NAME   Kshitiz Thapa & Catherine Sarmiento             COURSE ENGR 2730 Digital Logic

REPORT DATE November 15, 2018                             DUE DATE December 5, 2018

INSTRUCTOR: Shahram Rohani

**Term Project Fall 2018**

**Finite State Machines in VHDL**

EVALUATION TOPIC                                       COMMENTS

1.    Cover page                               \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.    Table of Contents                       \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3.    Introduction                            \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4.    Objectives                                   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5.    Equipment                              \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6.    Procedure                                   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7.    Discussion of Results                         \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8.    Analysis and Conclusions                          \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

9    Supporting Documents                          \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

                                    Composite Score     \_\_\_\_\_\_\_\_\_\_\_\_

EVALUATOR COMMENTS

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



https://lh4.googleusercontent.com/xtqCGuarXw0cSqZRhqsg1Pv9iXYogDNe66jdGbNrRVul-W14kxFCuxSLjB71S9QqYWrk07GKmaJ222zSLiDtXPaZpDLKlsL5PLbMwOt8CnTek6fRw2a8pxzWichsG73MitT8Lt7s

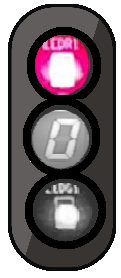
**Department of**

**Engineering Technology**

**ENGR 2730:  Digital Logic Lab**

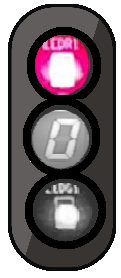
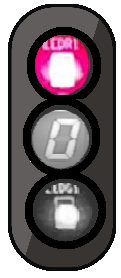
**Term Project Fall 2018**

**Finite State Machines in VHDL**

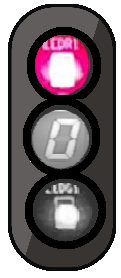
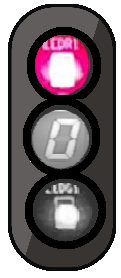
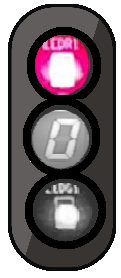
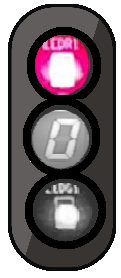


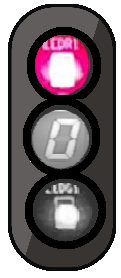














**TABLE OF CONTENTS**

Introduction…………………….…………………….………………………….……..………….. 4

Objectives…………………….……………………..…………………………..……..…………... 5

Equipment……………………….…………………………..…………………………..……..…… 6

Procedure……………………….…………………………..…………………………..……..……. 7

Discussion of Results……………………….…………………………..………………………….. 9

Analysis and Conclusions………………….…………………………..…………………………. 13

Supporting Documents…………………….…………………………..…………………………. 14

**INTRODUCTION**

The “Finite State Machines in VHDL” is a collaborative work to showcase the VHDL programming learned during the semester and how it applies to the abstract concept of Finite State Machines. The VHDL is programmed through Intel’s Quartus Pro Software with the use of Cyclone IV EP4CE115F29C7 Field Programmable Gate Array Integrated Circuit on a DE2-115 Altera Terasic board. With the use of VHDL, the goal is to program a simple traffic light circuit.

Traffic lights are signaling devices that control the flow of traffic and are very important to both pedestrians and vehicle drivers. Especially in big cities where there is a lot of congestion, traffic lights play an important role in saving time and ensuring road safety. Traffic light designs have evolved throughout the years, but recent designs feature the use of Red, Yellow and Green LED lights. Most traffic lights are programmed with a fixed cycle however modern traffic lights at metropolitan areas allow pedestrians to override and affect its operation.

Great emphasis is made to assure that our Traffic light design is in conjunction with the concepts of finite state machines. Finite state machines are defined as a system that follows a sequence of pre-defined states.

Hours were spent in the VHDL programming process to ensure the logical operation of the traffic light simulation on the Altera board. The state diagrams and figures provided were of great aid and helped the programmers in the design process.

This project emphasizes the use of road vehicle sensors in its design with the goal of improving traffic light systems operation which save vehicle operators waiting time. The project also aims to program the states with the correct logic to avoid collisions and to ensure the safety of drivers that depend heavily on the function of the road traffic light. The next sections will discuss the functionality and code structure of the traffic light simulation.

