30 sec.)
- Holding traffic for 3 more seconds before allowing **only** traffic to flow. (Yellow road, red pedestrian for 3 sec.)
- Allowing traffic to flow while still holding pedestrians for 27 seconds. (Green road, red pedestrian for 27 sec.)
- Preparing to stop the traffic flow in the following 3 seconds. (Yellow road, red pedestrian for 3 sec.)
- Halting traffic flow and preparing to allow pedestrians to cross which lasts 5 seconds. (Two red lights 5 sec.)

- Still holding traffic while allowing pedestrians to cross for 10 seconds. (Red road, green pedestrian for 10 sec.)
- Looping the process described above until the simulation reaches the end of run time.
- Looping the same process until the crossing button which in turn forces the traffic light to change to the appropriate state.

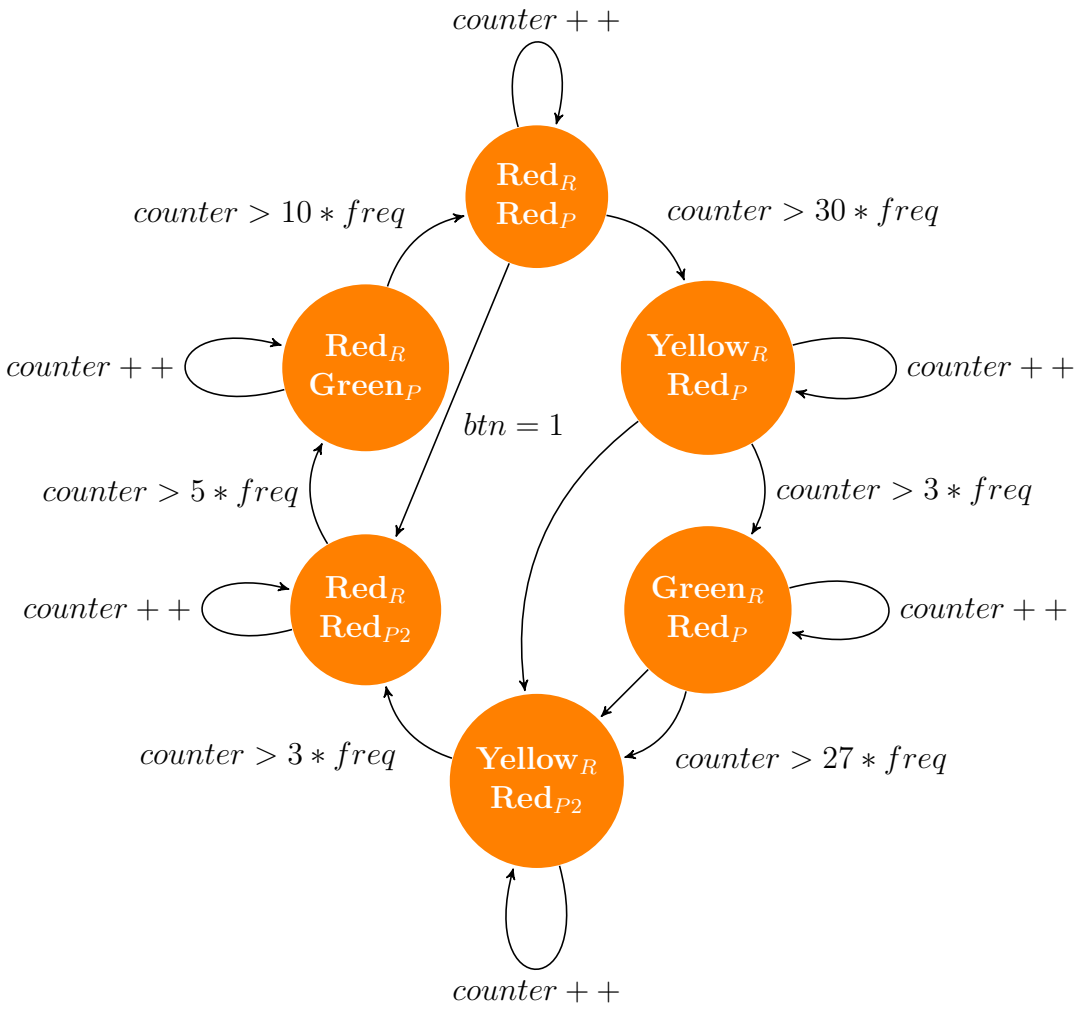

Figure 1: State flow graph of the traffic light states

To skip the implementation and see the simulation immediately please refer to section 4.

