Traffic-Light Control System
GORGONZOLA
10th June 2022

1 Team members

Yigitcan Aydin
Muhammad Rohail Usman
Syed Rafsan Ishtiaque

2 Introduction

Introduction:

Our project was the implementation of the Traffic-light Control System. Conventional traffic system has some constraints such as lagging, waste of time, and so on. We tried to solve those problems. Smart traffic systems enable vehicles to move in a faster and safer way. The volume of traffic will determine the waiting period of each crossing. Also the pedestrian crossing of the road will be guided by the smart system. That way if there is no pedestrian at a certain time available for the crossing, the system will detect that and reduce the halting of vehicles in that section of the crossing/ road. Another important feature is that, if any direction of a junction has more volume of vehicles then the system will provide longer enable channels for that direction. It will reduce the waiting period. Also the traditional traffic system has a huge amount of maintenance cost, we can reduce it with our implementation, as the use of physical switches will be replaced by sensors. The switches in every pedestrian crossing are subject to lots of maintenance of the parts and thus it increases the cost. But with a smart traffic system it will not only reduce but also for physical disability of persons, the safety feature will ensure an accident free road crossing.

Importance of FPGAs and VHDL for the project:

FPGAs allow developers to implement the desired hardware components by using the VHDL code only. This feature comes in handy, when the developers do not want or need to implement big scale boards and waste resources on the board such as extra unused hardware components. Since FPGAs can be programmed, developers do not need to design the circuit schematics using CAD tools. At this point VHDL is a very useful tool for us. We write only the VHDL code and upload it onto a FPGA board, then FPGA will arrange the necessary routings or wirings between the basic logic gates and we are able to run our VHDL design on a FPGA board.

3 Concept description

With reference to the below Block Diagram, we made a scrum methodology approach towards the project. After finalizing the project, a stepwise solution was tried to be achieved starting on with VHDL Implementation till the FPGA part. The main idea behind the selection of this topic was to deepen the understanding of VHDL and PCB Designing through real-world application in an illustrative manner.

Block diagram:

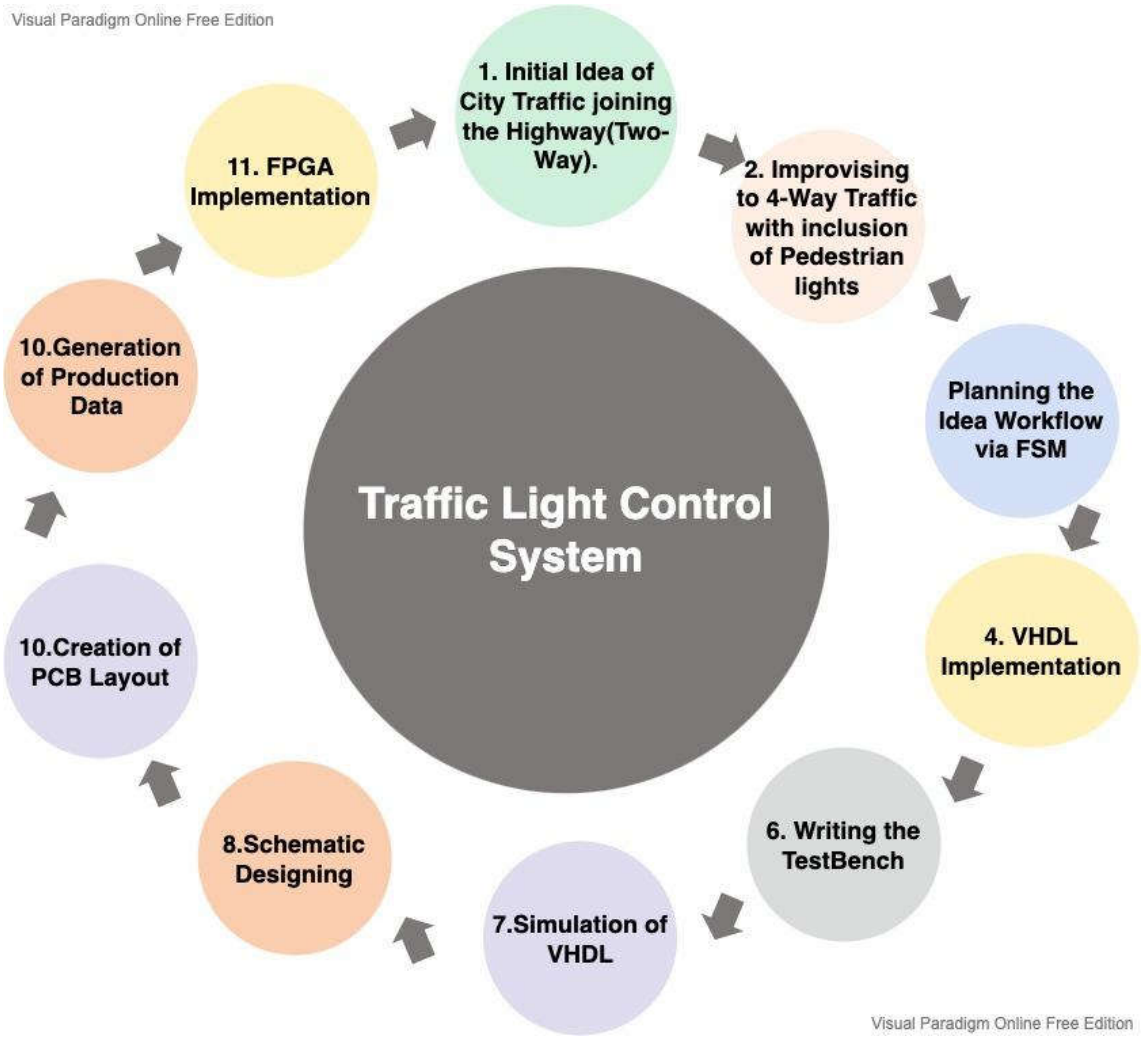

Figure 1: Block Diagram

Main application for the prototype:

As the name suggests, Traffic Light Control System, can be readily used under circumstances of high traffic flow, four-way conjunctions. Especially, near the commercial sector where the need for pedestrian lights are required to be working with precision. Moreover, not only this, this prototype can also be used within industrial areas having wider road networks.