**OBJECTIVES**

The main objective of this project is to design a Traffic Light controller using the State Machines. We will be creating three VHDL files under this project with file names FSM, Traffic and Divider. The FSM is the VHDL file where we write the codes for different states of the traffic light progression and what the output is going to be in binary.

The Divider module is used to slow down the external clock input CLOCK\_50 from 50 MHz to 1 Hz. The Traffic module  works as a tying module which connects the FSM and the Divider to the Altera board for code implementation.

The objective of the first part  of the project is to implement a design of a traffic light at an intersection of Main Street (1) with Second Street (2). The traffic light controller should go through transitions to get 3 green clocks, 2 yellow clocks and 3 red clocks. This means that the project will transition through 8 total clocks before repeating.

For the objective of the second part of the project, sensors are added to the traffic control system. Whenever there is a car waiting in one of the streets and there is no car in the other street, the lights for the waiting car should turn green. This design is structured so that sole cars will not need to wait when the road is empty of incoming vehicles thus saving time.

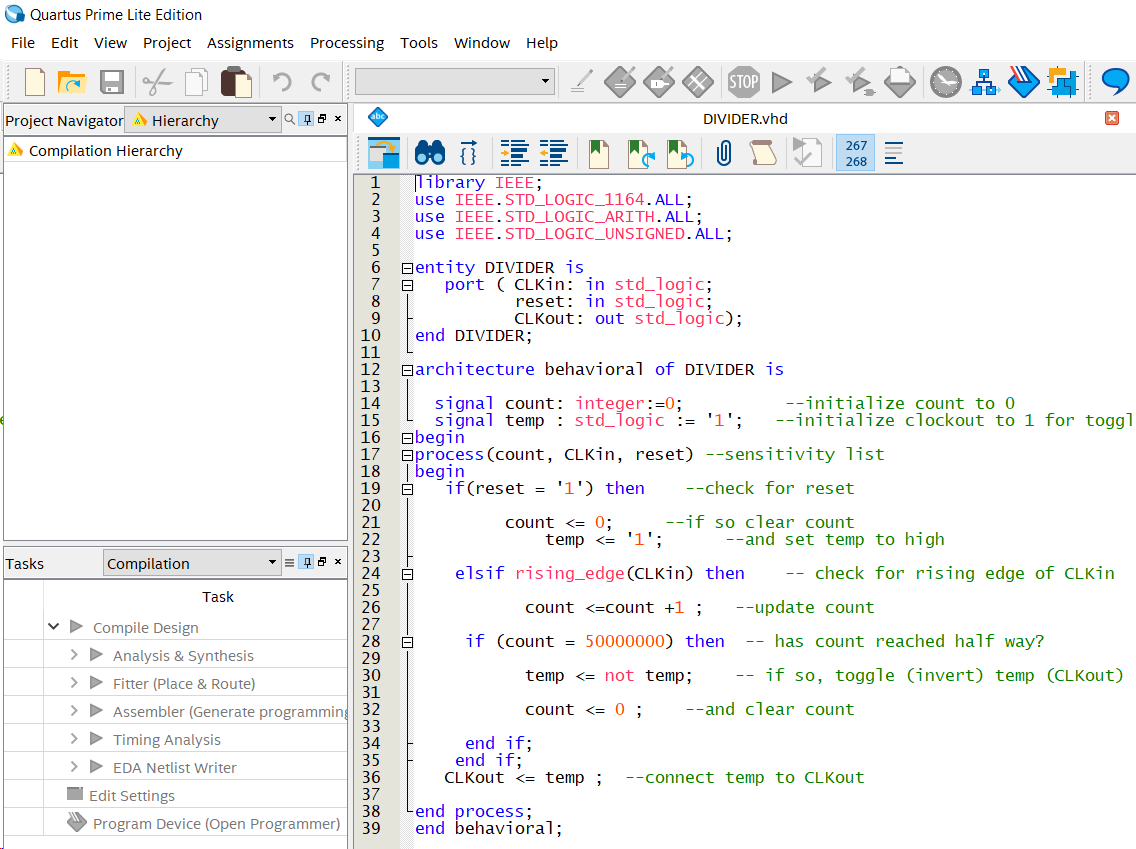
For the objective of the third part of the project, special features will be added to the project i.e. Fire Truck Override and Night Time Override. As a general overview, the Fire Truck Override is a feature that detects the sensor which turns the lights to red to signal caution whenever there is an incoming fire truck .