[0]"Counter++" alludes to the incrementation of the counter signal by one or $counter = counter + 1$.

3 VHDL Implementation

In this section we are going to describe the architecture and behavior of the traffic light circuit written in VHDL.

3.1 VHDL: Traffic Light Module

We begin with the module where the most important processes are present. Inside the file `traffic_light_circuit.vhdl`.

```
library ieee;
use ieee.std_logic_1164.all;
entity traffic_light_circuit is
        generic ( clkFrequency : integer );
        port(
        clk        : in  std_logic;
        btn        : in  std_logic;
        rst_n      : in  std_logic;
        Red_R      : out std_logic;
        Yellow_R   : out std_logic;
        Green_R    : out std_logic;
        Red_P      : out std_logic;
        Green_P    : out std_logic);
end entity;
```

Figure 2: Module: Entity

In the figure 2 we can see that we have defined inputs for the clocking process that we require for the circuit to work properly. We have defined ports for the input clock signal `clk` and a negative reset[1] `rst_n`. We have also defined the clock frequency in generic so that we can use it later for our counter in real time simulation. And for the outputs we have defined signals for every light respectively.

[1]Whenever the reset signal is 0 then the flip-flop is in reset and if the opposite is true (is 1) then it is out of reset.

```
architecture tfcarch of traffic_light_circuit is
type light_t is (RedR_RedP,YellowR_RedP,GreenR_RedP,
YellowR_RedP2,RedR_RedP2,RedR_GreenP);
signal light_s : light_t;
signal counter : integer range 0 to clkFrequency*60;
```

Figure 3: Module architecture

Then inside the architecture of `traffic_light_circuit` we have defined a data type `light_t` which will be used for defining the states in which the circuit enters for displaying the different lights. The signal `light_s` is defined as type `light_t` and will be used as a selector signal in the MUX[2] which is used to change the states.

The `counter` signal with the defined value range will be used for transitioning into different states after a specified amount of time.

```
begin
        process (clk) is
begin
        if(rst_n = '0') then
        light_s   <= RedR_RedP;
        counter   <= 0;
        Red_R     <='1';
        Yellow_R  <='0';
        Green_R   <='0';
        Red_P     <='1';
        Green_P   <='0';
elsif rising_edge(clk) then
        counter <= counter + 1;
        case light_s is
```

Figure 4: Model architecture: Process

After the `clk` signal changes the process initiates, first checking if its in active reset and if that condition is met it assigns the initial state `Red_RedP` to the selector signal `light_s`, as well as resetting the `counter` and assigning values to the outputs according to the state.

If the flip-flop is not in active reset, it then checks for the rising edge of the clock and if that condition is met the `counter` increments and we enter the leftmost defined or initial state.

This flip-flop is widely known as an asynchronous active low as cited by [1].

[2]Multiplexer

```
when YellowR_RedP =>
        Red_R     <='0';
        Yellow_R  <='1';
        Green_R   <='0';
        Red_P     <='1';
        Green_P   <='0';
        if rising_edge(btn) then
                counter <= 0;
                light_s <= YellowR_RedP2;
        end if;
        if(counter >= clkFrequency *3 - 1) then
                counter <= 0;
                light_s <= GreenR_RedP;
        end if;
```

Figure 5: Architecture: Sample State

In the figure 5, the `light_s` signal is set to `YellowR_RedP` and as we mentioned previously the outputs change in accordance with the current state. After that is finished, it checks for a button press by registering a rising edge from the `btn` signal. If a rising edge is registered then the counter resets and `light_s` changes state so it can allow pedestrians to cross. If not, the counter continues to increment (4) until it reaches its time threshold (which is dependent on the clock period and frequency and state that it is in, in this case for the yellow it would last 3 seconds in real-time simulation), then it resets and `light_s` changes to the next predefined state.

All the other states are defined in a similar fashion, to see the rest of the architecture please refer to the `traffic_light_circuit.vhdl` file.

