DEVELOPMENT OF FPGA BASED SMART TRAFFIC LIGHT CONTROLLER SYSTEM WITH IMAGE PROCESSING

TAN KENG SHUEN

A project report submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering (Hons) Electronic Engineering

Faculty of Engineering and Green Technology
Universiti Tunku Abdul Rahman

January 2019

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	___________________________
Name	:	___________________________
ID No.	:	___________________________
Date	:	___________________________

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"DEVELOPMENT OF FPGA BASED SMART TRAFFIC LIGHT CONTROLLER SYSTEM WITH IMAGE PROCESSING"** was prepared by **TAN KENG SHUEN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons) Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	___________________________
Supervisor	:	Dr. Loh Siu Hong
Date	:	___________________________

ACKNOWLEDGEMENTS

I would like to express my deep sense of gratitude to my supervisor, Dr. Loh Siu Hong who has given me the golden opportunity to be involved in this project, and he helped me a lot with his invaluable advice, guidance, support, and encouragement during the development of the project. He has been an excellent advisor demonstrating good patience and enormous help throughout the development of the project.

Furthermore, I would also like to express my special thanks to my loving parents, colleagues and academic staffs who had helped and contributed to the successful completion of this project.

DEVELOPMENT OF FPGA BASED SMART TRAFFIC LIGHT CONTROLLER SYSTEM WITH IMAGE PROCESSING

ABSTRACT

As the population of the city, as well as the number of automobiles travel on the roads are increasing day by day, traffic congestion at the junctions is becoming a huge problem for many major cities nowadays. One of the reasons behind this traffic issue is the inefficiency of the techniques and algorithms used in the existing traffic light system which unable to adapt to the continuous changing traffic situation and eventually lead to traffic congestion spreads and occurrence of road accidents increase. Therefore, to prevent the situations further deteriorate, there is a pressing need for the introduction of advanced system and technology that able to improve the current traffic light control system to better accommodate this increasing demand.

In this project, an FPGA based Smart Traffic Light Control System (STLCS) is proposed with the aid of Computer Vision techniques in traffic signal control. The developed STLCS is comprised two development boards; Altera DE2 board and Raspberry Pi 3B+. Altera DE2 board is mainly used to execute the automated traffic light signalling system while the Raspberry Pi 3B+ is utilized to in-charge the computer vision tasks, for instance, video acquisition, processing, segmentation, object detection, etc. In this system, a wide view angle digital camera is fixed at the intersection road for real-time monitoring on four intersecting roads. A specific traffic lane will be captured when the Raspberry Pi received a request signal from the Altera DE2 board. The captured image of a particular road is processed through a series of image processing techniques for vehicle detection and count. After this, the detected number of vehicles is further used to calculate the proper time duration for controlling signal lights by using the dedicated timing algorithms. Next, the calculated time is sent to the FPGA traffic light controller via UART serial communication, and the traffic light controller will discharge the vehicles based on received time value. Once the green light countdown timer reached the limit, the yellow light is turned ON and a request signal is feed to the Raspberry Pi to calculate the green time for the next turn of traffic light controller, these processes are repeated continuously.

The smart feature of the system is able to compute the flexible green light time that dependent upon the detected traffic density on traffic lanes. Moreover, as a precautionary measure, a maximum and minimum green light time limit are fixed within the timing algorithm to prevent any vehicle detection error as well as over waiting of vehicles in queue in other lanes. In this project, ModelSim and Quartus II software are used to program and simulate the traffic light controller system that written in Verilog Hardware Description Language (HDL) and upload the program to **ALTERA DE2 CYCLONE II 2C35 FPGA** while the Syder IDE and OpenCV library along with Python, high-level programming language (HLL) are used to implement the computer vision program.

Keywords: Traffic Congestion, FPGA, Image Processing, Raspberry Pi 3B+, Altera DE2 FPGA, Verilog, Python

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

Ω	Ohm, resistance
V	Voltage
I	Current
e_n	Start-up loss time
S_{loss}	Total start-up loss time,
h	Saturation headway
sloss	Standard start up loss time
n	Number of detected vehicles
f	decrease factor
bps	bits per second

STLCS	Smart Traffic Light Controller System
TLC	Traffic Light Controller
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
RXD	Receive
TXD	Transmit
UART	Universal Asynchronous Receiver Transmitter
FSM	Finite State Machine
GPIO	General Purpose Input Output
RGB	Red, Green, and Blue
HSV	Hue, Saturation, and Value
OpenCV	Open source Computer Vision
UI	User Interface

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background of Study

It is undeniable that traffic congestion is one of the main challenges in today’s society and this issue is becoming serious day after day. Traffic congestion occurs when the available street capacity is not able to accommodate the traffic demand and there are several factors which cause traffic congestion, such as high rate of urban population growth, rapid increase in the number of automobiles, ineffective road construction strategies and management by government, and especially those non-adaptive traffic light system which is becoming obsolete and unable to handle high traffic density nowadays.

Traffic congestion has led to multiple negative impacts on our society and environment, for instance, wasting the road users’ time and delays which reduces productivity of employees, wasting the fuel and increasing the greenhouse gases (especially CO_2) emission which worsens the air quality and lead to global warming, interferes or blocks the passage of emergency vehicles which are travelling to their destination, etc. In order to overcome impacts from traffic congestion, the invention of the Smart Traffic Light Controller System (STLCS) is vital to provide smooth motion of vehicles in the transportation routes and also regulates the flow of vehicles through the traffic intersections of many roads.

In Malaysia or any developing country, the traffic lights that are widely seen nowadays consist of three lights: Green is the go sign, Yellow is the wait sign and Red is the stop sign, while in some urban areas, there are some traffic light control systems with countdown timers, which are able to improve the safety in the traffic by allowing drivers to make better and safer traffic decisions based on the remaining time of red or green light. Even though there are plenty of features that were implemented to the traditional traffic light system to improve the traffic flow efficiency, these traffic light control systems are still hardwired at the time of installation which means most of them are pre-programmed for a fixed duration for every change in the signals. These conventional traffic light control systems react accordingly to the traffic density on the roads and always keep constant during its operation, sooner or later, it will become one of the major causes that leads to several traffic issues due to the limited abilities to control the high traffic density in urbanized areas.

To avoid those limitations, a STLCS with the feature of adapt itself to the continuous changes is important. Therefore, the STLCS that is proposed will be image processing based adaptive signal controlling with FPGA traffic light control system which can fit in continuously changing traffic scenarios.

1.1.1 Image processing with OpenCV

OpenCV is an open-source computer vision and machine learning library introduced by Intel more than a decade ago. Since then, a lot of software programmers have contributed to the most recent library, therefore this makes OpenCV containing more than 2500 optimized computer vision algorithms in 2009 (Ivan Culjak, 2012). Nowadays, the latest version of OpenCV is 4.1.0 (OpenCV 4), this version is the most functional, fastest and stable OpenCV ever because it is supported by a variety of programming languages such as C++, Python, etc. and supported on many operating platforms including Linux, Android, and macOS. (Alexander et al, 2014)

Image processing is a digital signal processing method that utilizes the image processing algorithms in the OpenCV library to manipulate every pixel from an image or video to obtain the required information. (Gholamreza Anbarjafari, 2014). Image processing which treats input image or video frame as a 2D signals (x,y) and after this apply signal processing operations to them and retrieve an output image that contains the extracted characteristics associated with input image (Rose Mary, 2011). With the rapid growth of technologies today, image processing becoming popular due to highly applicable in various aspects of business, free and open source. Thus, it forms a core research area within computer science and artificial intelligence.

Image processing includes the following steps:

- Import an image or video via acquisition tools which can either be software-based source or hardware-based source.
- Manipulating and analysing the image using image processing algorithms.
- Output can be altered or extracted image.

In the proposed system, OpenCV installed Raspberry Pi 3B+ and digital camera module are used to acquire the image, process, and estimate the number of vehicles on each lane.

1.1.2 ALTERA Development and Education Board

Altera DE2 Development and Education board is an FPGA board that widely used for the development of FPGA design and implementations. The purpose of the Altera DE2 is to provide users an ideal working station for design prototyping in multimedia, networking, and storage. The state-of-the-art feature available within the DE2 board offers a favourable practical environment for university and college courses, for different levels of digital systems project design and development (Terasic Technologies, 2012). In the Altera DE2 board FPGA board, it consists of Cyclone II 2C35 FPGA chip which offers low costs, high densities (35000 logic element (Les) in a 672- pin package) and more features with exceptional performance. Therefore, all the dedicated components mounted on the board like communication ports, switches, display, I/O pins, etc. are connected to the pins of this chip so that the users can manipulate all aspects of the board's operation.

Altera DE2 with Cyclone II 2C35 FPGA consists of abundance of features and interfaces to accommodate various application needs. Altera board provides an adequate number of LEDs, switches, and seven-segment displays that suitable for a simple project or experiment. While for the advanced project, there are SDRM, SRAM, as well as liquid crystal display can be used. Furthermore, for a project that requires I/O interfaces and processor, the user can easy to instantiate the Altera's Nios processor and communication ports such as RS-232, USB 2.0, and PS/2. Moreover, Altera DE2 board also provides standard connectors for the line-in, microphone, line-out audio decoder, video-in (TV) and VGA (10-bit DAC) for student or researcher to conduct project related to analysis and processing of digital signal (Philipp, C. et al, 2019). Lastly, the DE2 board also offers USB 2.0 connectivity, 100/10 Mbps Ethernet network, an infrared (IrDA) port, and an SD memory card connector.

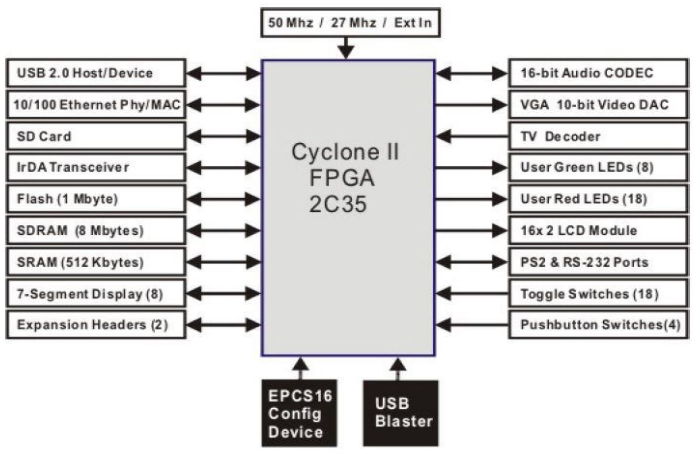


Figure 1.1 : Block Diagram of Altera DE2 board (Philipp, 2019)

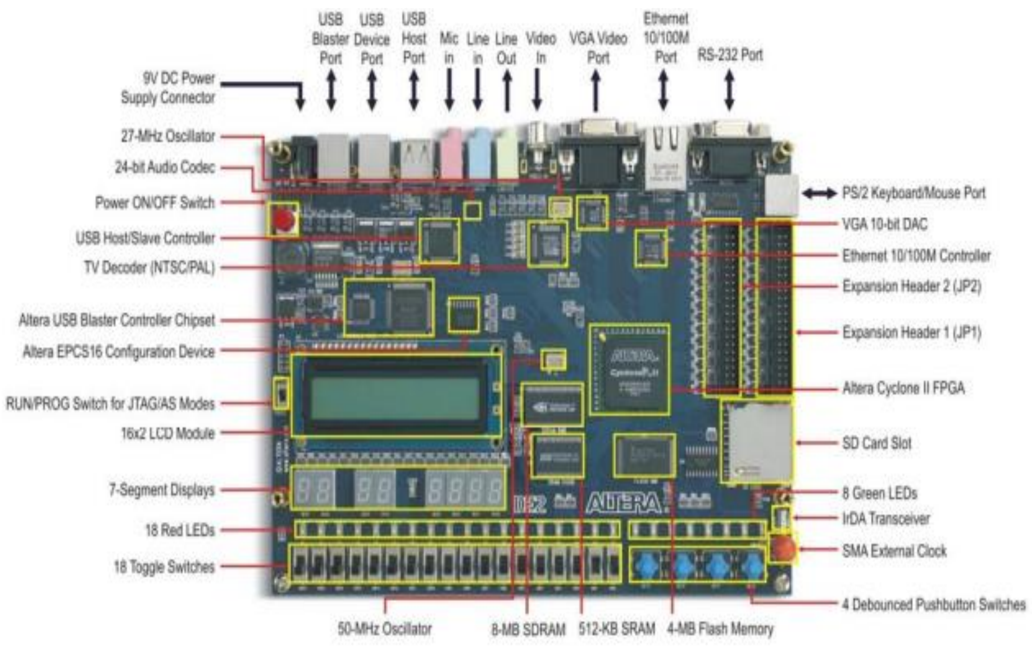


Figure 1.2 : Altera Development and Education Board (Terasic Technologies, 2012)

In the proposed system, the Altera DE2 FPGA Board is used to construct a Traffic Light Controller (TLC). The design flow of the TLC using FPGA board is shown in the figure below. In the circuit description process, Verilog HDL is used to program the TLC software and then followed by the functional simulation and synthesis. The design flow is followed until the time simulation and then the generated file is downloaded into the Altera FPGA Board.