A second feature incorporated in this design is a Night Time Override which is a state transition that keeps light blinking red to signal caution whenever there is some maintenance going on ahead or simply when lights do not run the regular cycle at night. The next section will discuss the equipment involved in the design of this project.

The biggest objective of the programmers is to make sure the logic is incorporated well into the code and that states progress to the next state correctly.

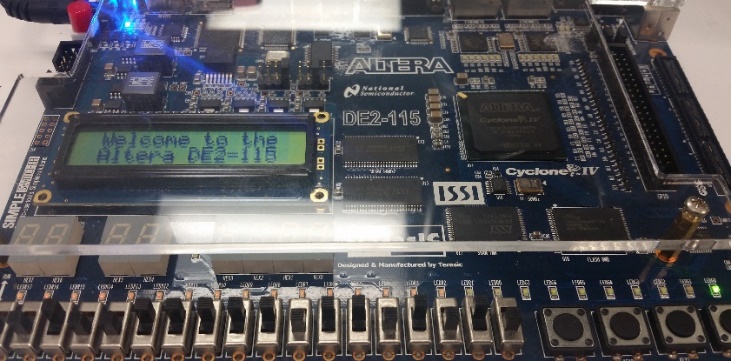
**EQUIPMENT**

Discussed below are the equipment used in this project along with their descriptions:

***Quartus Prime Lite software***

    Intel® Quartus® Prime Design Software is a design software that includes everything needed to design for Intel FPGAs, SoCs, and CPLDs. This includes design entry, synthesis to optimization, verification, and simulation. Quartus dramatically increases capabilities on devices with multi-million logic elements, providing designers with an ideal platform for next-generation design opportunities. There are numerous versions of this software, but in this project, Intel Quartus Prime Lite was used. This software may also be utilized to create VHDL files and to translate codes for logic circuits. The software compiles the code and uploads the code to the Altera board.

Figure : Intel Quartus Software

***Altera DE2-115 Cyclone IV E hardware***

Altera DE2-115 Cyclone IV E is the main hardware used in this project. The Altera logic board receives the written code from Quartus and in this simulation acts like a mini traffic light controller. The Cyclone IV E device equipped on the DE2-115 features 114,480 logic elements (LEs), the largest offered in the Cyclone IV E series, up to 3.9-Mbits of RAM, and 266 multipliers.

Figure : Altera DE2-115 Cyclone IV E Hardware

**PROCEDURE**

    For Part I of the project, we first need to create a new Quartus II project in the software and name it as you want. Then , we need to create three new VHDL files within the project and name them Divider, FSM and Traffic respectively.

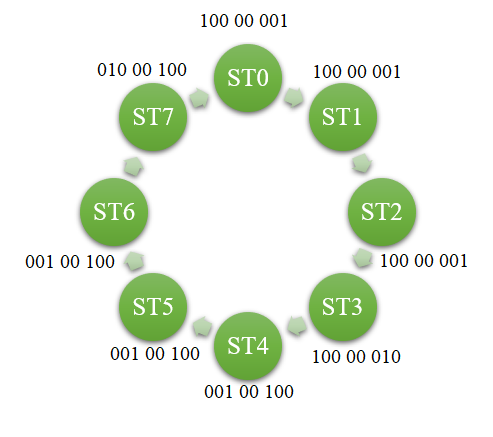
    In the FSM module, we will declare a clock input variable and 8 different variables for 8 lighting condition( 3 green, 2 yellow and 3 red) as a signal type. Then we can simply implement the traffic light as a form of 6 flip-flops using the case statement. The case statement will update the present state to the next state to result in change of lights. A With-Select statement is also used which defines and differentiates the different traffic light signal variable from each other. In the Divider module,  we use some if-else statement to divide the clock input from 50 MHz to 1 Hz. In the Traffic module, two port map statement are written each for FSM and Divider module to tie together the written code with the logic board for real-time implementation.

Figure 3: Finite State Diagram

For Part II of the project, changes will be made to the codes from Part I. Two switches SW(1) and SW(2) are used in the board as sensor which will detect if a car is waiting in one street. The Divider module is similar to preceding programs that have been worked in the class. In the FSM module,  few if-else statements are used to implement the conditions when both are street have waiting cars(sensor is “11”), both don’t have any cars(sensor is “00”) or a car is waiting in one street(sensor is “01” or “10”).