3.2 VHDL: Test bench

Now we need to write a test bench where we can test our complete module in a real time simulation. We will call that test bench `traffic_light_circuit_tb.vhdl`.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity traffic_light_circuit_tb is
end entity;
architecture simulation of traffic_light_circuit_tb is
constant clkFrequency: integer := 100;
constant clkPeriod : time    := 1000 ms / clkFrequency;
signal clk      : std_logic := '1';
signal rst_n    : std_logic := '0';
signal btn      : std_logic := '0';
signal Red_R    : std_logic;
signal Yellow_R : std_logic;
signal Green_R  : std_logic;
signal Red_P    : std_logic;
signal Green_P  : std_logic;
```

Figure 6: Test bench architecture: Signals

In the test bench file we define the previously mentioned signals so that we can port them over to the corresponding input and output ports on the traffic light module. We also define the clock period and frequency constants which **are of utmost importance** for the real-time simulation as the clock and counter signal depend on them.

```
tfc_test : entity work.traffic_light_circuit(tfcarch)
generic map (clkFrequency => clkFrequency)
port map (
clk      => clk,
rst_n    => rst_n,
btn      => btn,
Red_R    => Red_R,
Yellow_R => Yellow_R,
Green_R  => Green_R,
Red_P    => Red_P,
Green_P  => Green_P );
```

Figure 7: Test bench architecture: Ports

This is where we declare an instance for our module and where we port the signals.

```
clk <= not clk after clkPeriod / 2;
process is
begin
wait until rising_edge(clk);
wait until rising_edge(clk);
rst_n <= '1';
btn <= not btn after 14000 ms;
wait until rising_edge(btn);
btn <= not btn after 2000 ms;
wait;
end process;
end architecture;
```

Figure 8: Test bench architecture: Process

The first statement is a concurrent one so the sensitivity list is on the right. After that, the moment when 2 rising edges are registered the flip-flop comes out of reset and the process begins according to [2]. The moment the `btn` has a rising edge (in this case after 14 seconds) then the pedestrian traffic light outputs begin to change as well. After 2 seconds the signal returns to 0 and simulation continues as usual.

4 Simulation

After we compile the module and the test bench we simulate the test bench to see if our traffic light circuit is working as intended. The signals in the simulation:

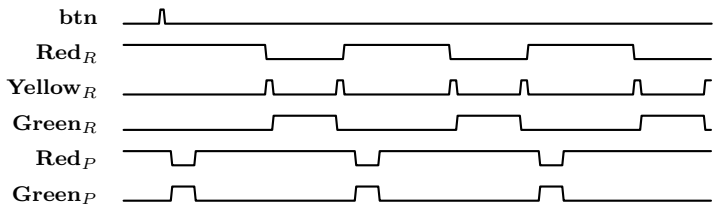

Figure 9: Signals in simulation

After running a 5 minute simulation we can see that our traffic light circuit is working as we intended it to, where the time difference between the states are correct and the light signals in each state are changed accordingly.

5 Final Words

Tackling this project has been an interesting experience as me and my team throughout the ordeal have learned a great deal about LaTeX, VHDL and ModelSim simulations. We have come to appreciate more the smaller digital circuits that also have their own part in society as well as better understanding them.
The members of our team are: Marko Gjorgjievski, Adam Yousif Algallaf and Maximilian Schaefer.

Special thanks to Professor Bonenberger for his lectures and introducing digital circuits to us.

List of Figures

References

[1] VHDL-Wizz: `https://vhdlwhiz.com/` Date of access: 17.06.20

[2] VHDL-Renerta: `http://vhdl.renerta.com/` Date of access: 21.06.20

[3] Traffic light info: `https://en.wikipedia.org/wiki/Traffic_light`

[4] Traffic light picture:“Traffic lights display green to indicate ”go ahead”.” `https://en.wikipedia.org/wiki/Green-light#/media/File:LED_traffic_light.jpg` Date of access: 15.06.20

[5] `https://www.youtube.com/playlist?list=PL-p5XmQHB_JSQvW8_mhBdcwEyxdVX0c1T` Date of access: 22.06.20

[6] `https://www.youtube.com/watch?v=h4ZXge1BE80&list=PLIbRYKjjYOPkhpxnkQ0fwTXnmgsiCMcVV` Date of access: 17.06.20