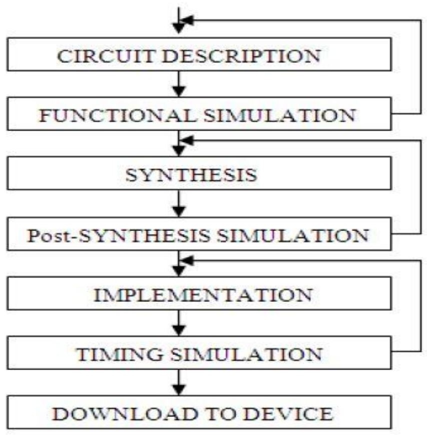


Figure 1.3 : FPGA Design Flow (Dilip, 2012)

1.1.3 Hardware Description Language (HDL)

Verilog is a Hardware Description languages (HDL) used to program the FPGA chips. Verilog is relatively recent, and the programming concept is quite similar to the C programming language, therefore, it allows the hardware design programmer to easily familiarize and understand the programming styles that are used in Verilog. Moreover, Verilog is a weakly typed programming language and it can be used to design digital hardware at any level of abstraction. For example, Behavioural level, Register-transfer level, and Gate level. In Behavioural level, the behaviour of a system is described with a set of instructions or algorithms that are executed in sequence, for example, Verilog uses always block, functions, and task to perform sequential instructions in behavioural level design. Furthermore, in the Register Transfer Level, the characteristic of a digital system is described using operations (i.e. always, assign, etc). Operations in the register transfer level play around with the data transfer between the registers, and it must be synchronized with the clock cycle. In Gate level, it refers to the logic level design of a digital system that comprises a network of gates and registers instanced from the technology library. It uses predefined logic primitives like NOT, AND, NOR, and other gates in the design.

Nowadays, one of the two most commonly-used languages in digital hardware design is Verilog HDL and the other is VHDL. In this project, Verilog HDL is used to write the TLC and UART serial communication program into the Altera DE2 board. The reason Verilog HDL is selected because Verilog programming is easy to understand and write compare to VHDL since the programming concept is quite similar to C programming.

1.1.4 Universal Asynchronous Receiver Transmitter (UART)

UART is a communication interface that widely used for budget, short distance, and offers moderate data exchange speed between two computers or between computer and peripheral devices. UART is designed for the asynchronous serial communication by serializing a byte of parallel data (8-bits) at the transmitter with same extra overhead bits using transmitter shift register (TSR) and vice versa at the receiver site (Neha R. Laddha et al, 2013). UART transmit a byte of data sequentially one bit at a time from the transmitter and receive the byte of data at the destination by decoding sequential data with control bits. Since the entire transmission processes does not associate with clock input from source, hence it is termed as asynchronous communication.

UART communication requires only two independent lines (RXD, TXD) for full-duplex data communication. TXD is referred to transmit side, the transmission line is at idle as not data to transmit. While the RXD is referred to receive site, which is the input of the UART device. Since the type of data used for UART communication is in binary form, therefore are only two states in the signal lines; HIGH (logic 1) and LOW (logic 0).

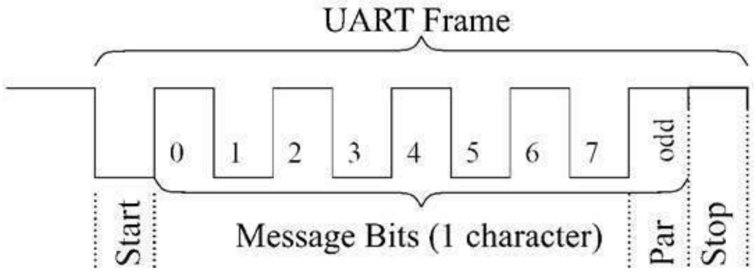


Figure 1.4 : UART Frame Format (Neha, 2013)

From the UART frame format above, when the transmitter is idle, logic 1 state will be transmitted continuously to the receiver site. When a byte data is given to the UART for asynchronous transmission, a "START BIT" with logic 0 (LOW) will be added to the beginning of the byte that is to be transmitted and function of this "START BIT" is used to tell the receiver module that 8-bit data (1 byte) is ready to be received and at the same time, the baud rate generator will synchronize the receiver's clock with the clock in the transmitter. After the “START BIT” the Least Significant Bit (LSB) of data bits are being sent first, an 8-bit “Message BIT” is sent bit by bit until the Most Significant Bit (MSB). After the 8-bit binary data has been sent to receiver, a Parity Bit may be added at the end of transmitted data that is used for error checking. After this, a “STOP BIT” is sent by the transmitter to the receiver to inform the receiver that the data was completely transferred. In the receiver, regardless of whether the data was received correctly or not, the UART will just ignore the START BIT, PARITY BIT and STOP BIT.

1.1.5 Raspberry PI 3B+

The Raspberry Pi (RPi) is the world’s most inexpensive, credit-card-sized single-board computer that can be plugged into a computer monitor, and uses peripheral devices like keyboard and mouse. RPi is one of the most popular microprocessors that is widely used by the students in Universities to carry out their projects that are related to image or video processing, IoT based application, robotics controller, server application, etc. Raspberry Pi Foundation officially provides an optimized OS, Debian based Raspbian for developer to interface with the hardware, but users can install several third-party OS like Ubuntu, Windows 10 IoT Core, RISC OS, etc (Umesh Lokhande, 2017). The Raspberry Pi with installed Raspian OS have GUI that capable of doing everything you would expect from a desktop, from document editing, internet browsing, and playing games to simple graphic design. Although the performance of the RPi is incomparable with a typical laptop or desktop, it gives the user flexibility to access over the on-chip hardware, for example GPIOs can be used for developing an application by connecting to the external components like analogue to digital converter (ADC), LEDs, motor, etc.

Raspberry Pi 3B+ Specifications:

- Broadcom BCM2837 64-bit Quad-Core 1.4GHz
- 1GB RAM
- 2.4GHz and 5GHz Wireless LAN and Bluetooth 4.1
- 4 x USB ports (USB 2.0)
- Composite video port and 4 pole Stereo output
- Full size HDMI port
- 15 pins CSI camera port
- Micro SD slot for storing operating system and data
- Supports 40 GPIO Pins

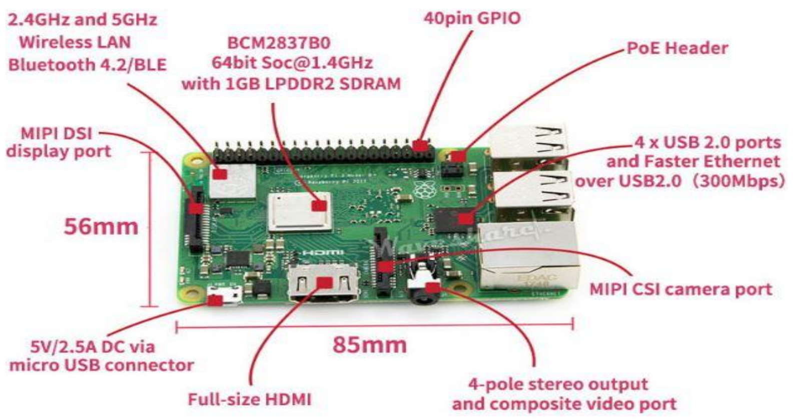

2.4GHz and 5GHz Wireless LAN Bluetooth 4.2/BLE
BCM2837B0 64bit Soc@1.4GHz with 1GB LPDDR2 SDRAM
40pin GPIO
PoE Header
MIPI DSI display port
4 x USB 2.0 ports and Faster Ethernet over USB2.0 (300Mbps)
56mm
MIPI CSI camera port
5V/2.5A DC via micro USB connector
85mm
Full-size HDMI
4-pole stereo output and composite video port

Figure 1.5 : Raspberry Pi 3 Model B+

1.1.6 Colour Model

Colour Model is a colour system that utilizes abstract mathematical model to create a full range of colours from the three primary colours which are Green, Red, and Blue colour (RGB). There are two categories of colour models, the additive and the subtractive model. Each colour model serves different purposes because of slightly different in creatable colour range which is shown in the figure below. In additive colour model like RGB, the colours perceived are the result of transmitted light, therefore, the additive model requires light to create colour, and usually used in digital display. While for subtractive colour models like CMYK, the colours perceived is the result of reflected light, as a result, subtractive colour model is uses in printing inks.

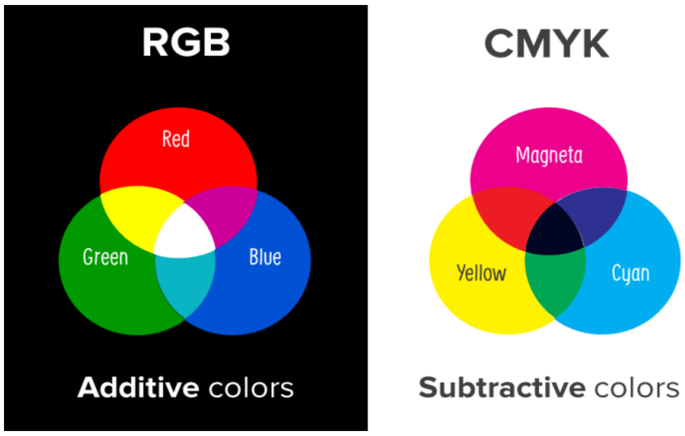


Figure 1.6 : Additive (RGB) and Subtractive (CMYK) Colour Model

HSV (hue, saturation, and value) is a subset of RGB colour model, but it is an improved version of RGB colour model which is capable to describe the colours in terms of hue, saturation, and value, therefore, this colour model is useful in image processing tasks regarding to the colour-based segmentation. From Figure 1.7 below, the cylindrical shape of the HSV colour model tells that the variation of the saturation is determined by the percentage of white component adding into the colour space, high saturation represents no white component (shows pure colour), while unsaturated represent shades of grey. Moreover, hue represents the colour type and it ranges from 0 to 360° with red at 0, blue at 240° green at 120° and so on. While for value channels, it describes the brightness or intensity of the colour.

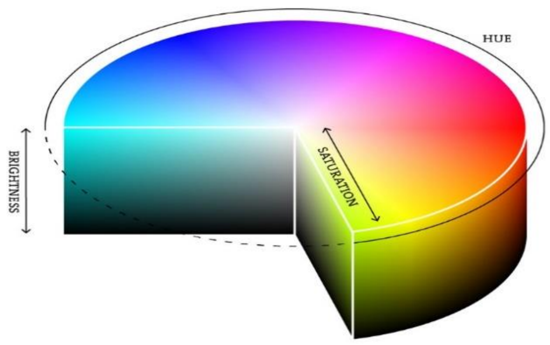


Figure 1.7 : HSV Colour Model (Andrey, 2014)

1.2 Problem Statements

As urbanization increases, the conventional traffic light control system becomes insufficient and incapable to handle a huge amount of traffic due to the pre-programmed traffic light algorithms and management system. As a consequence, road users have to face the following drawbacks:

1. Heavy traffic jams.
   -Heavy traffic congestion has substantially increased at certain lanes before and after office hours, however, the conventional traffic light system still provides a fixed duration of green signal to the lanes that having congestion and the lanes that free of vehicles, as a result, it leads to heavy traffic jam.

2. Waiting for empty roads
   -There are times where there is no cars on certain intersecting roads, but the traffic light still give the green signal for that road. This is because the existing traffic light system is pre-programmed and the road users need to wait until the light turns to green and this is wasting the time of the road users.