    In two conditions where both streets have cars, and none have waiting cars, (both switches high (11) or low (00) together ), the system works and goes through the cycle shown in Figure 3 above. Whenever there is only one car waiting in one of the streets ( means if only one of the switches is high (10 or 01)), the light on the car side is changed to green by updating the present state to next state using the case statement. The diagram in Figure 3 can be used to implement the different sensor condition. For the Traffic module, two port map statements are written to tie together the variables in the VHDL file with the ports in the board with the help of Figure 2 given below.

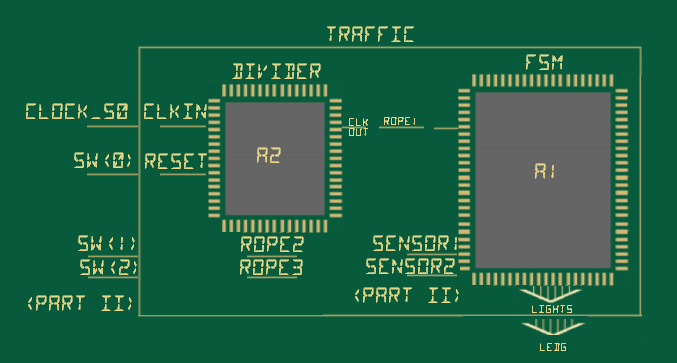
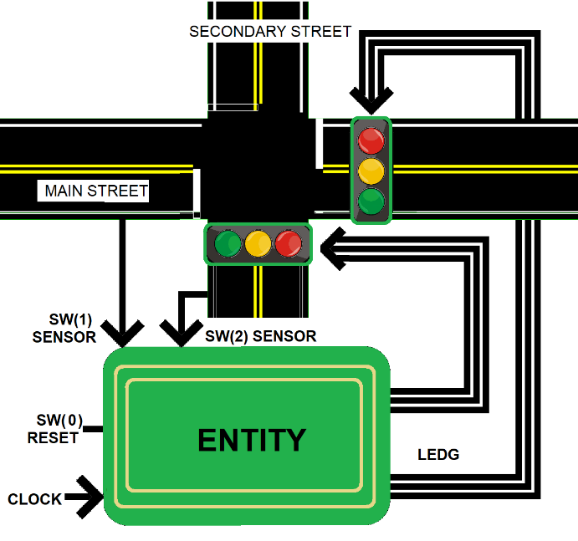


Figure 4: Traffic Schematic Diagram



|  |  |  |  |
| --- | --- | --- | --- |
| A | Sensor1 | Sensor2 | Go where? |
| If | 0 | 0 | Move on to ST1 |
| If | 1 | 1 | Move on to ST1 |
| Elsif | 1 | 0 | Stay at green1: ST0 |
| else |  |  | Move on to ST3 |

|  |  |  |  |
| --- | --- | --- | --- |
| B | Sensor1 | Sensor2 | Go where? |
| If | 0 | 0 | Move on to ST5 |
| If | 1 | 1 | Move on to ST5 |
| Elsif | 0 | 1 | Stay at green1: ST4 |
| else |  |  | Move on to ST7 |

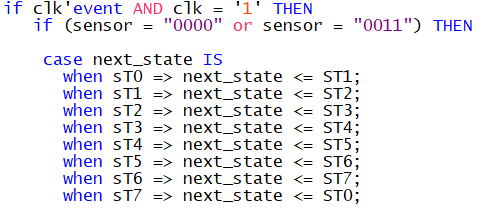
For Part III of the program, special features such as a fire truck override was programmed and activated whenever Switch 4 is on. The firetruck override turns the stoplights on Main Street and Second Street to red to signal caution. Intuitively the cars move to the right then stop.

Figure 5: Entity Declarations and Case Statements

Another feature of the Traffic Light project is a  night time override or a maintenance status where the sensors take a break and just flash or blink Red until the Traffic Light is turned on the next morning. This feature is activated when Switch 5 is turned on.

**DISCUSSION OF RESULTS**

The states and signals were declared and once the initial code was written, the program was burned into the Altera Board. On the following pages are the codes and diagrams of the Finite State Machine progression results of the program:



The initial state diagram features 7 states in which there are 3 counts of green lights, 2 yellows and 3 red state progressions. States 8 to 14 were added as it was needed in the special functionality of our traffic light simulation.