4 Project/ Team management

Project methods used in the project:

For our hardware project we divided our tasks and worked accordingly. As the project started with the basic concept, everyone had a clear idea about what was our goal and how to achieve that. We had a lab session where we discussed the steps and individual tasks for the project. We used Github to keep our work and share among members. We categorized our project into four main parts, VHDL, Testbench, PCB and FPGA. From a modeling perspective, it was a WATERFALL model.

Managing our tasks:

We had several meetings throughout the week. The lab sessions were included in addition to extra meets for work. The virtual meetings were also a way to get in touch. For paper and slides, we used google sharing so that way everyone can put their pieces into the whole. And whenever we faced any challenge, we consulted with the Professor.

Roles of the team members in the project:

Aydin: Implementation of the VHDL code, designing the Finite State Machine (FSM)

Rohail: Implementation and Simulation of the VHDL testbench code, Block diagram

Rafsan: PCB section, FPGA section, Writing the project management section

5 Technologies

- **VHDL + VHDL Testbench**

  ModelSim - Intel FPGA Starter Edition 2021.1

- **Eagle**

  For the PCB section, we have used Autodesk Eagle version 9.6.2. free license for non-commercial use. Also we have used the altium.com website for recording the outlook of the PCB design.

- **FPGA**

  ISE 14.7 Virtual Machine for Windows 10. Release version: 14.7 (nt64), Application version: P.20131013, Copyright: Xilinx, Inc

6 VHDL

In the implementation of the code, a finite state machine is being used to demonstrate the overall system behavior. Since the lights have to be interconnected depending on their light states in order to prevent accidents, the states of the car lights 1 and 2 are being used as the reference states.

States:

Car lights 1 and 2: Green(10s) -> Yellow(3s) -> Red(3s) -> Red(10s) -> Red_b(3s)->Yellow_b(3s)

(1) (2) (3) (4) (5) (6)

Car lights 3 and 4: Red(10s) -> Red(3s) -> Yellow(3s) -> Green(10s) -> Yellow(3s) -> Red(3s)

Ped lights 1,2,5,6: Red(10s) -> Red(3s) -> Green(3s) -> Green(10s) -> Green(3s) -> Red(3s)

Ped lights 3,4,7,8: Green(10s) -> Green(3s) -> Red(3s) -> Red(10s) -> Red(3s) -> Green(3s)

Following are the state signal names used in the VHDL code implementation:

State signal name explanations:

(1) G12_3478_R34_1256: Green is 'on' for car lights 1,2 and pedestrian lights 3,4,7,8;
Red is 'on' for car lights 3,4 and pedestrian lights 1,2,5,6

(2) Y12_R34_1256_G3478: Yellow is 'on' for car lights 1,2;
Red is 'on' for car lights 3,4 and pedestrian lights 1,2,5,6;
Green is 'on' for pedestrian lights 3,4,7,8

(3) R12_3478_Y34_G1256: Red is 'on' for car lights 1,2 and pedestrian lights 3,4,7,8;
Yellow is 'on' for car lights 3,4;
Green is 'on' for pedestrian lights 1,2,5,6

(4) R12_3478_G34_1256: Red is 'on' for car lights 1,2 and pedestrian lights 3,4,7,8;
Green is 'on' for car lights 3,4 and pedestrian lights 1,2,5,6

(5) R12_3478_Y34_G1256_b: Red is 'on' for car lights 1,2 and pedestrian lights 3,4,7,8;
Yellow is 'on' for car lights 3,4;
Green is 'on' for pedestrian lights 1,2,5,6

(6) Y12_R34_1256_G3478_b: Yellow is 'on' for car lights 1,2;
Red is 'on' for car lights 3,4 and pedestrian lights 1,2,5,6;
Green is 'on' for pedestrian lights 3,4,7,8

Finite State Machine (FSM) of the complete process is as following:

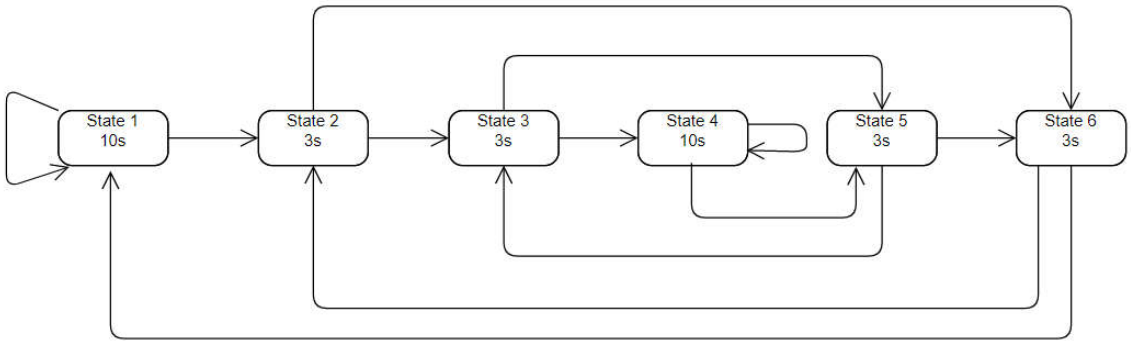

Figure 2: Finite State Machine

“State 1” is the initial state of the system. If sensor_one or sensor_two == 1, then stay in the “State 1”. If sensor_three or sensor_four == 1, then change state to “State 2”. Without sensor interrupt (normal flow of the system) change state to “State 2.

In “State 2”, if sensor_one or sensor_two == 1, then change state to “State 6”. If sensor_three or sensor_four == 1, then change state to “State 3”. In the normal flow of the system, change state to “State 3”.

In “State 3”, if sensor_one or sensor_two == 1, then change state to “State 5”. If sensor_three or sensor_four == 1, then change state to “State 4”. In the normal flow of the system, change state to “State 4”.

In “State 4”, if sensor_one or sensor_two == 1, then change state to “State 5”. If sensor_three or sensor_four == 1, then stay in “State 4”. In the normal flow of the system, change state to “State 5”.

In “State 5”, if sensor_one or sensor_two == 1, then change state to “State 6”. If sensor_three or sensor_four == 1, then change state to “State 3”. In the normal flow of the system, change state to “State 6”.

In “State 6, if sensor_one or sensor_two == 1, then change state to “State 1”. If sensor_three or sensor_four == 1, then change state to “State 2”. In the normal flow of the system, change state to “State 1”.