3. Requires more manpower to control the traffic
   -During certain festivals, there will be a higher traffic flow on roads. Owing to the constant timer in the existing traffic light system, it will lead to serious traffic congestion. As a result, a lot of manpower is required to control and maintain the traffic order.

Thus, to minimize the mentioned drawbacks above, there is a need to develop a real-time operating smart traffic light controller system (STLCS) that able to provide a proper timing signal that depends on continuously changing of traffic density at the traffic junction. Accurate to detect and count the vehicles and signal timing calculation according to traffic density will be the evaluation points of the performance for the proposed intelligence traffic light system.

1.3 Aims and Objectives

This project aims to design and develop an adaptive Smart Traffic light Control System (STLCS) to improve the efficiency of the existing automatic traffic light signalling system by reducing the waiting time of each lane of the vehicles and maximize the flow of vehicles across a traffic intersection given the dedicated timing algorithms to calculate the green time.

The objectives in this project are:

i. Design and develop a workable automatic Traffic Light Control System using Altera DE2 board with Verilog programming.

ii. Familiar with the knowledge related to image processing and utilize the techniques in the project using Raspberry Pi, OpenCV library and Python programming.

iii. To implement a communication system on both Raspberry pi and Altera DE2 board.

iv. Implement a workable Smart Traffic Light Controller System with prototype model.

1.4 Thesis Organization

Chapter 2 provides a brief review of the approaches as well as studies that have been done by other researchers. Throughout this chapter, we are able to identify the invaluable source of knowledge from other people who is working in the same field and the gaps in current knowledge, so that we can avoid reinventing something that has already been done. In Chapter 3, methods and approaches were described, and this allows us to have more systematic and clearer implementation and design system to accomplish the proposed project. In Chapter 4, the outcomes of this proposed approaches were attached and discussed. Lastly, Chapter 5 gives a summary of this project as well as discusses on some directions for future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, few papers including a variety of approaches and techniques used to implement a smart traffic light controller were researched and reviewed. Literature reviews of these papers are classified into three parts which include recent techniques used to develop a Smart Traffic Light Controller, FPGA based Traffic Light Controller System and Image Processing based Traffic Light Controller System. In these papers, different approaches that were used in the implementation of those traffic light systems will be identified, and the advantages and disadvantages of the techniques that is used in the papers will also be discussed. Furthermore, places where new contributions as well as improvements of those limitations could be made was found to avoid those limitations recurring in an endless cycle. Lastly, a brief summary of the literature review is shown in Table 2.3 below.

2.2 Background of Study

2.2.1 Smart Traffic Light Controller System

Smart Traffic Light Controller System (STLCS) is always an active research topic due to serious traffic problems that worsening the transportation system in urban cities. The structure of STLCS is sophisticated because it integrates various systems, for instance, detecting system, adaptive control system, communications system, etc. those systems are combined for the purpose of providing efficiency, safety, mobility and comprehension on traffic control. There is a plethora of researchers from different disciplines collaborating to find out feasible solutions to reduce traffic congestion. In this chapter, proposed methodologies and techniques that are used to construct an STLCS using microcontroller, manufactured devices, and sensors in the literature will be discussed and analysed.

2.3 Research on Techniques applied in Traffic Light Controller System

The following are some journal researched which contain the different techniques are applied to implement a STLCS.

2.3.1 Review of Journal: *Intelligent Traffic Light and density control using IR sensors and microcontroller* by Sinhmar, P.

Based on the paper (Sinhmar, P., 2012), the researcher implemented an intelligent traffic light system using 89V51RD2 (MCS-51 family) microcontroller with IR transmitters and receivers as traffic density sensing device. In her proposed system, assembly language was used to program the traffic light system into the microcontroller and synchronized it with the external hardware - IR sensors. The IR sensors were mounted on either side of roads respectively and this enables the microcontroller to make a comprehensive timing decision based on the number of vehicles detected by sensors at each intersecting road. During the operation, the IR system will be triggered whenever any obstacle like vehicles passes on the road between the IR transmitter and receiver. After this, the microcontroller will increase the count number of vehicles and the number will be stored in microcontroller's memory. Based on the different vehicles count, the microcontroller will define different ranges for each traffic light delays and kept update the delays accordingly.

In this research, the system will record the number of vehicles count in memory at a predefined recording interval on real time basis. Therefore, this indicates the intelligent traffic light controller has a database system to store the number of vehicles at a specific time interval. These recorded data can be downloaded from the microcontroller to the PC through the serial communication method so that the user can analyse the traffic condition at respective traffic lights connected to the system. The administrator can perform some function to traffic light such as update the light delays or algorithms parameters, erase the memory and download the data by using the command system in the microcontroller. Figure 2.1 below illustrates the architecture of the Intelligent Traffic Light Controller using IR sensors developed by Sinhmar, P.

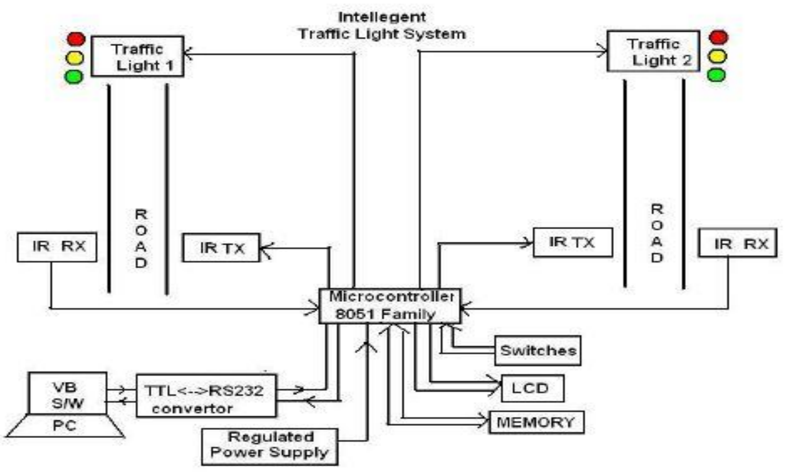


Figure 2.1: Architecture of the Smart Traffic Light System (Sinhmar, 2012)

2.3.2 Review of Journal: ***Traffic Control System Using Inductive Loop Detector*** by Rakesh, V.S. et al

Based on the journal (Rakesh, V.S. et al, 2007), researchers implemented a TLC system using the inductive loop detector that is placed below the surface of the roadway which as shown in Figure 2.2 below.

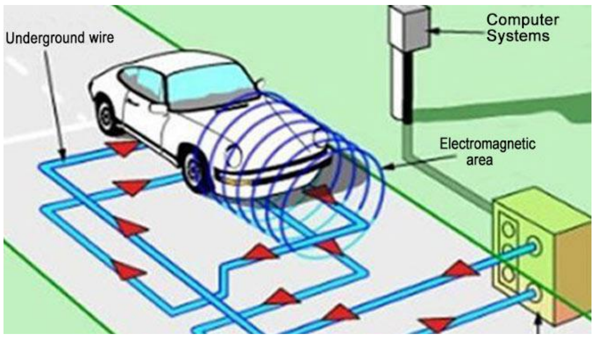


Figure 2.2 : Inductive Loop based Traffic Light Controller System (Tom, 2001)

In the proposed system, the inductive loop was supplied with an AC current, this makes the inductive loop system act as a tuned electrical circuit and lead-in cable are the inductive elements (V.S. Rakesh et al, 2007). When a vehicle enters the inductive loop, the electromagnetic field produced by the AC current in the loop sensor will induce a small current in the vehicles. According to Lenz's law, the induced eddy current in the vehicle will generate its own electromagnetic field that opposite direction to the electromagnetic field from the sensor coil, this presence of eddy current will reduce the inductance of inductive loop. As a result, the decrease in inductance tends to reduce the impedance and trigger the oscillator circuit to send a pulse signal to the data acquisition system to indicate the presence of a vehicle and then used as an input for traffic light control unit. Figure 2.3 below shows the block diagram of the inductive loop based TLC system.

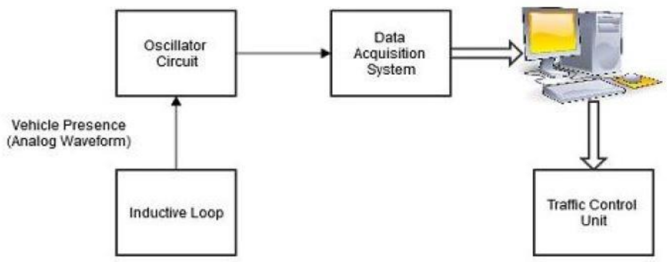


Figure 2.3 : Block Diagram of Inductive Loop based Traffic Light System (Rakesh, 2007)

Rakesh, V.S. et al pointed out that the change in amplitude of loop inductance in an inductive loop will give the corresponding information such as type (car, bus, or bicycle), speed, count, and length of the vehicle to the traffic control unit. Thus, researchers conducted studies to analyse three types of inductive loop structures which as shown in Figure 2.4 below. From the Figure 2.4, the small loop designated as Loop 1 was used to detect small vehicles like bicycle, motorbike, etc., whereas the Loop 2 was used to detect the large size vehicles like cars, buses, trucks, etc. However, there is a limitation in both Loop 1 and Loop 2. For example, when bicycles or other small size vehicles pass over the Loop 2 detector, there will be no changes in the loop inductance or induced current, as a result, this may lead to missing detection and eventually affect the outcome of traffic light controller. Hence, researchers developed a new inductive Loop 3 to overcome the limitations by merging both inductive Loop 1 and 2. Since inductive Loop 3 comprised characteristics of both Loop 1 and 2 structures, therefore, Loop 3 can detect any vehicle regardless of the size of the vehicle passing over.

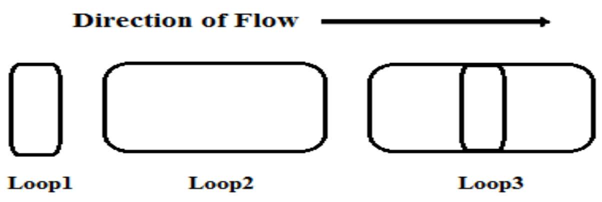


Figure 2.4 : Types of Inductive Loop Detectors (Rakesh, 2007)

Table 2.1 below shows the result that was done by researchers to compare the true and experimentally vehicle count using their new inductive Loop 3 in the traffic light controller system. From the result, the proposed system is able to detect the vehicle with around 97% accuracy.

Table 2.1 : Comparison between True Count and Experiment Count using Inductive Loop Detector (V.S. Rakesh et al, 2007)

Vehicle type	**True count**	**Experimental count**
Bicycle	10	9
Car	10	10
Bus	10	10

2.3.3 Review of Journal: *Intelligent Real Time Traffic Controller Using Image Processing* by Nidhi D. Agrawal et al.

As for the research (Nidhi D. Agrawal. et al, 2013), the researchers applied image processing techniques in traffic light management system by using digital cameras to perform real-time monitoring on the traffic condition on intersecting roads. In this research, there are several image processing operations used to extract the information from the video frames to detect the vehicles.

1. Image Acquisition: Uses a web camera to monitor traffic status. When there are no vehicles on the road, the image of the road is captured as a reference image for other image processing algorithms, for example, background subtraction or frame differencing. Next, the acquired image is converted into grayscale and to a binary image.

2. Image Enhancement: Involves operation like brightening, sharpening, etc.
, to extract certain detail and features from an unfavorable image.

3. Image restoration: Remove or reduce the noise that degrades an image to improve the outcome for further image operations.

4. Image segmentation: Partitioning an image into its constituent objects. The segmented output data is used for analysis or served for representation.

5. Morphological processing: Involves two basic operations Erosion and Dilation.
Erosion operation uses to shrink the size of the foreground mask object while dilation operation uses to expand the size foreground mask.

The proposed intelligent traffic light controller system by Nidhi D. Agrawal et al combines both traffic surveillance and traffic control technologies. This research emphasized the different types of edge detection techniques that are used to detect the presence of the vehicles by identifying and locating the boundaries of the vehicles in every frame (Nidhi D. Agrawal. et al, 2013). From Table 2.2 below, researchers compared the performance of edge detection techniques such as Boolean, Marr-

Hildreth, Sobel, Prewitt and Canny edge detection used in vehicle detection. From the result, it is known that by using appropriate operations, the performance of vehicle detection can be highly accurate.