Two versions of the Simulation were programmed, one of which was a complex code which routes the output binary to the LEDR, LEDG and HEX output. LEDR representing red, LEDG representing green and lastly the HEX display representing the yellow light. Below is the result of the **complex** program that was written:

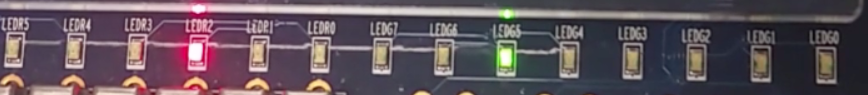


Figure : LED Output when an input of Switch 0001 or State 0, State 1, State 2 is activated



Figure : LED Output of Complex Code at State 3



Figure : Led Output when an input of Switch 0010 or State 4, State 5, State 6 is activated

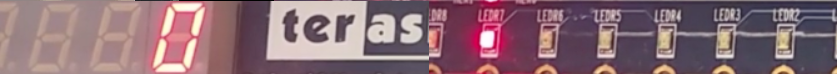
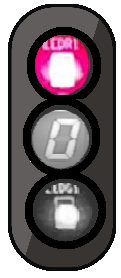
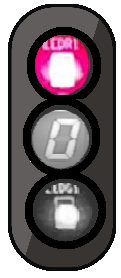
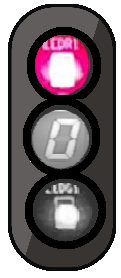
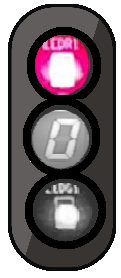
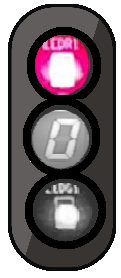
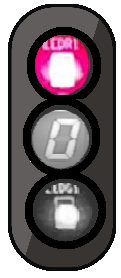
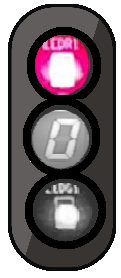
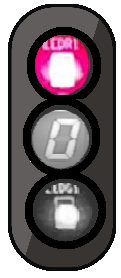


Figure : LED Output of Complex Code at State 7

On the next page is the result from the finite state machine progression of the **simple** code:



For the special feature, when switch 3 was on, the firetruck override was activated. Two case statements were coded that enabled a Firetruck Emergency function when a firetruck is approaching and a Night time override or maintenance mode switch that will put the lights on a Flashing red status. Switch 0 controls the reset function of the traffic light cycle. The two user special switches 3, 4 and its functionality is shown in the code below:

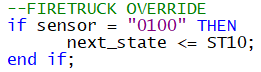


Figure : Led Output of Firetruck Override when an input of Switch 10XXX is activated

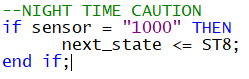
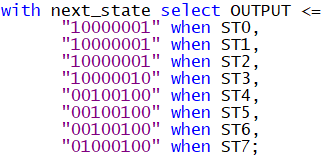
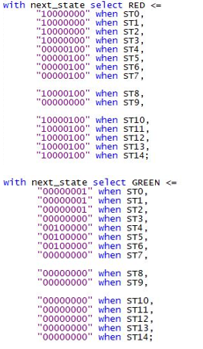
 

Figure : LED Output of Night Time / maintenance mode when an input of Switch 1XXX is activated

The main difference between our simple and complex program was how the outputs were routed and how program was coded. Below is a snippet code of the simple version:



In the complex file, there were multiple outputs that led to either LEDG, LEDR, or HEX outputs. The design was intended to showcase innovation and creativity through representing the most accurate colors. On the next page is an overview of the complex program and how VHDL is effective at organizing codes and how readable it can be:

There are three modules in this project. The first module, FSM, contains the functionality of the traffic lights. Entities, architecture and signals are declared. The behavior of states, which is a cycle from 0 to 7, is declared in this section. The states will cycle repetitively until inputs are changed. The FSM module contains Case statements that signal how the lights should react in different input sensor scenarios. With select statements program how the outputs get routed to the LEDG, LEDR, and HEX displays when it is at a certain state.