Entity of the VHDL code:

```
entity light_controller is
 port (  -- sensor ports (sensor_one,etc.) are ports for the push buttons
          sensor_one     : in STD_LOGIC; -- Sensors to detect the cars
```

```
        sensor_two     : in STD_LOGIC;
        sensor_three    : in STD_LOGIC;
        sensor_four     : in STD_LOGIC;
        -- clk: stands for clock for the system
    clk  : in STD_LOGIC; -- clock
        -- rst_n: resets the system (assigns current state to G12_3478_R34_1256)
    rst_n: in STD_LOGIC; -- reset active low
        --RED_YELLOW_GREEN (CAR TRAFFIC LIGHTS)
    light_one            : out STD_LOGIC_VECTOR(2 downto 0);
    light_two  : out STD_LOGIC_VECTOR(2 downto 0);
        light_three      : out STD_LOGIC_VECTOR(2 downto 0);
        light_four       : out STD_LOGIC_VECTOR(2 downto 0);

        --RED_GREEN (PEDESTRIAN TRAFFIC LIGHTS)
        ped_one                         : out STD_LOGIC_VECTOR(1 downto 0);
        ped_two                         : out STD_LOGIC_VECTOR(1 downto 0);
        ped_three               : out STD_LOGIC_VECTOR(1 downto 0);
        ped_four                : out STD_LOGIC_VECTOR(1 downto 0);
        ped_five                : out STD_LOGIC_VECTOR(1 downto 0);
        ped_six                         : out STD_LOGIC_VECTOR(1 downto 0);
        ped_seven               : out STD_LOGIC_VECTOR(1 downto 0);
        ped_eight               : out STD_LOGIC_VECTOR(1 downto 0)
  );
end light_controller;
```

7 TESTBENCH

In the testbench part, we are assigning all the possible sensor (or button) values to the related ports. To begin with the architectural explanation of the test bench. Firstly we initiated by defining the component where the entity of VHDL code was utilized. Moreover, we began the unit under test by performing the mapping of the ports. In addition, in the stimulus process, we designed the workflow of the sensors with respect to the decided states in our VHDL. Apart from that, we are creating the processes for the clock and the reset ports in order to assign them different values during the simulation. Conclusively, to have a clean visualization we selected the default compiler timing for the clock that is in picoseconds.

Few of the value assignments can be found in the following code snippet:

```
-- Clock process definitions
  clk_process :process
  begin
                clk <= '0';---clock is high for one pic seconds
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;---clock is low for one pic seconds
  end process;
```

```
 rst_process :process
  begin
                    rst_n <= '0';
                    wait for 100 ns;
                    rst_n <= '1';
                    wait ;
  end process;

  -- Stimulus process which is used to define the inputs and their combinations
  stim_proc: process
  begin
---all the sensors are zero for initial two pico seconds
                    sensor_one<='0';
                    sensor_two<='0';
                    sensor_three<='0';
                    sensor_four<='0';
                    --rst_n <= '0';
                    wait for clk_period*20;
---sensor one is high for two pico seconds
                    sensor_one<='1';
                    sensor_two<='0';
                    sensor_three<='0';
                    sensor_four<='0';
                    --rst_n <= '1';
                    wait for clk_period*20;
```

According to information that provided above, the simulation results can be seen in the following images:

Simulation result 1:

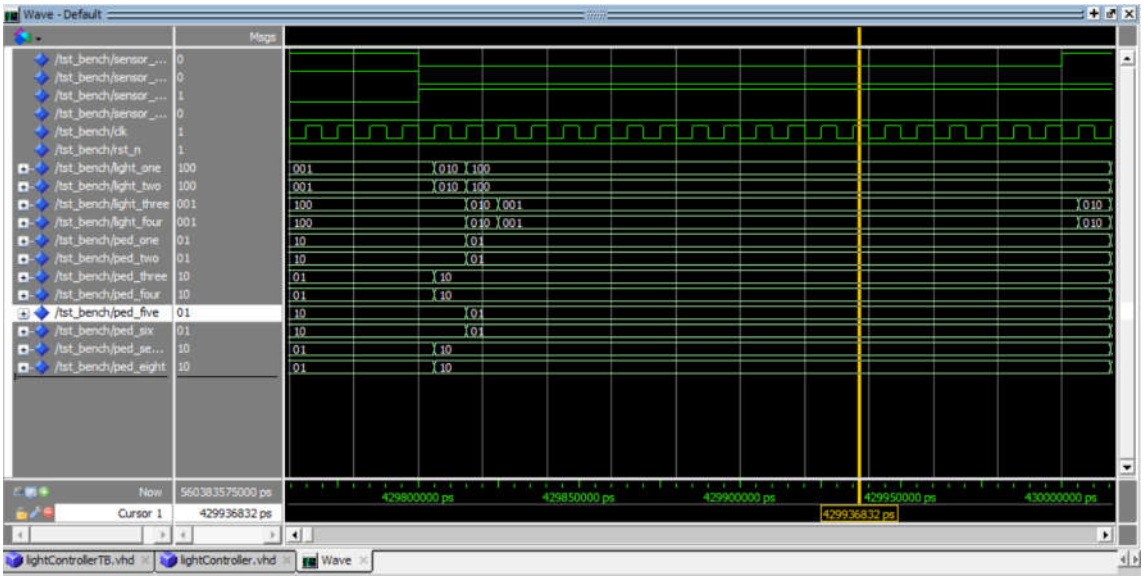

Figure 3: Simulation result 1

Simulation result 2:

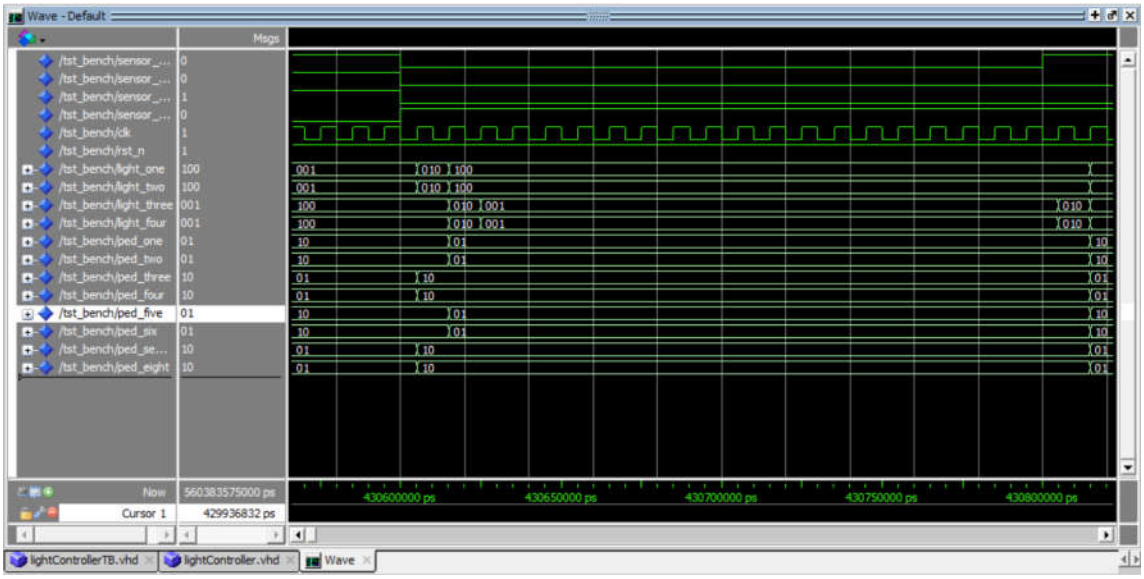

Figure 4: Simulation result 2

8 PCB Design

Schematic:

We have used the Intel / Altera EP4CE22E22C8N, Device: FBGA256_THIN_WIRE-BOND-A:1.55_, Parts/ Description: U1 to design our PCB.
Besides that, there are:
12 **GREEN** LED-GREEN0603, Package: LED-0603, Parts/ Description: D1 to D12/ Green SMD LED
12 **RED** LED-RED0603, Package: LED-0603, Parts description: D13 to D24/ Red SMD LED
4 **YELLOW** LED-YELLOW0603 (SMD), Package: LED-0603, Parts/ Description: D25 to D28/ Yellow SMD LED
1 **OSCILLATOR**, Device: OSCILLATORSMD-5X3, Package: OSCILLATOR-SMD-5X3.2, Part: Y1
4 **MOMENTARY-SWITCH**, Device: MOMENTARY-SWITCH-SPST-SMD-4.5MM, Package: TACTILE_SWITCH_SMD_4.5MM, Parts/ Description: S1, S2, S3, S4/ Momentary Switch - SPST
32 **220 Ohm Resistor**, Device: 22OHM-0805-1/4W-1%, Package: 0805, Parts/ Description: S1, S2, S3, S4/ 220Ω resistor