Table 2.2 : Comparison the Performance of Edge Detection Techniques (Nidhi, 2013)

Image sample	Actual no.	Boolean	Marr-Hildreth	Sobel	Prewitt	Canny
1	4	2	6	2	2	4
2	3	0	4	1	1	2
3	4	2	3	2	3	4
4	5	2	3	2	3	6
5	5	2	3	3	3	5
6	7	3	5	3	2	6
7	4	1	5	1	1	4
8	5	2	5	3	2	5
9	3	0	3	0	1	2
10	6	4	3	2	3	6
Accuracy		39.13%	84.78%	41.30%	45.65%	93.47%

2.3.4 Comparison of the Techniques Used by Other Researchers

In this chapter, it can be concluded that traffic sensing devices are important to indicate the presence of vehicles and feed the information to the smart traffic light controller system for adaptive signal control. From the research above, the vehicle sensing devices can be categorized into two types, intrusive and non-intrusive types (V.S. Rakesh et al, 2007). Inductive loop detector proposed by V.S. Rakesh et al is one of the intrusive sensors that are placed below the surface of the roadway, whereas infrared (IR) sensors proposed by Sinhmar, P. and video image processor by Nidhi D. Agrawal et al belong to non-intrusive type that are installed above the roadway. A comparison among those sensors used in the smart traffic light controller system from all the paper is in the Table 2.3 below.

Table 2.3 : Comparison between Methods Applied by Other Researchers to Implement a Smart Traffic Light Controller System

Author	**Type of sensor**	**Pros**	**Cons**
P. Sinhmar, (2012)	IR sensors	-Easy to construct. -Detection system is inexpensive. - System is dynamic since the proposed system has ability to collect the data, and modified the algorithm parameters.	- Sensors need to be well protected or secured in a safe place. -Vehicle counting may not accurate due to: pedestrian pass through, two vehicles across in parallel or vehicle stop at between the sensors.
Rakesh, V.S., and Shaithya, V. (2007).	Inductive loop detector	-High accuracy in vehicle detection. - Detection will not be affected by weather. -Able to count and determine the types of vehicle.	- Expensive and difficult to install. - Maintenance process may obstruct the traffic flow.
Nidhi, D. A. and Amit, S. (2015).	Image processing	-High accuracy. -Relatively inexpensive and easy to implement. -Can use for both surveillance and traffic control system at the same time. - Image quality and performance of vehicle detection can be improve by utilize suitable image processing algorithms.	-Image or video quality is depends on digital camera used and affected by the weather.

2.4 FPGA Based Traffic Light Controller System

2.4.1 Review of Journal**: *FPGA Implementation of an Advanced Traffic Light Controller using Verilog HDL*** by Dilip, B. et al.

Based on the research (Dilip, B. et al, 2012), a low-cost advanced TLC was implemented in the hardware using Spartan-3E FPGA with Verilog HDL programming. In this research, a few design steps were conducted by researchers during the implementation of the FPGA based TLC system. Firstly, a TLC flowchart which is as shown in Figure 2.5 below was constructed. TLC flowchart is critical in the TLC implementation process because it illustrates the predefined sequential order of the light signals that are interchanged between the traffic lights and also provides a general mapping for the programmer in the design regarding the flow of the program. From the TLC flowchart below, the criteria that must be included in the TLC flowchart are defining the action, for example which colour of the light should be turned ON or OFF for certain traffic light, describing the sequence of the actions, for instance, which traffic light should light up the green light first then follow by the next traffic light (phase) and also the sequence of the three lights: red, yellow and green in each traffic light. Lastly, arrow symbols were used to show the sequence and the relationship between them.

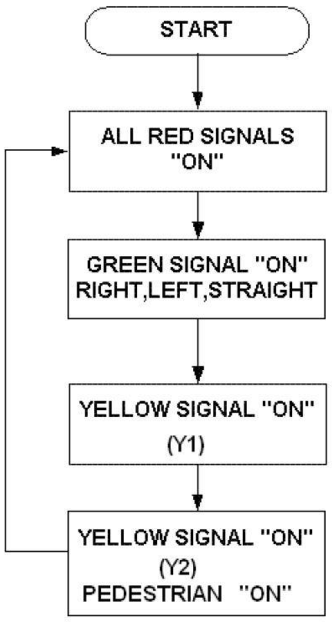

START
ALL RED SIGNALS "ON"
GREEN SIGNAL "ON" RIGHT,LEFT,STRAIGHT
YELLOW SIGNAL "ON" (Y1)
YELLOW SIGNAL "ON" (Y2) PEDESTRIAN "ON"

Figure 2.5 : Proposed TLC Flowchart (Dilip, 2012)

After designing the general flow of TLC, the TLC state diagram was used to illustrate the behaviours or actions of the system. From the state diagram below, the proposed TLC system composed of a finite number of states that represent the different actions in the system. For example at state cnt=00, dir=00, green light in the north traffic light will be turned ON while red signal light in the other traffic lights will be turned ON, while at state cnt 00, dir 01, the green light of the east traffic light will light up whereas the other traffic lights will turn ON the red signals. The transition between those states will occur when there are changes in input condition or triggered related variable like timing out.

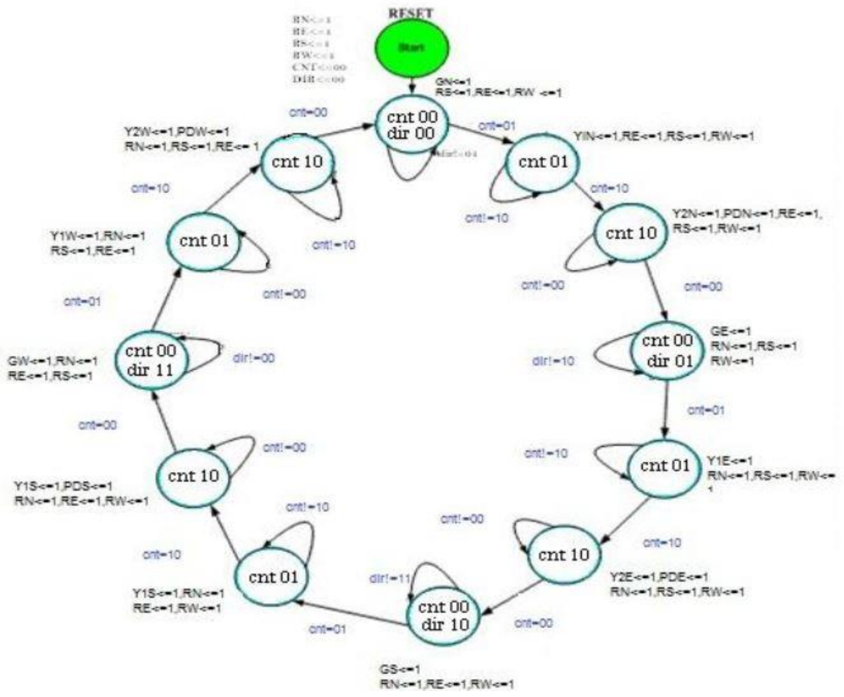


Figure 2.6 : Proposed TLC State Diagram (Dilip, 2012)

.

The final step is the hardware implementation. In this step, the designed state machine of the TLC is described or coded using the Verilog Hardware Description Language and is dumped into Spartan-3E FPGA trainer kit. The I/O pins from the FPGA were connected to the external LEDs as shown in Figure 2.7 below.

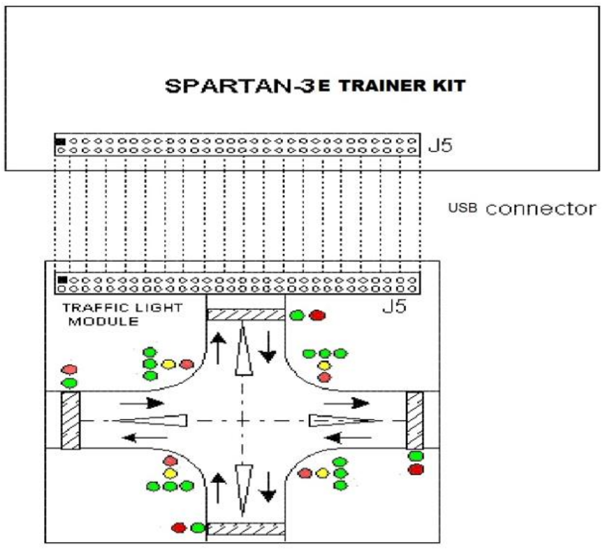


Figure 2.7 : Hardware Implementation of FPGA based TLC (Dilip, 2012)

2.4.2 Review of Journal: ***Field Programmable Gate Array Based Intelligent Traffic Light System*** by Agho, O et al.

In this journal (Agho, O et al, 2015), researchers designed and implemented an FPGA based intelligent traffic light controller system using Xilinx Spartan-6 LX16 FPGA with the aid of infrared (IR) sensors. In the proposed system, the IR sensors play crucial roles to detect the vehicles at each intersecting road and send the information to the TLC system, the FPGA based Intelligent Traffic Light Controller System will then respond accordingly to adjust and achieve the optimal traffic signal setting so that the vehicles discharge rate is maximized. In this paper, each oncoming lane in the traffic junction was installed with infrared transmitter and receiver sensors which is as shown in figure below.

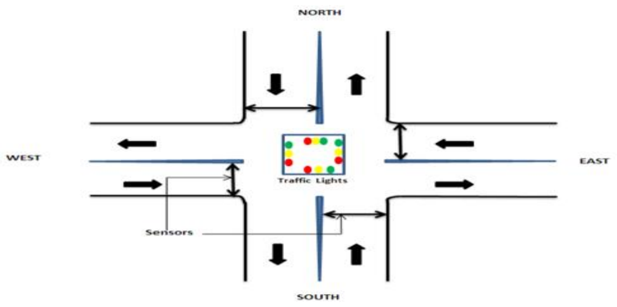


Figure 2.8 : Model of Traffic Junction (Agho, 2015)

The proposed traffic light system operates depends on the vehicles detected by the IR sensors. When there is no vehicle detected in each lane, all the red traffic signals will be ON. While if there is a vehicle detected in one lane, the traffic light will turn ON green light only for that lane, and at the same time, other traffic lights will remain red. If vehicles are detected in every lane, the traffic will be allowed to move in a sequence: North (highest priority), East, South and then the West direction. The proposed traffic light controller consists of two major components: using state machine to keep track on the current state and the next state of the traffic and using counter to control the transition from one state to another. In the design process, finite state machine design was used by the designers to define the behaviour of the system into several states and then sequentialize together. From the table below, it shows a state table of the proposed traffic light system by researchers based on specifications above.

Table 2.4 : State Table for the Intelligent Traffic Light Controller Proposed by Researchers (Agho, O et al, 2015)

State	**State Description**	**NORTH**	**EAST**	**SOUTH**	**WEST**
S0	All red signals on	Red	Red	Red	Red
S1	NORTH	Green	Red	Red	Red
S2	EAST	Red	Green	Red	Red
S3	SOUTH	Red	Red	Green	Red
S4	WEST	Red	Red	Red	Green
S5	NORTH	Yellow	Red	Red	Red
S6	EAST	Red	Yellow	Red	Red
S7	SOUTH	Red	Red	Yellow	Red
S8	WEST	Red	Red	Red	Yellow

After constructing the state table, the flow of each state was described using a flowchart and state diagram as shown in the Figure 2.9 and Figure 2.10 below.