The FSM and DIVIDER files are tied together in the third file, TRAFFIC. The ropes are tied, and the inputs and outputs are declared in this file. Each sub file played a great role in making sure the whole program runs. Programming in modules instead of doing one single file creates less errors and contributes to the easy readability of the program.

**ANALYSIS AND CONCLUSIONS**

Traffic Lights are important in ensuring safety on the road and are a great example of demonstrating how a finite state machine works. These machines go through a simple cycle with defined progressions depending on which state it is currently in.

Before starting the code, a great way to save time in programming projects is to strategize, do some planning, analyze diagrams and simplify the logic.  Due to the limitations of the fixed LED arrangement, there was great difficulty in trying to accurately simulate a traffic light with the exact color and placement. There were plenty of ways to account for the circuit design differences between a Traffic Light Circuit versus an Altera Board and substitutions were done to represent the sensors and light colors. When all other groups stuck to the standard instruction of routing outputs to LEDG displays, our group wanted to incorporate some creativity in our design by routing the outputs to similarly colored displays such as LEDG, LEDR and HEX. This plan was successfully achieved and shown in the demonstration of our code.

The project “Finite State Machines on VHDL” focuses on the abstract concept of Finite State Automata and how they are incorporated in the design of Quartus Pro. Patterns occur frequently in nature and with the application of Finite State Machines, these repetitions can be simplified. The built in Finite State Machine feature in VHDL makes programming easier and faster.

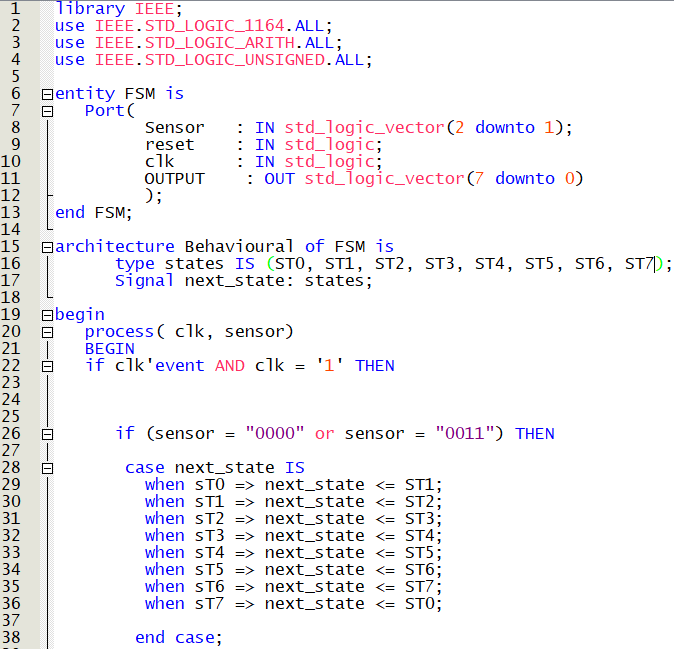
Learning various examples throughout the semester have given students a preview of the countless logic designs that can be expressed through VHDL from adders, multiplexers, binary coded decimal counters etc. Traffic light variations like timed crosswalks, sensors, exclusive lefts, exclusive rights can be programmed easily with VHDL. VHDL is also very intuitive and similar to C++ so learning the language is easier to achieve.

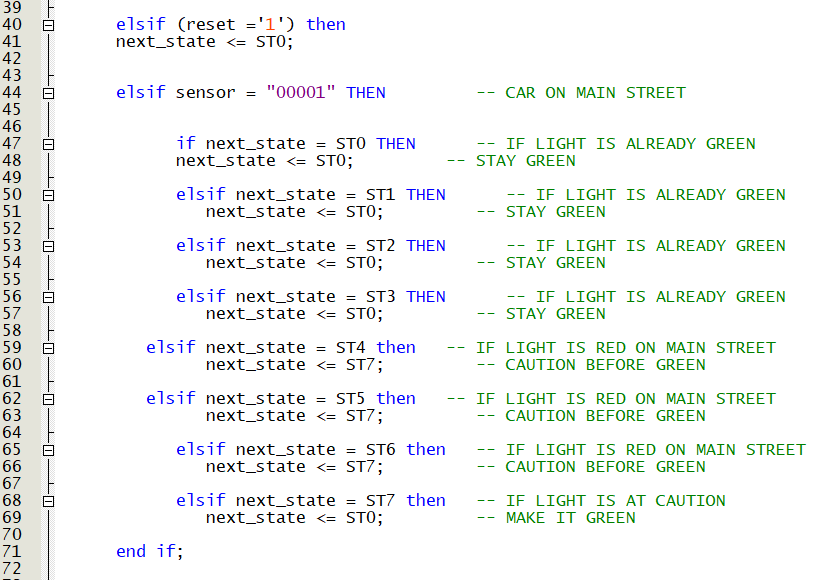
Simplicity and easy usability are at the heart of Intel’s Quartus Pro. With VHDL programming, there is so much more that can be programmed, and programmers can relax, type less and save some time.