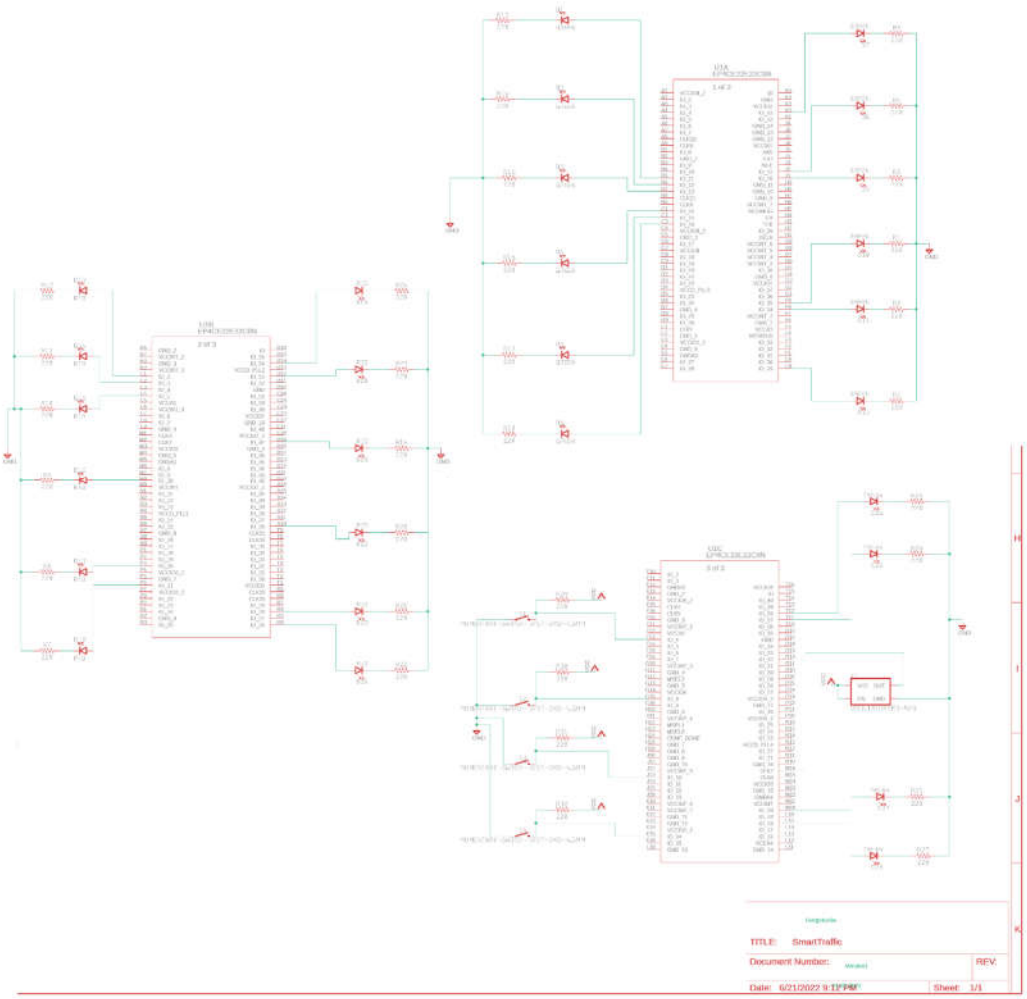

Figure 5: Schematic diagram

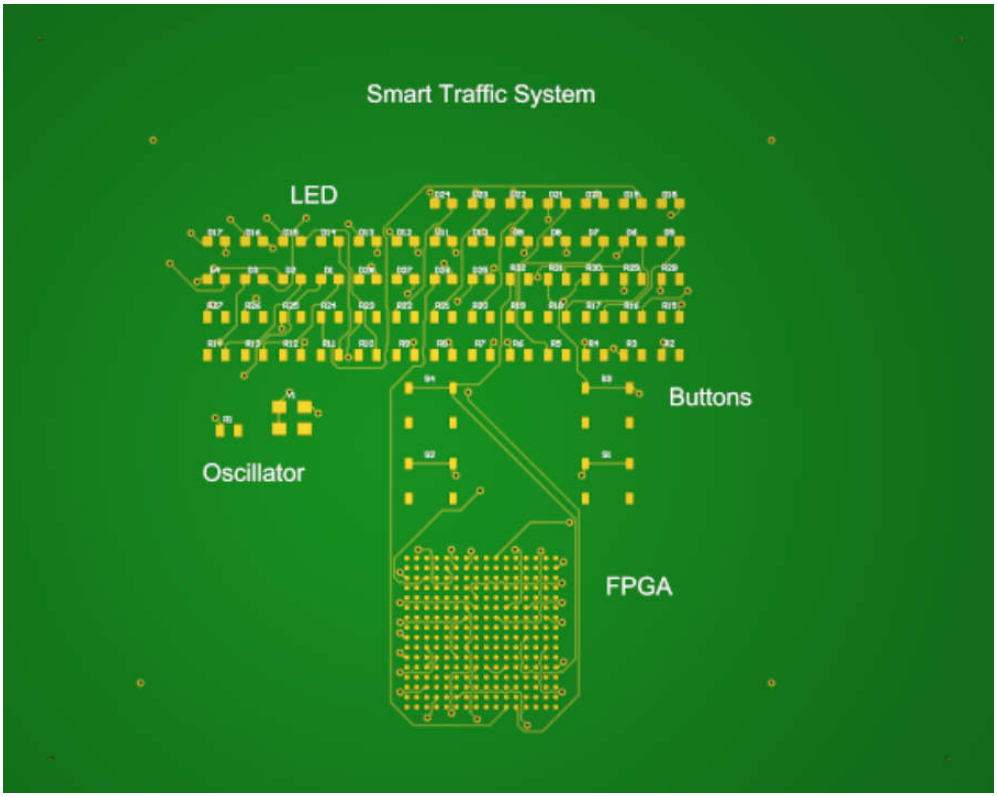

Figure 6: PCB Front view

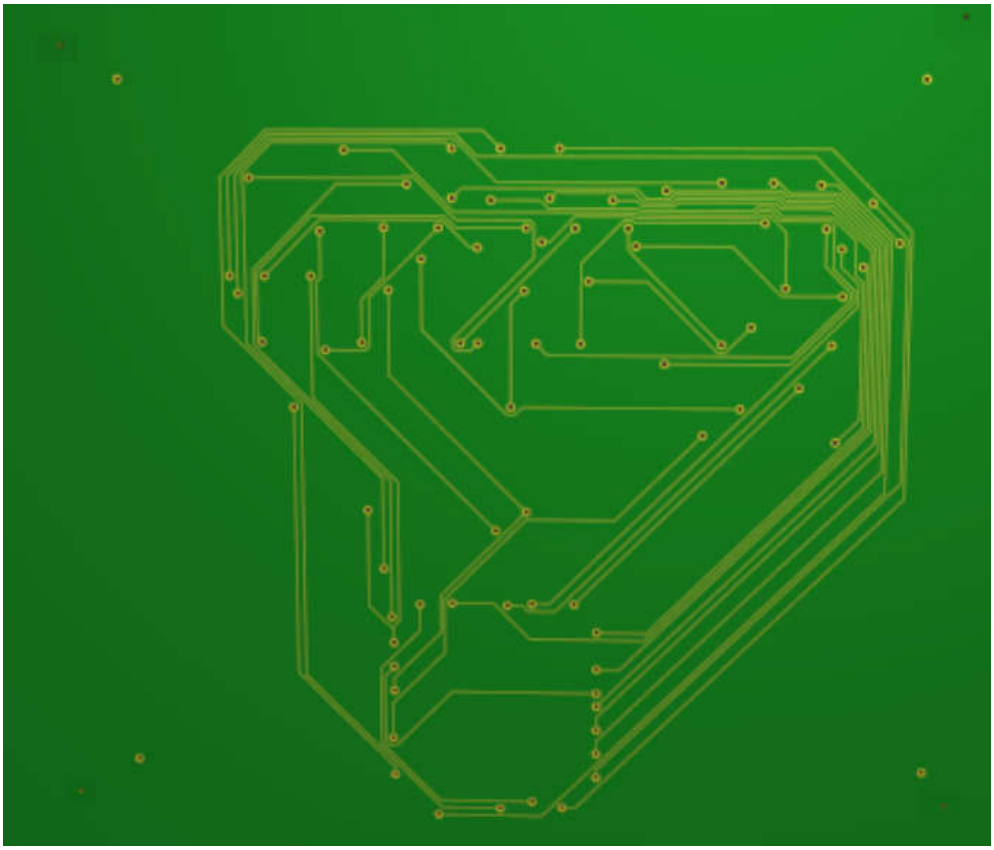

Figure 7: PCB Rear view

BOM:

Qty	Value	Device	Package	Parts/ Description
32	220	220OHM-0805-1/4W-1%	0805	R1 to R32 220Ω resistor
1	EP4CE22E22C8N	EP4CE22E22C8N	FBGA256_THIN_WIRE-BOND-A:1.55_	U1
12	GREEN	LED-GREEN0603	LED-0603	D1 to D12/ Green SMD LED
4	MOMENTARY-SWITCH-SPST-SMD-4.5MM	MOMENTARY-SWITCH-SPST-SMD-4.5MM	TACTILE_SWITCH_SMD_4.5MM Momentary Switch	S1, S2, S3, S4/ SPST
1	OSCILLATORSMD-5X3	OSCILLATORSMD-5X3	OSCILLATOR-SMD-5X3.2	Y1/ Oscillators (Generic)
12	RED	LED-RED0603	LED-0603	D13 to D24/ Red SMD LED
4	Yellow	LED-YELLOW0603	LED-0603	D25, D26, D27, D28/ Yellow SMD LED

Figure 8: BOM

Cost:

FPGA: 75 €
Resistor: 32 × 0.25 € = 8 €
LED: 32 × 0.25 € = 8 €
Oscillator: 3 €
Switch: 4 × 1 € = 4 €

Total: 98 €

Size: 100 × 80 millimeter

Remark:

In our conceptual model, we said that the sensors will be used to detect vehicles/ pedestrians. But the complexity and cost of introducing sensors along with the limitation of time, force us to use Switches. In the **future development** of the project, we will use the actual sensors.