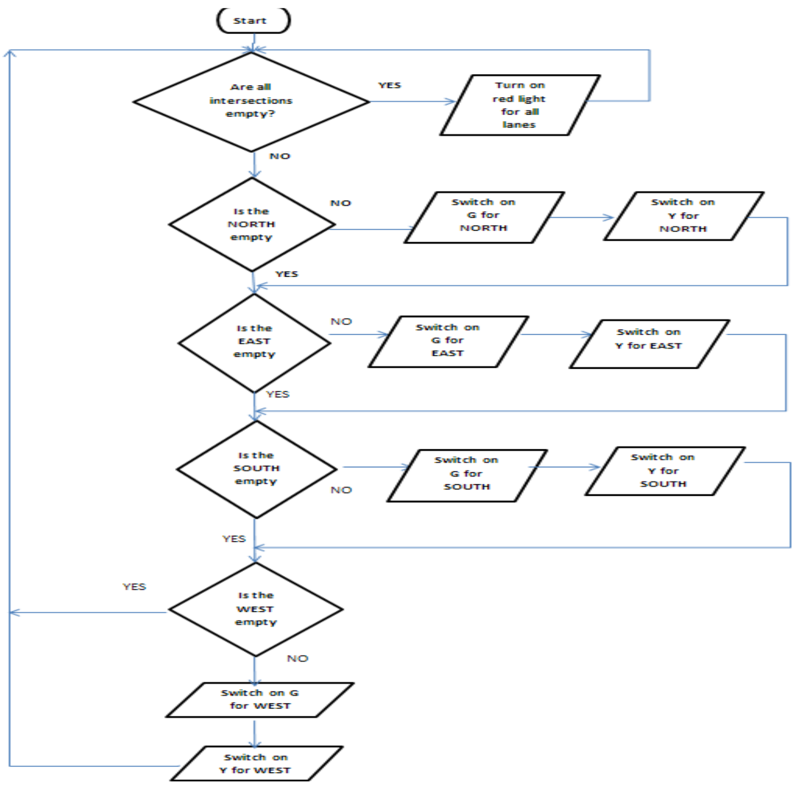


Figure 2.9 : Proposed Intelligent Traffic Light System Flowchart (Agho, 2015)

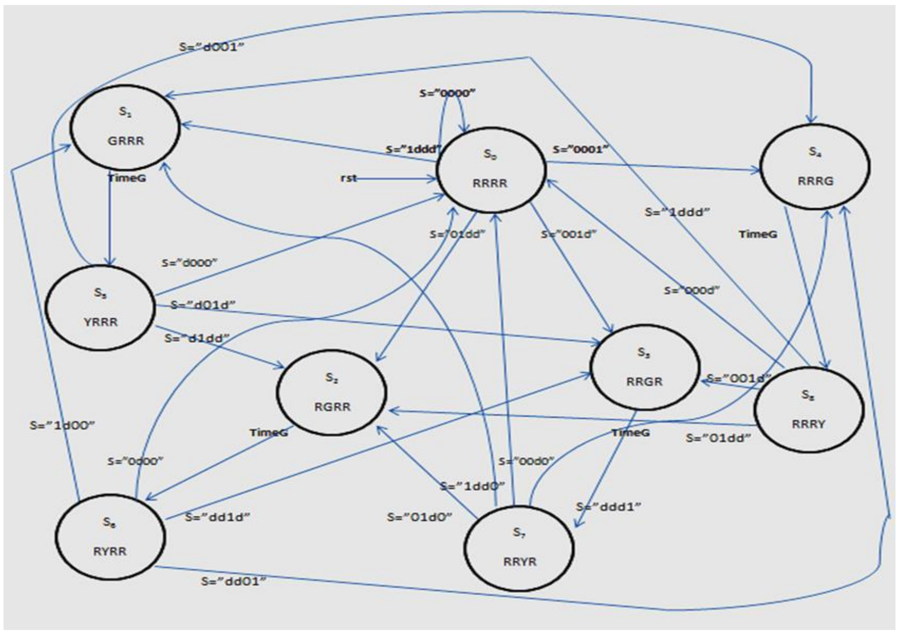


Figure 2.10: Proposed Intelligent Traffic Light System State Diagram (Agho, 2015)

2.4.3 Summary for FPGA based Traffic Light Controller System

From the papers above, most of the FPGA based TLC system was developed using a finite state machine (FSM) design. This is because simplicity makes any system easy to be designed or implemented and quick in execution. To summarize the research above, there are a few processes involved in FSM design.

1) Separate the actions or outputs of the system into a finite number of states.
2) Draw a state table to organize all the current and next states.
3) Construct a general flowchart that illustrates how the system flow.
4) Build a state diagram that shows the order and transition between the action states.
5) Use hardware description language to program the system.
6) Simulate, test and optimize the program.

2.5 Image Processing Based Traffic Light Controller System

2.5.1 Review of Journal: ***Design of Real Time Smart Traffic Light Control System*** by Almawagani, A.H.M.

Based on the journal (Almawagani, A.H.M., 2018), a STLCS using image processing techniques to control the traffic signal was implemented with the aid of MATLAB software and Arduino microcontroller. The use of MATLAB software in this system is focused on the image processing part, while the Arduino microcontroller was used as a TLC. The proposed STLCS consists of four web cameras that are connected to a computer with installed MATLAB software. After this, series image processing algorithms in MATLAB software was utilized to process the video frames from each camera for vehicle detection and counting. The number of vehicles detected is used as an input parameter of timing algorithm to calculate and allocate a proper green time for particular lane to discharge the vehicle. If a certain road is high in traffic load, the system will calculate and assign a longer time for this crowded road compared to the other less congested roads. Figure 2.11 below shows the block diagram of the developed system.

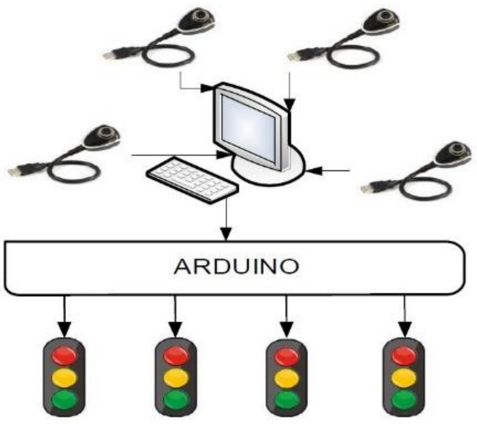


Figure 2.11 : Block Diagram of the Proposed System (Almawagani, 2018)

Feature of the proposed system is able to adjust its timing algorithms with time changes; daytime and night-time. In daytime, system will carry out daytime mode. In the daytime mode, the system will apply background subtraction algorithms to obtain a foreground mask between the background reference image and continuous moving frames. After this, RGB to grayscale conversion will be used to convert 3 channel foreground image (red, green and blue) into 1 channel image (black and white) to reduce computation power for further image processing operation. Furthermore, filtering techniques will be applied to remove the noises and lastly, bwboundaries function is used to find the boundaries of the detected object and then count the number of object. However, in the night-time, the researchers had proposed a solution to cope with insufficient light condition issue and they called it night-time mode. In night-time mode, it operates quite similar to daytime mode, the system will detect the front headlights of vehicles rather than direct detect and count the vehicles. Since every car has two headlights, the number of detected headlight divided by two will be the number of the car. Figure 12 and Figure 13 shows the vehicle detection at daytime mode and night-time mode

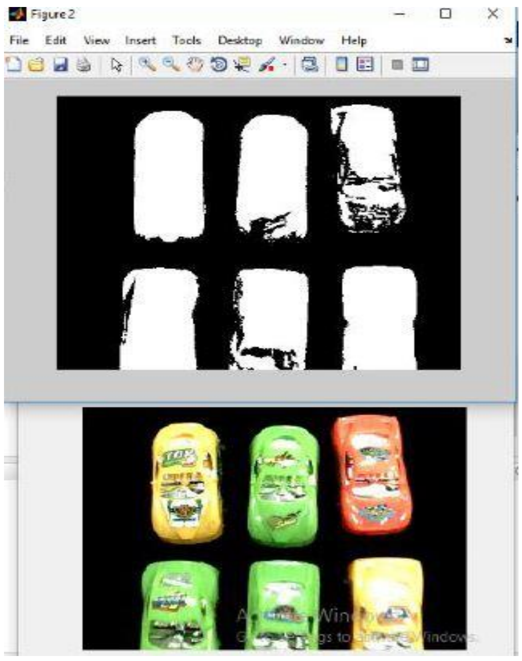


Figure 2.12 : Vehicle Detection at Daytime Mode (Almawagani, 2018)

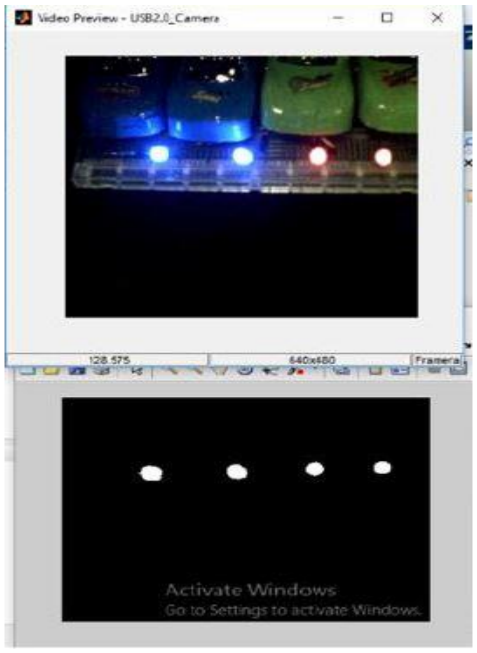

Video Preview - USB2.0_Camera
128.575
640x480
Activate Windows
Go to Settings to activate Windows.

Figure 2.13 : Vehicle Detection at Night-time Mode (Almawagani, 2018)

Lastly, the proposed system will use the estimated number of vehicles inside the lanes to calculate the total time to complete one cycle. After the cycle period was calculated, the weight factor method was used to divide the cycle period into each lane according to the number of vehicles detected. Figure 2.14 below shows the prototype of the proposed system.

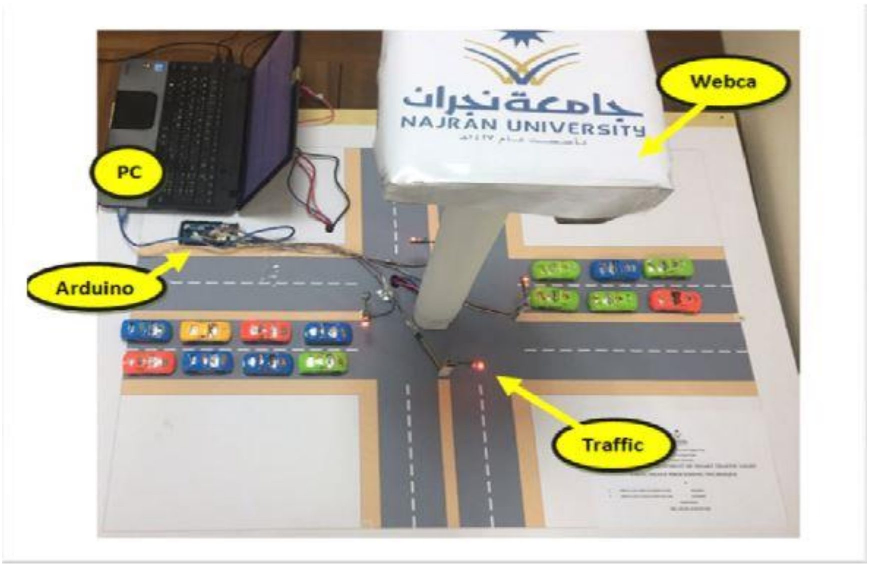


Figure 2.14 : Prototype Model of the Proposed Smart Traffic Light System (Almawagani, 2018)

2.5.2 Review of Journal**: *Smart Control of Traffic Signal System using Image Processing*** by Parthasarathi, V. et al.

As for the journal (Parthasarathi, V. et al, 2015), the researcher implemented a smart traffic signal system using the image processing techniques. In the system, the web camera was mounted at a certain height and provides real-time monitoring on each intersecting lanes. After this, the real time frames of the traffics obtained from the webcam are transfer to the PC with MATLAB installed for video processing. In video processing, the traffic density estimation was done by calculating the number of vehicles in each lane. In addition, throughout this research, Parthasarathi, V. et al pointed out that the emergency vehicles get stuck in traffic congestion is a serious issue nowadays, thus they added a feature in their system which able to detect the emergency vehicles like ambulance. In this research, researchers come out with a solution with regards to traffic density and the presence of emergency vehicles by setting a signal priority condition to the traffic light system from high priority to low priority as shown below.

1. If multiple emergency vehicles are detected in a different lane, the lane with emergency vehicle closest to the traffic signal will be given highest priority.

2. If an emergency vehicle is detected, the signal priority will be given to that lane.

3. If no emergency vehicle is present, the signal priority is given to the lane with the highest traffic density.

4. If traffic density is the same in the different lanes, no signal priority is given to any lane and the traffic signal will follow the pre-set timer.

With regards to the image processing techniques involved in this system, these techniques includes the following steps. First, image acquisition in which live stream frames of the road were acquired from the web camera and a background image with no traffic load will be taken as a reference frame. The figures below shows the background reference frame and the current frame obtained.