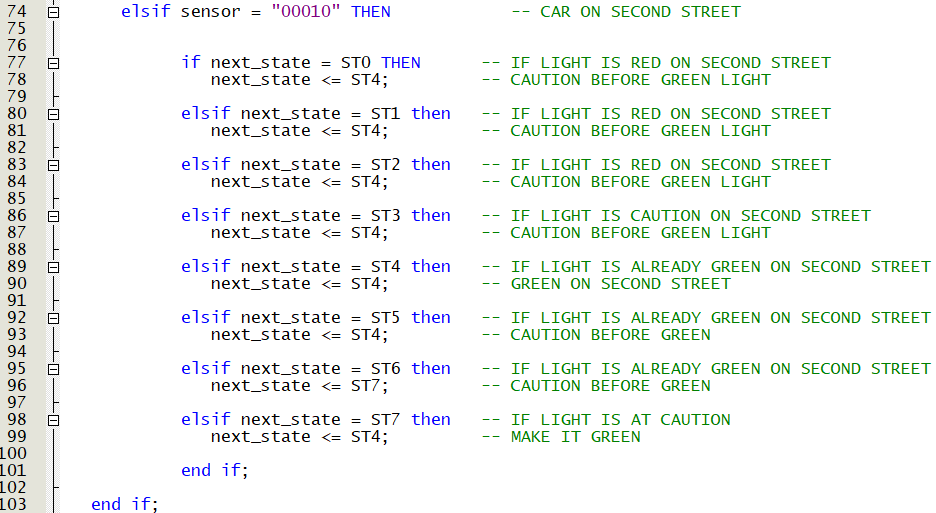
**SUPPORTING DOCUMENTS**

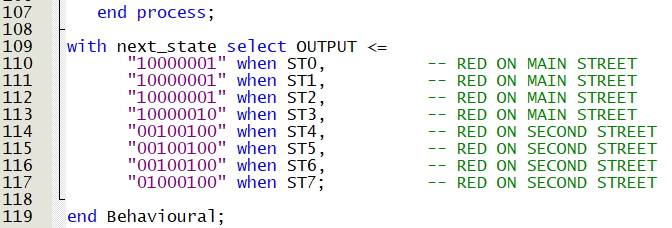
SIMPLE CODE VERSION I (All LEDG outputs used)