All the files related to PCB are uploaded in the Github repository. Such as .BRD, .SCH, .EPF and Gerber files.

Link: Advanced_Embedded_Lab-Hardware_Engineering_Lab/Hardware Engineering Lab/Project/PCB/

9 FPGA

For our FPGA implementation, we have used the ISE 14.7 Virtual Machine for Windows 10 software (Spartan3/ xc3s1000/ft256). The VHDL code developed during the project work was used. For reference, we have used the lecture slides by the professor. Below are some outcomes of our work.

1. RTL Schematic:

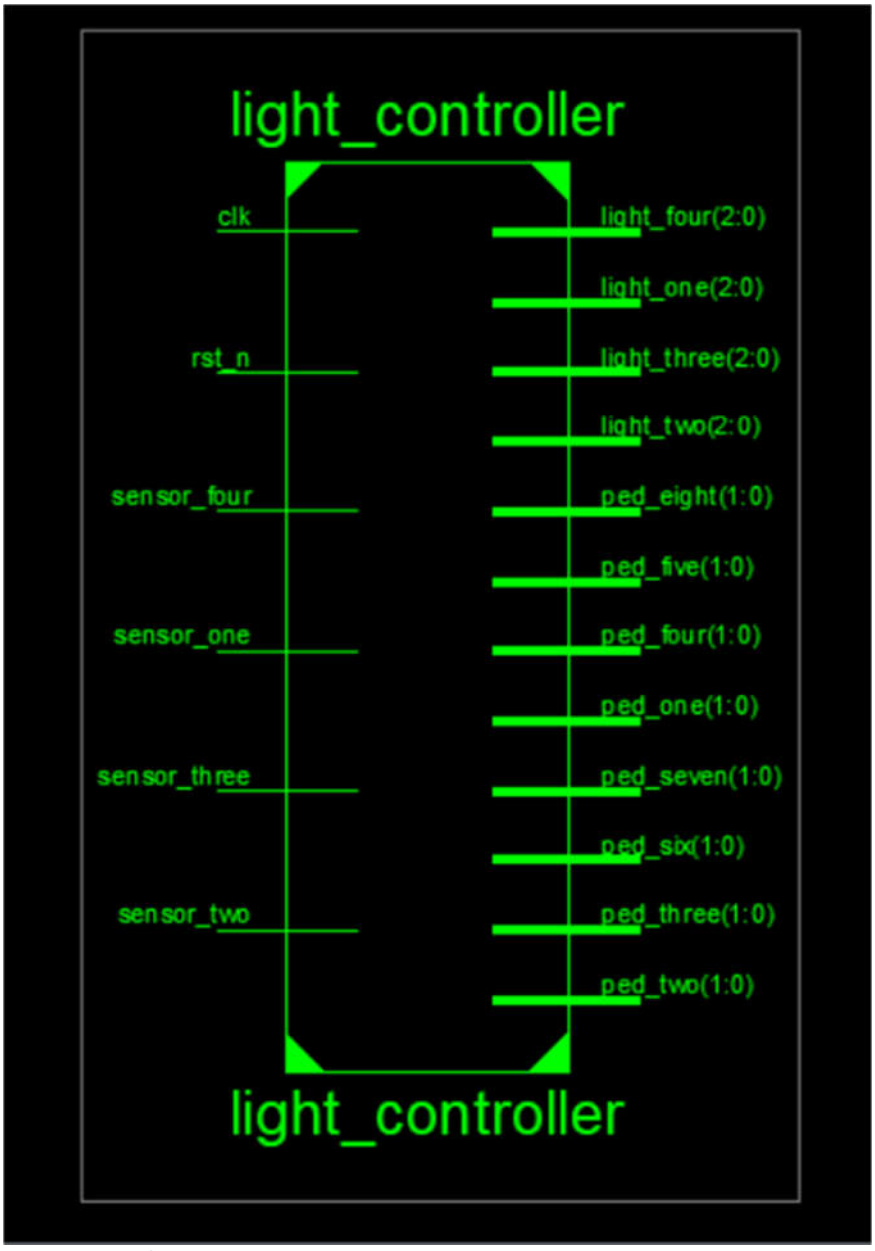

Figure 9: RTL Schematic

2. Technology Schematic:

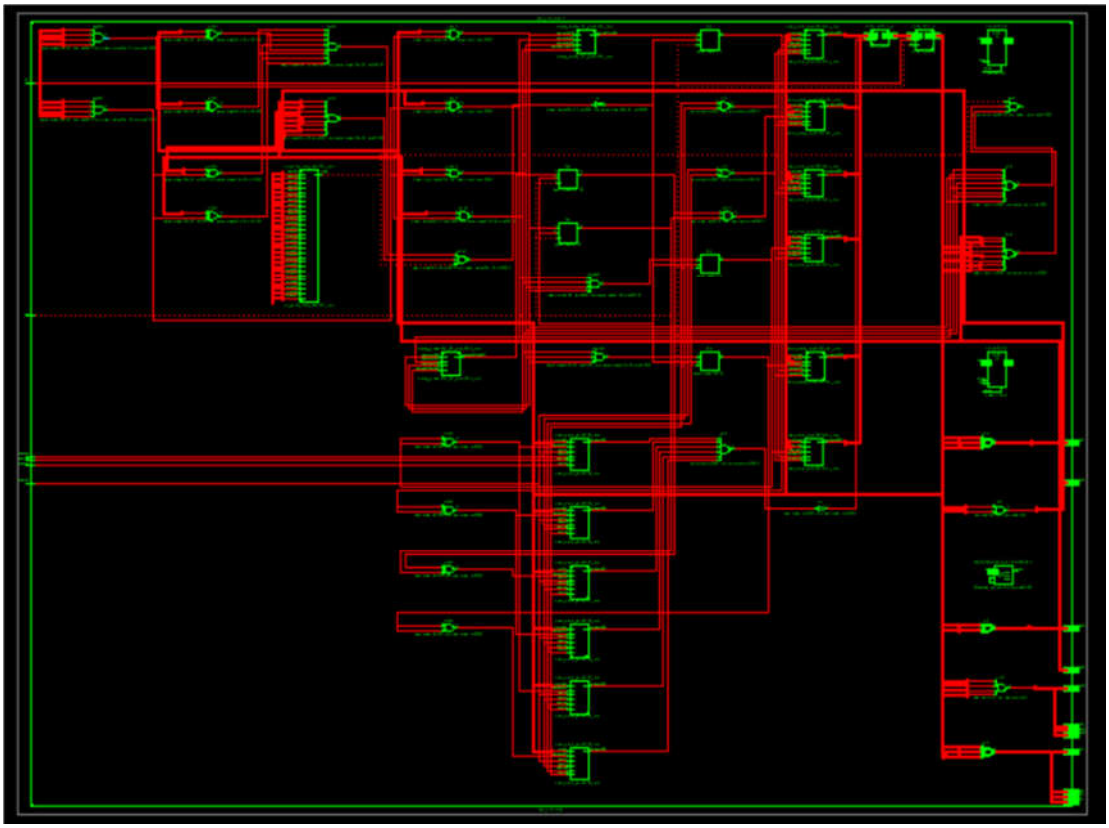

Figure 10: Technology Schematic

3. Technology Schematic FPGA component group 1:

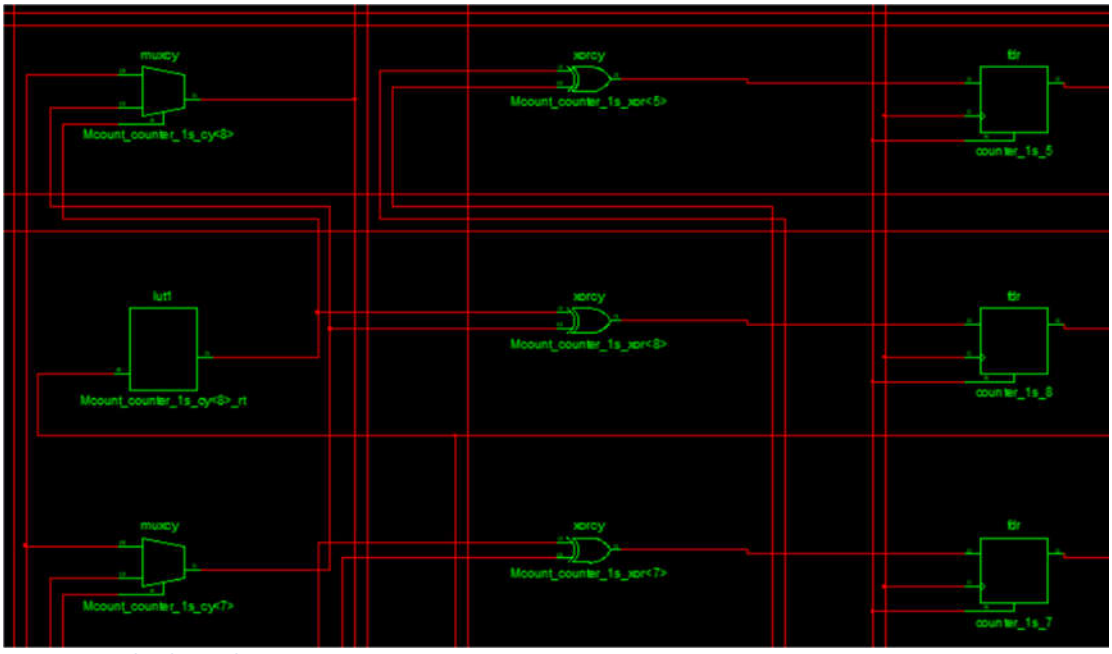

Figure 11: Technology Schematic FPGA component group 1

4. Technology Schematic FPGA component group 2:

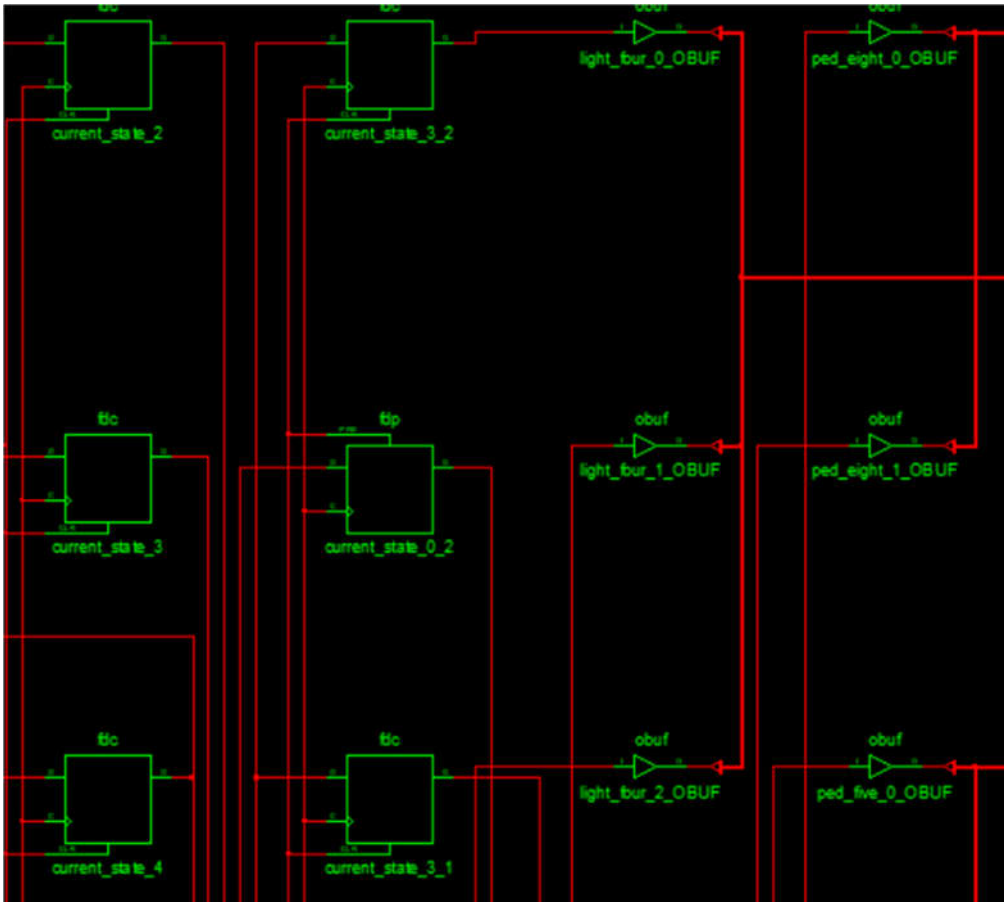

Figure 12: Technology Schematic FPGA component group 2