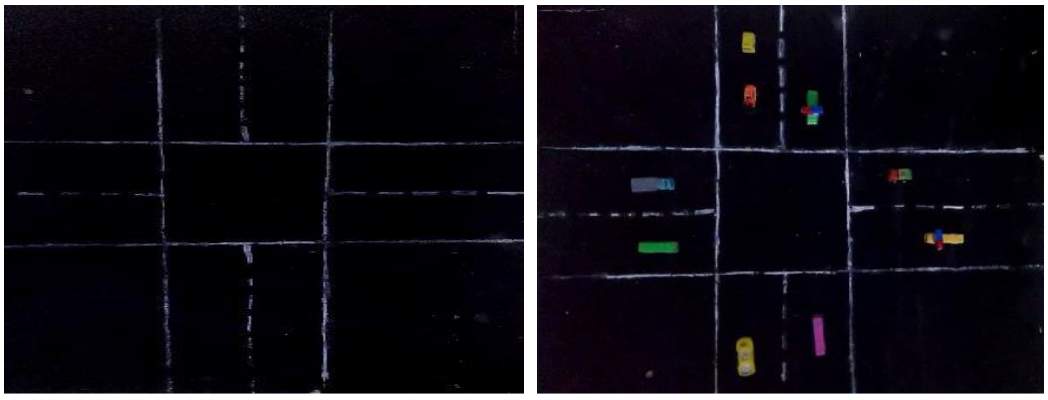


Figure 2.15: Background Reference Frame (left) and the Real Time Frame (right) (Parthasarathi, 2015)

Next, crop the region of interest from the frames, so that the system will focus on the cropped region and eliminate the details in unwanted regions. In this system, lanes of roads were cropped from the reference frame and real time frame and were used as the region of interest. Figure 2.17 and Figure 2.18 below shows the region of interest of the reference frame and real-time frame.

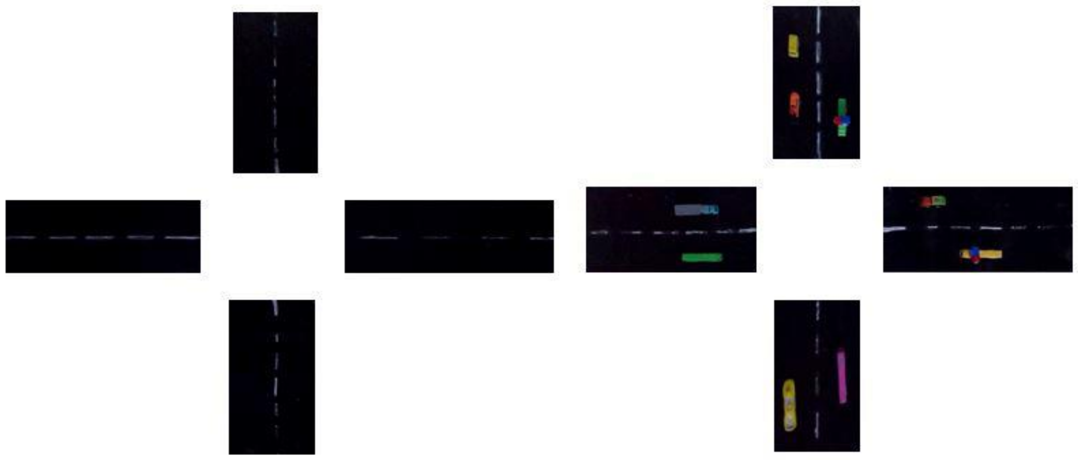


Figure 2.16: Cropped Reference Frame (left) and the Cropped Real Time Frame (right) (Parthasarathi, 2015)

Furthermore, for emergency vehicle detection, the image segmentation operation based on colour will be carried out to detect blue and red colour from the siren of an ambulance. If there is a red and blue colour object detected, the distance between red and blue objects is calculated. If the calculated distance is fall within the predefined threshold distance, the system will be considered it as the presence of an ambulance and gives signal priority to that lane. From the Figures below, it shows a cropped frame from a certain lane and red colour and blue colour detected using image segmentation technique.

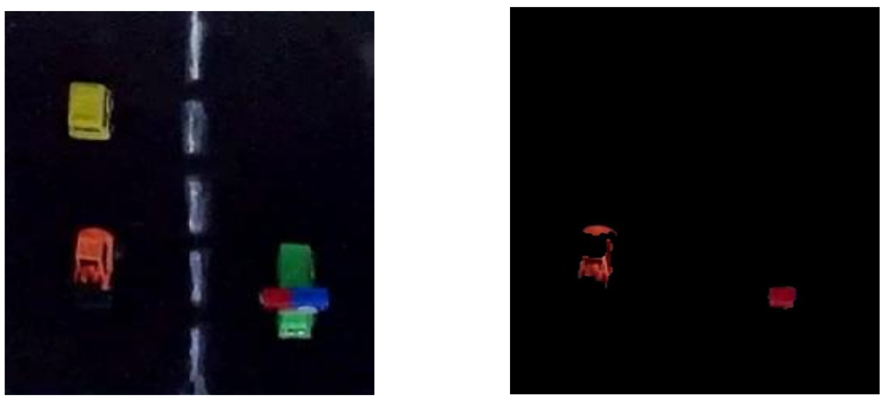


Figure 2.17: Cropped Frame at Certain Lane (left) and Red Colour Detected (right) (Parthasarathi, 2015)

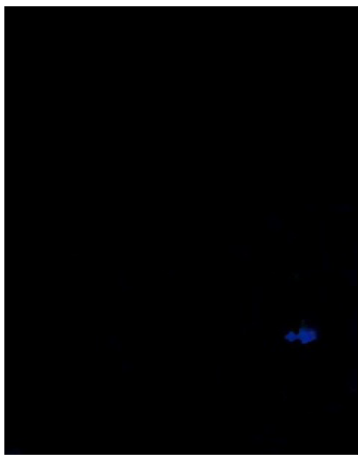


Figure 2.18 : Blue Colour Detected (Parthasarathi, 2015)

Next, for traffic density estimation, the reference frame and real time frame were converted to grayscale image using RGB to grayscale conversion algorithm and the background subtraction method was used to subtract grayscale real time frame with a grayscale reference frame to obtain foreground masks which indicate the presence of vehicles on the road. Furthermore, several morphological operations are applied to remove the unwanted noises existing in the 8-bit grayscale image. After this, the enhanced image is further converted to binary image and filtered by Gaussian filter operation to acquire only vehicles on the road.

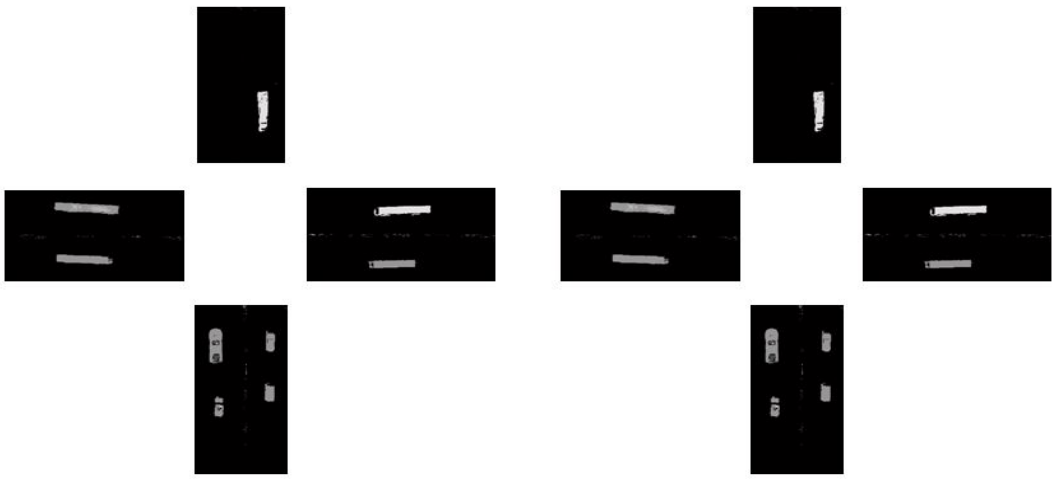


Figure 2.19 : Grayscale Image after Background Subtraction (left) and the Enhanced Binary Image (right) (Parthasarathi, 2015)

Lastly, a specific image operation was used to draw the boundaries of white regions in a binary image. The number of boundaries found will be used to indicate the number of vehicles. From Figure 2.24 below, it shows the boundaries of the white region was connected.

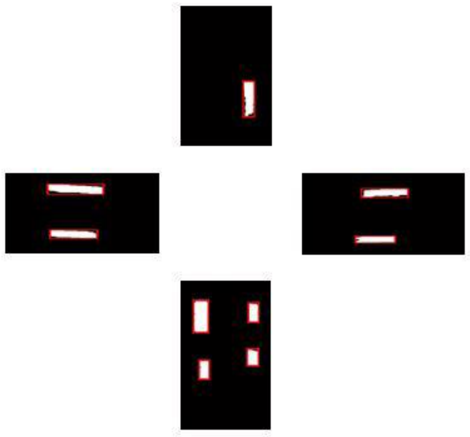


Figure 2.20 : Boundaries of the Objects (Parthasarathi, 2015)

CHAPTER 3

METHODOLOGY

3.1 Proposed Project Architecture

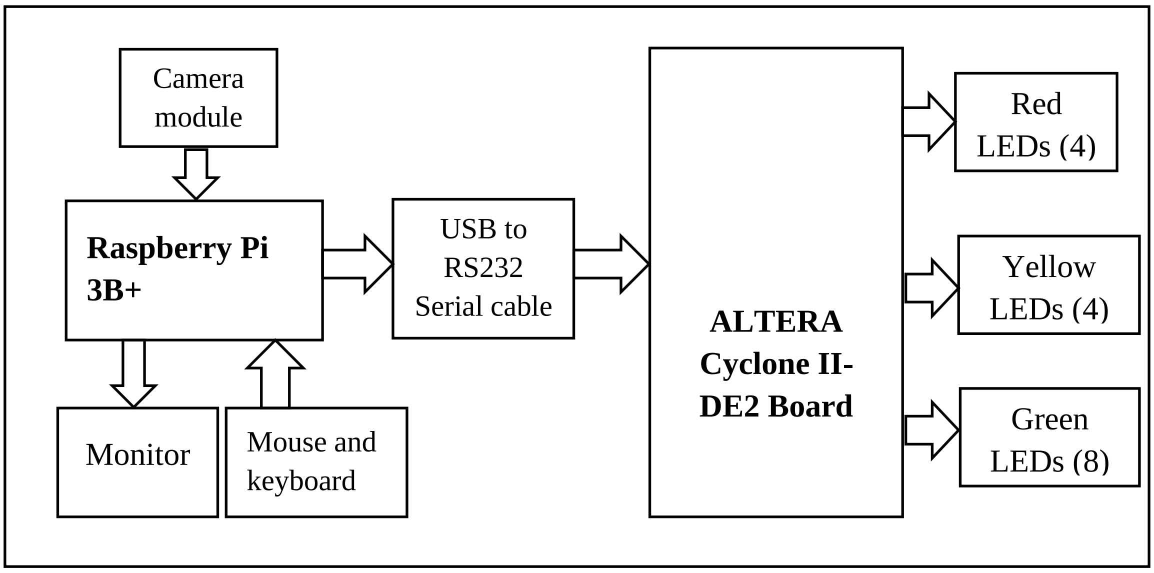


Figure 3.1 : Block Diagram of the Proposed System Architecture

Hardware Modules:

- Raspberry Pi 3B+
- Traffic lights circuit
- Serial cable
- Monitor
- Camera module
- Altera DE2 Board

Software Modules:

- Spyder IDE
- ModelSim
- Quartus II

3.2 System Working Principle

3.2.1 General flow of the system

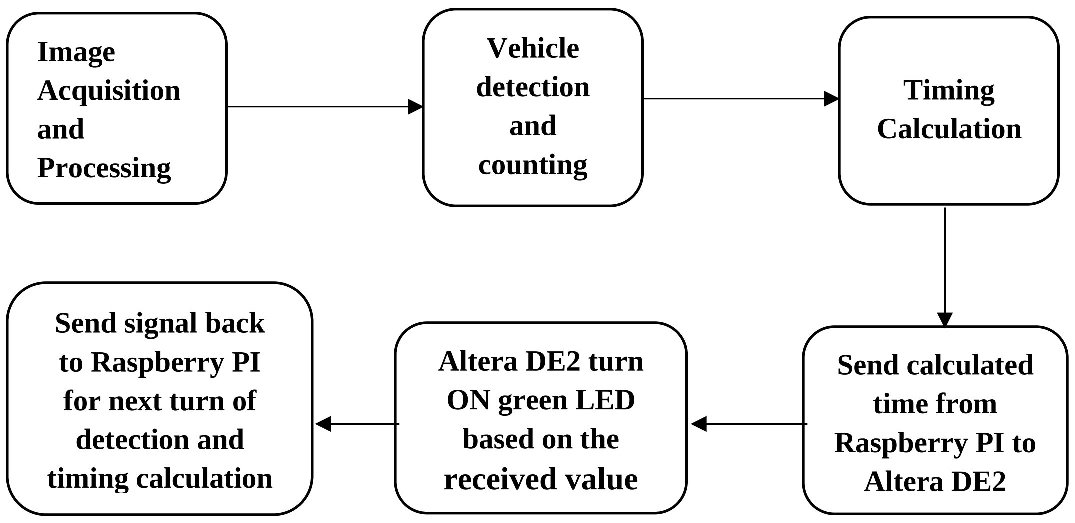


Image Acquisition and Processing
Vehicle detection and counting
Timing Calculation
Send calculated time from Raspberry PI to Altera DE2
Altera DE2 turn ON green LED based on the received value
Send signal back to Raspberry PI for next turn of detection and timing calculation

Figure 3.2 : General Flow of the System

The Figure 3.2 above shows the general flow of the developed Smart Traffic Light Controller System (STLCS) with image processing. In the first step, a wide-angle camera module is used to monitor and capture the traffic lanes at the junction. Next, image processing techniques such as image enhancement, restoration, morphological process, segmentation are applied to the captured image for queuing vehicle detection and count on particular traffic lane. Moreover, the number of vehicles counted on a particular lane is further used in the dedicated timing algorithm to calculate the proper green time to maximize the traffic flow across the junction. After timing calculation, the calculated time is sent via UART serial communication to Altera DE2 which function as TLC, the TLC will display the green light according to the received time value. After the green light, the yellow light will be turned ON and the Altera DE2 Board will send a request signal to Raspberry Pi to perform timing calculation for the next turn of green light and return the calculated value to Altera DE2 to display it. These processes are repeated to provide optimal traffic flow at the junction.

3.2.2 Flowchart of Proposed System

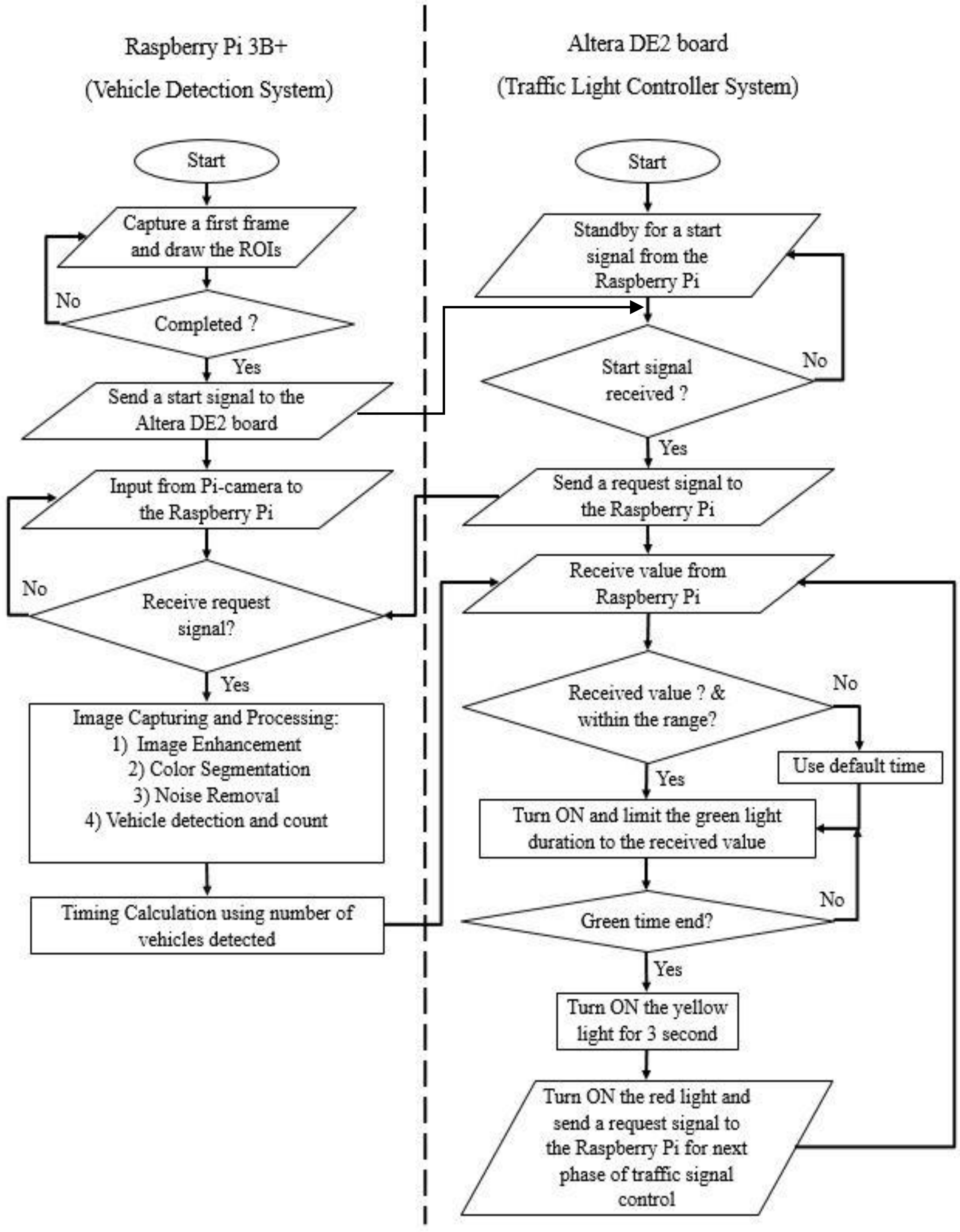

Raspberry Pi 3B+
(Vehicle Detection System)
Start
Capture a first frame and draw the ROIs
Completed ?
No
Yes
Send a start signal to the Altera DE2 board
Input from Pi-camera to the Raspberry Pi
Receive request signal?
No
Yes
Image Capturing and Processing:
1) Image Enhancement
2) Color Segmentation
3) Noise Removal
4) Vehicle detection and count
Timing Calculation using number of vehicles detected
Altera DE2 board
(Traffic Light Controller System)
Start
Standby for a start signal from the Raspberry Pi
Start signal received ?
No
Yes
Send a request signal to the Raspberry Pi
Receive value from Raspberry Pi
Received value ? & within the range?
No
Yes
Use default time
Turn ON and limit the green light duration to the received value
Green time end?
No
Yes
Turn ON the yellow light for 3 second
Turn ON the red light and send a request signal to the Raspberry Pi for next phase of traffic signal control

Figure 3.3 : Flowchart of the Proposed STLC System.

3.3 Project management

The schedule of the project is shown in the Gantt chart below.

Table 3.1 : Gantt chart for FYP 1

Activity/ week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Project Proposal	■													
Implement project idea			■	■	■	■	■							
Literature Review									■	■	■			
Methodology						■	■	■	■	■	■			
Oral presentation														■

Table 3.2 : Gantt chart for FYP 2

Activity/ week	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Project work (continue)	■	■	■	■	■	■	■	■						
Results and Discussions						■	■	■	■					
Conclusion and recommendations								■	■					
Report Checking											■	■		
Final Report Submission													■	
Oral Presentation														■

3.4 Image Processing Operations

3.4.1 Flowchart of Image Processing Operations

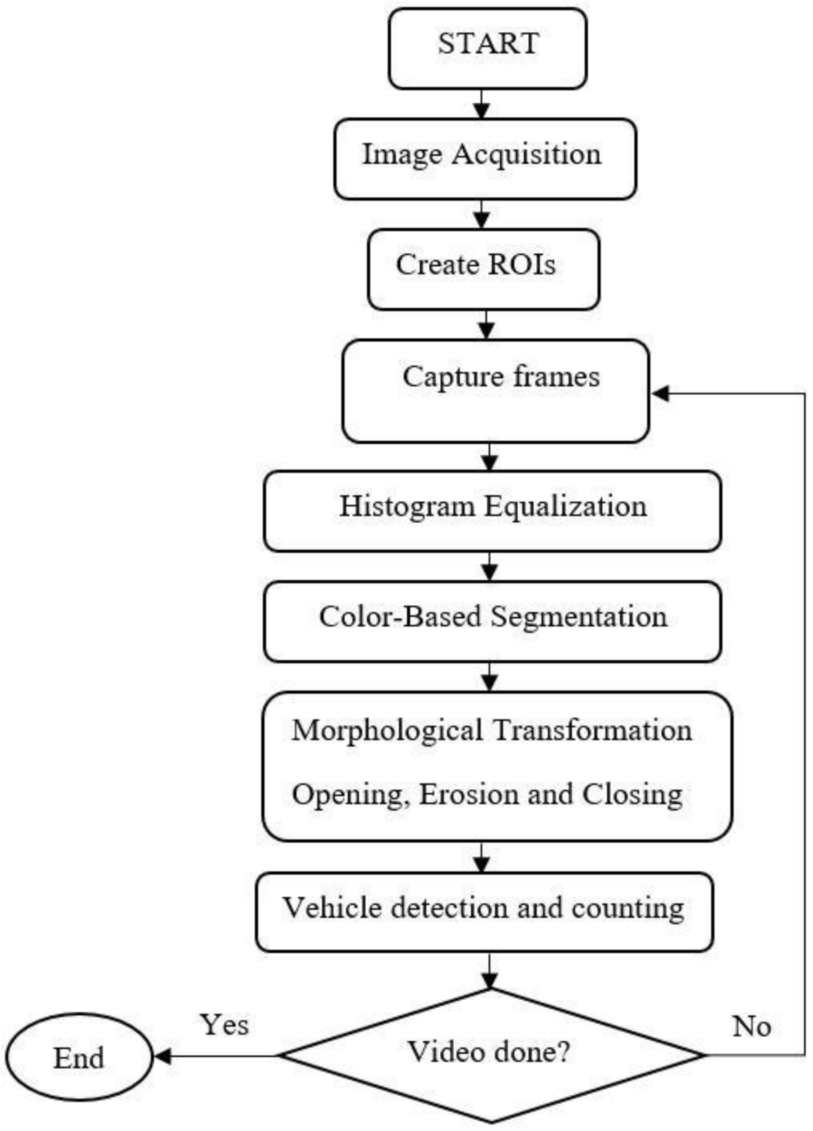


Figure 3.4 : Image Processing Flowchart

3.4.2 Image Acquisition

Image acquisition is an action to retrieve an image from the external source. In the proposed system, image acquisition is performed by the hardware-based source which means the Pi camera module as shown in Figure 3.5 is used to capture the real-time image of intersecting roads. Image acquisition plays an important role and a few precautions need to be taken before proceeding to coming image processing steps. First, the camera module must be placed at a position to ensure that the camera's field of view covers the junction roads which is as shown in Figure 3.6 below. Next, owing to the limited focal length, the lack of autofocus abilities, and the relatively low resolution of the Pi Camera, the distance between the camera module and traffic prototype model becomes critical which may affect the image clarity and give rise to some image noise that cannot be removed through image processing techniques. Therefore, the position of the camera is vital and required gradual trial and error attempts during the installation process for the better detection process.