**FSM.VHD**

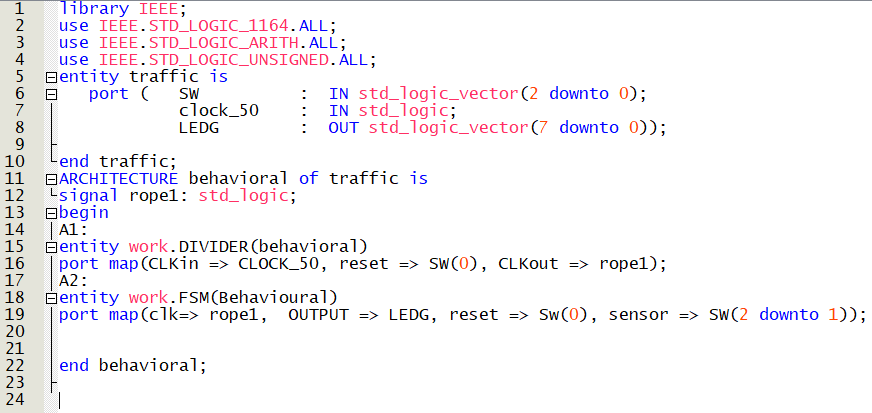






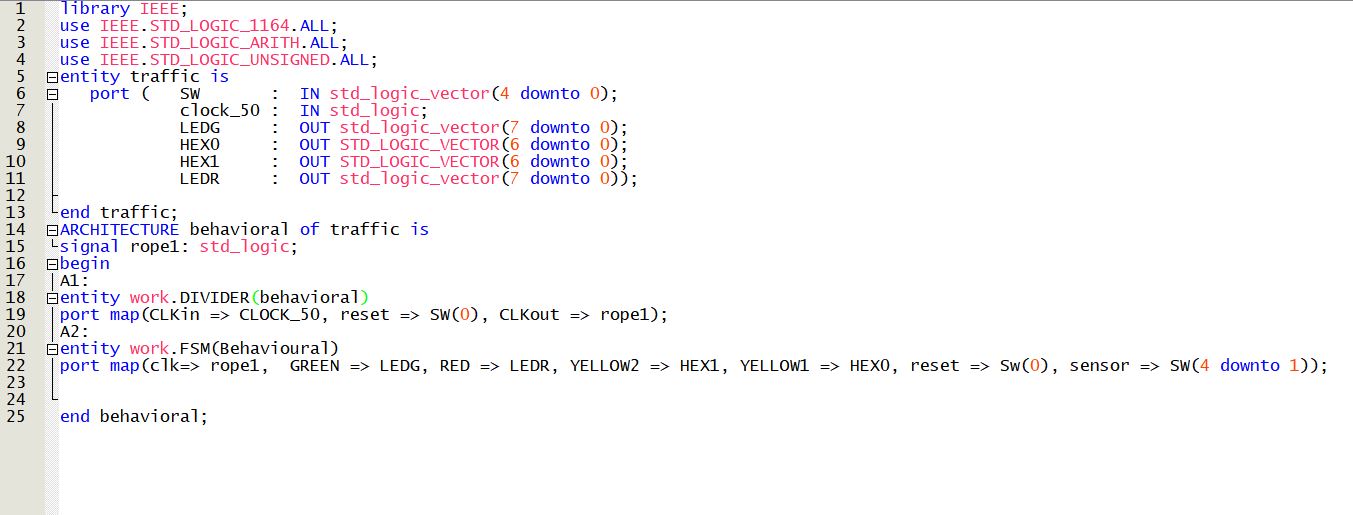


**TRAFFIC.VHD**

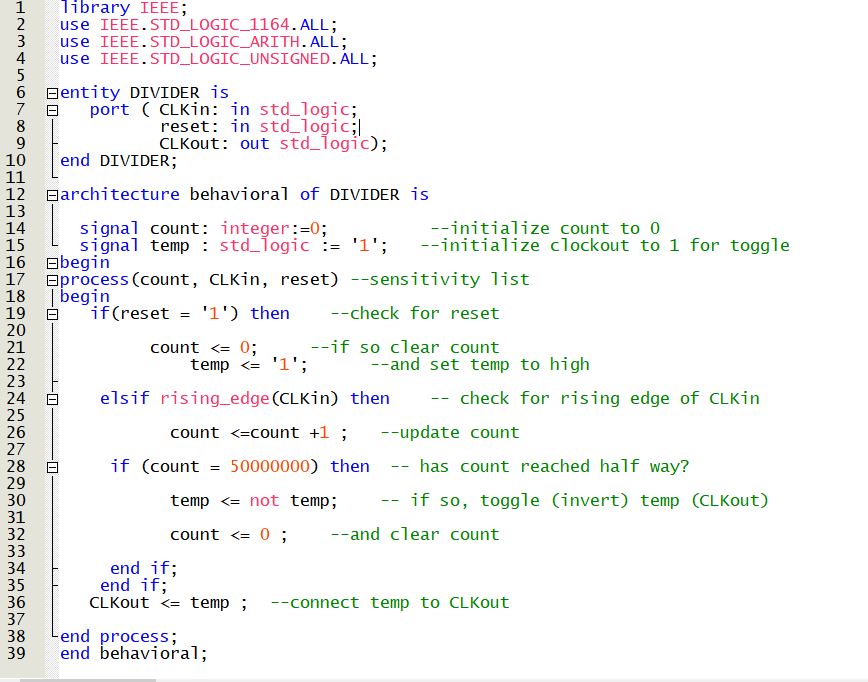


COMPLEX CODE VERSION II (LEDG, LEDR, and HEX outputs)

**TRAFFIC.VHD**



**DIVIDER.VHD**

****

**FSM.VHD**  