5. Routed Design for switches:

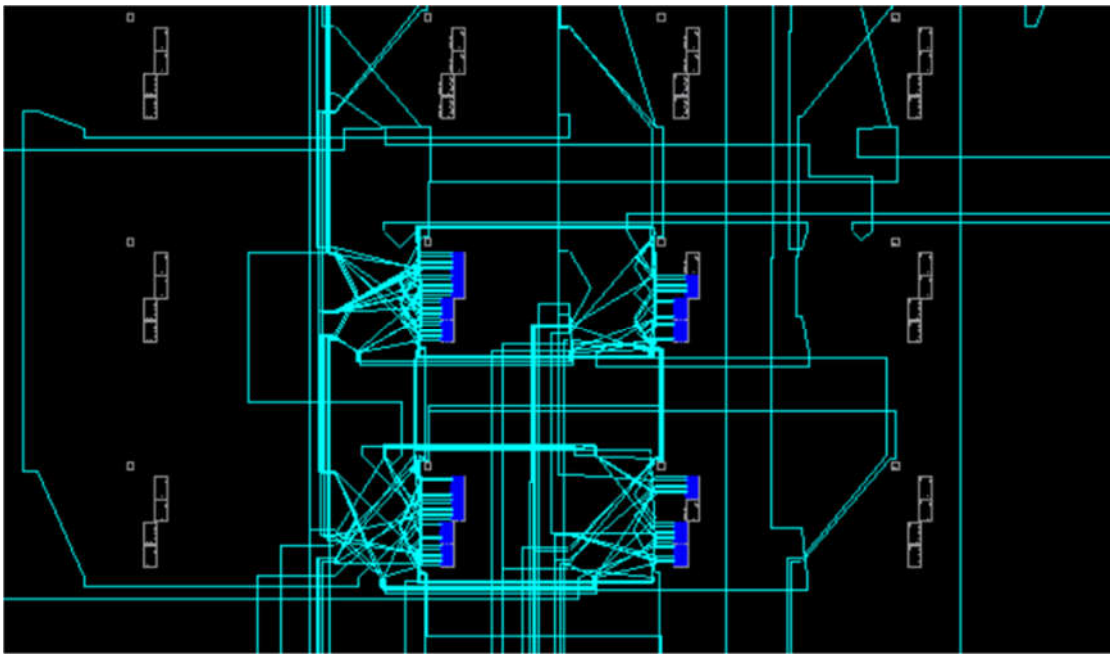

Figure 13: Routed Design for switches

6. Routed Design with partial components:

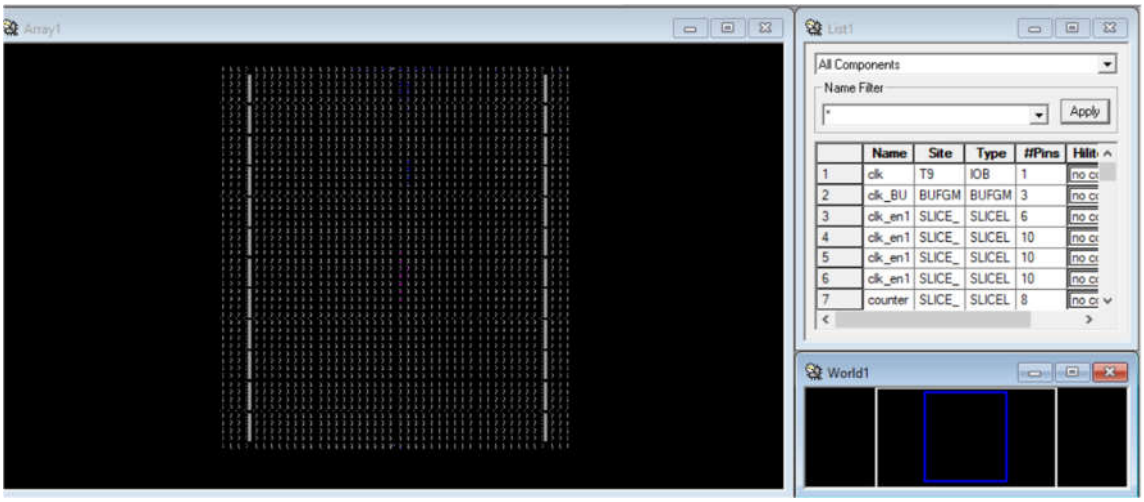

Figure 14: Routed Design with partial components

7. Design summary Partial:

TST_BENCH Project Status (06/23/2022 - 18:36:07)			
Project File:	traffic_light_controller.xise	**Parser Errors:**	No Errors
Module Name:	light_controller	**Implementation State:**	Programming File Generated
Target Device:	xc3s1000-5ft256	**• Errors:**	No Errors
Product Version:	ISE 14.7	**• Warnings:**	13 Warnings (11 new)
Design Goal:	Balanced	**• Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	All Constraints Met
Environment:	System Settings	**• Final Timing Score:**	0 (Timing Report)

Device Utilization Summary [-]				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Total Number Slice Registers	49	15,360	1%	
Number used as Flip Flops	43			
Number used as Latches	6			
Number of 4 input LUTs	61	15,360	1%	
Number of occupied Slices	48	7,680	1%	
Number of Slices containing only related logic	48	48	100%	
Number of Slices containing unrelated logic	0	48	0%	
Total Number of 4 input LUTs	88	15,360	1%	
Number used as logic	61			
Number used as a route-thru	27			
Number of bonded IOBs	34	173	19%	
IOB Flip Flops	4			
Number of BUFGMUXs	1	8	12%	

Figure 15: Design summary Partial

To generate the final bitstream file, we need the complete UCF file. Since we don't have all pins mapped in UCF file, it's not possible to generate the bitstream file at this moment. It is now included in our future development phase.

Still we have our UCF file uploaded in the Github repository. It contains te mapping of almost 90% pins.

10 Sources/References

1. B J Lameres ,,"Introduction to Logic Circuits Logic Design with VHDL,“ Springer Publisher, 2017 1 st Edition

2. V A Pedroni ,,"Circuit Design with VHDL," 2004 MIT

3. P J Ashenden ,,"The VHDL Cookbook," July 1990