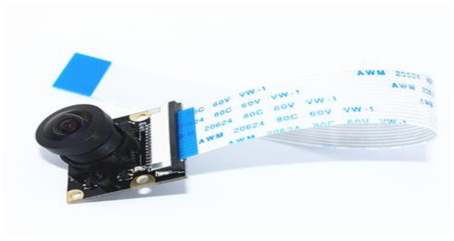


Figure 3.5 : Pi Camera Module

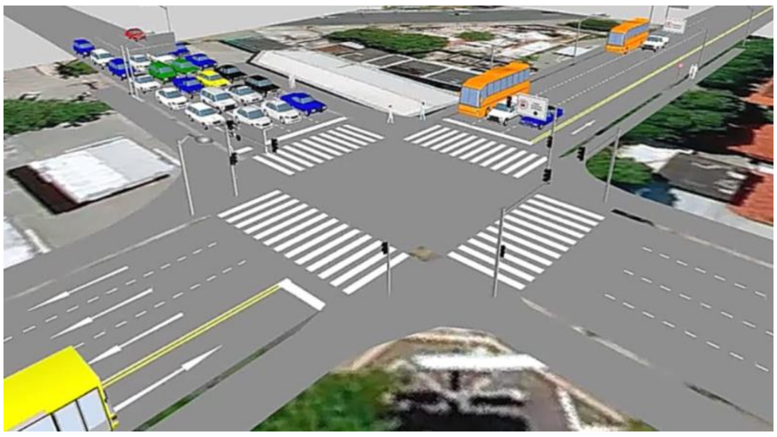


Figure 3.6 : Camera's Field of View

3.4.3 Region of interest (ROI)

Region of interest (ROI) is referred to a portion or region of an image that consists of useful or meaningful information and may be extracted out with the need image operations. Region of interest (ROI) is a useful function in this smart traffic light project because it is used to provide the boundaries for vehicle detection and ignore the interferences or noises appearing outside the boundaries. In this project, ROIs are created on each of four intersecting roads at the junction which is as shown in Figure 3.7 below. Vehicles that are present within the ROIs created will be detected using segmentation operation.

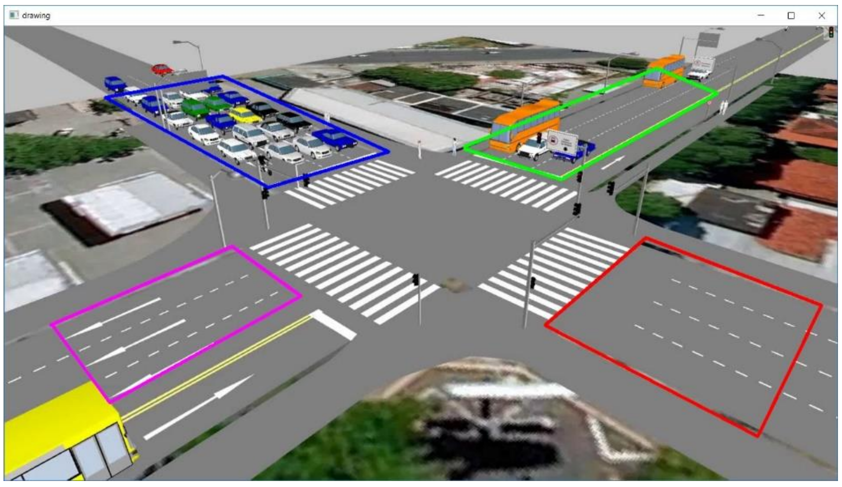


Figure 3.7 : Region of interests (ROIs) selection

3.4.4 Image Enhancement (Histogram Equalization)

Histogram Equalization is an image enhancement technique that modifies or adjusts the image intensities to have a better contrast image. The working concept of histogram equalization is basically "equalizing" the original intensity histogram, so that the output image contains a uniform distribution of intensities. From Figure 3.8 below, before applying the histogram equalization, the colour intensity of a colour image is low in dynamic range which means a certain range of intensity levels are held by many pixels than other intensity levels. As a result, the image shows fewer details. However, when the histogram equalization is applied, it will alter the image by adjusting the probability density function (PDF) of the original histogram so that all the range of intensity levels are spread to each pixel almost equally, as a result, the image will have more uniform distribution of intensities, improved image quality, enhanced brightness and at the same time more details in low illumination image.

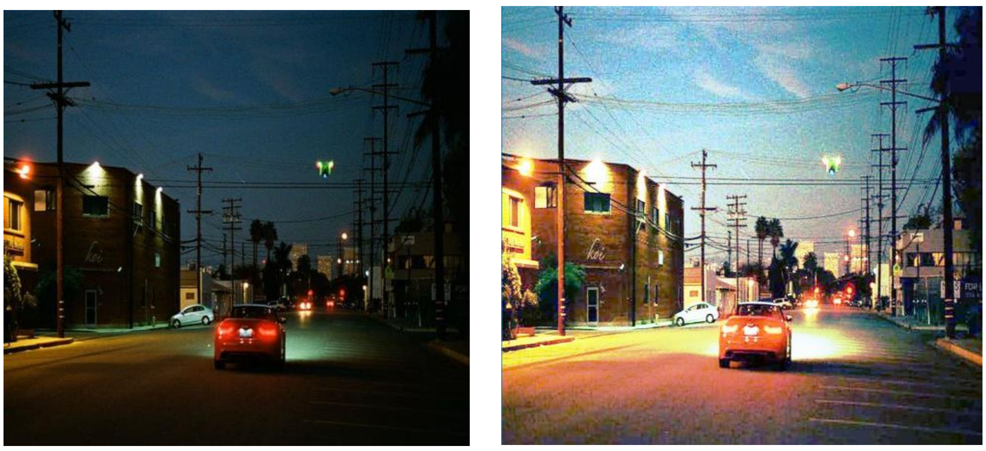


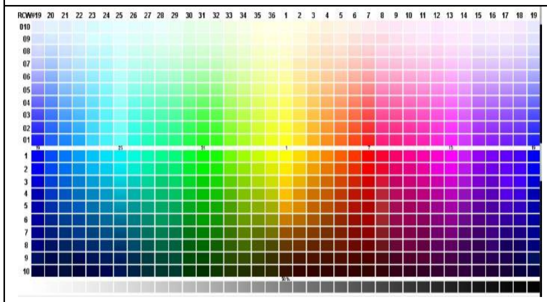
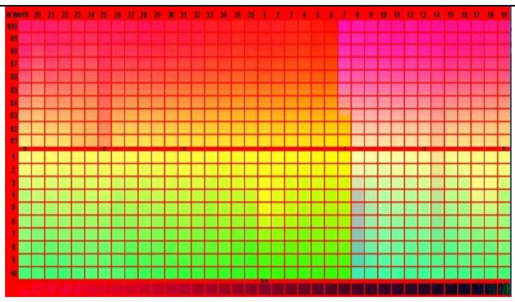
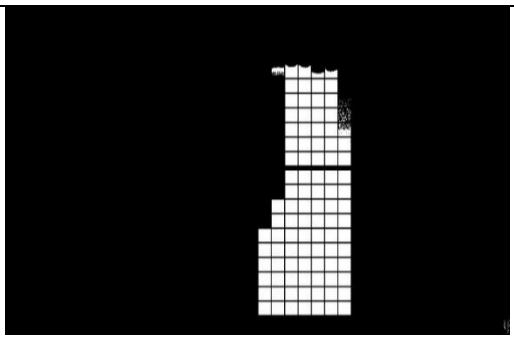
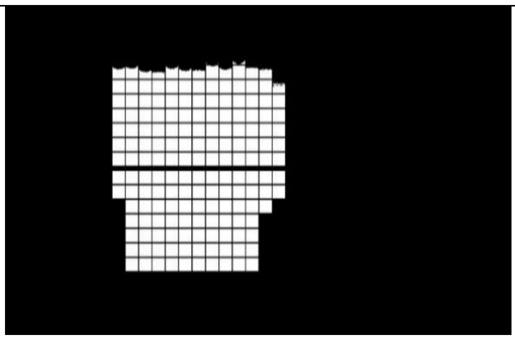
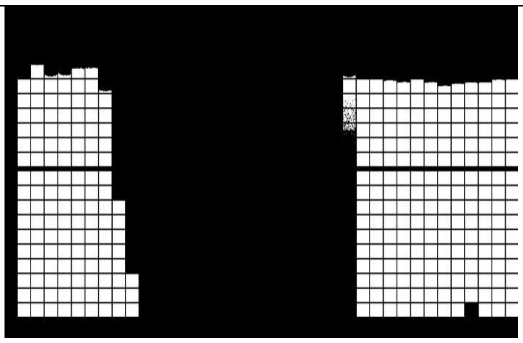
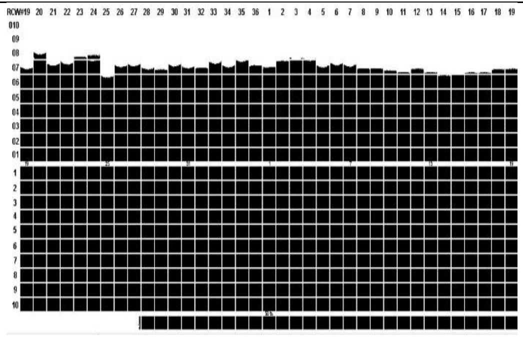
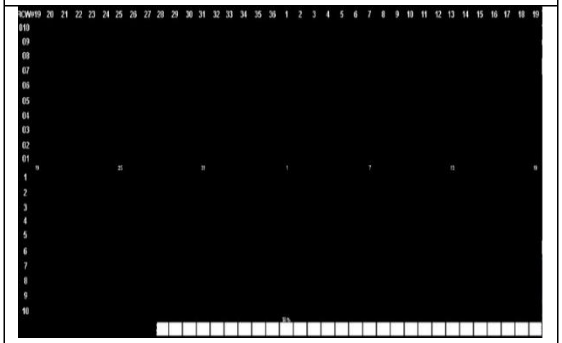
Figure 3.8: Image before and after Histogram Equalization

3.4.5 Colour-Based Segmentation

Colour-based segmentation is a segmentation technique that partition an image into different regions such that each region shares homogeneous colours correspond to useful information for the objects in the image. In other words, each region defines a class of pixels that has similar colour properties. In this project, colour segmentation is conducted using HSV (Hue, Saturation, and Value) colour model, because HSV is able to separate colour from different intensity which is robust to remove the shaded region and adapt to different illumination, thus, it is able to improve the accuracy and effectiveness in the detection process.

In Table 3.3 below, it shows an RGB image that is converted to the HSV colour model. After this, different range of hue, saturation and, value are adjusted to create multiple masks for each RGB, white and black colour. The output HSV colour segmentation will be a binary image which HIGH or 1 represent the pixels within the threshold while the remaining pixels which are not within the threshold will be set to 0 or LOW.

Table 3.3 : Colour Models and Masks

RGB colour image	HSV colour image
	
Red Mask	Green Mask
	
Blue Mask	White Mask
	
Black Mask	
	

3.4.6 Morphological Operations

After segmentation, the images may be highly distorted due to amplified noises. Therefore, morphological operation are required to remove all of these noises that will affect the extracted shape and texture of the images. The morphological operation requires two inputs, one is the binary images while the other is the kernel which decides the nature of the operation. In this part, morphological opening with 3x3 kernel is applied first on the filtered binary image to remove the amplified noises outside the segmented object which is as shown in Figure 3.9 below.

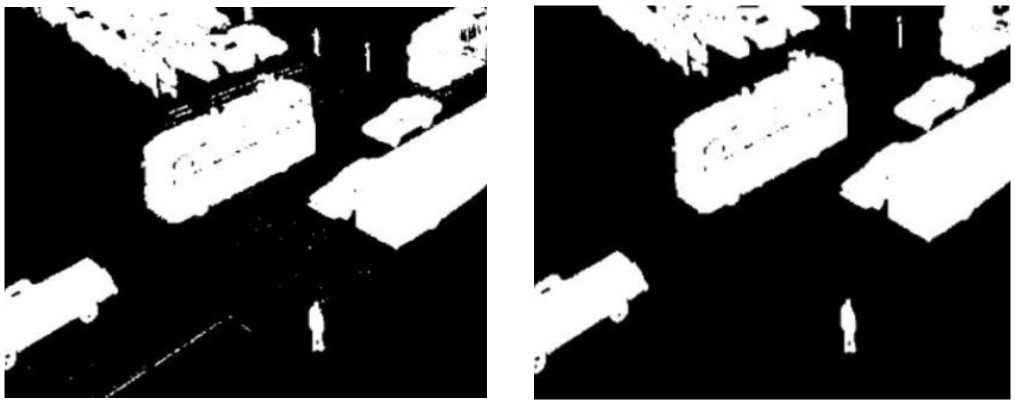


Figure 3.9: The Binary Image before and after Morphological Opening Operation

After removing the outer noises, erosion function with 4x4 kernel is used to erode the boundaries of the foreground object to separate the clustering vehicles region into two separate objects for better vehicle number estimation. From Figure 3.10 below, it shows the images before and after the morphological erosion operation.

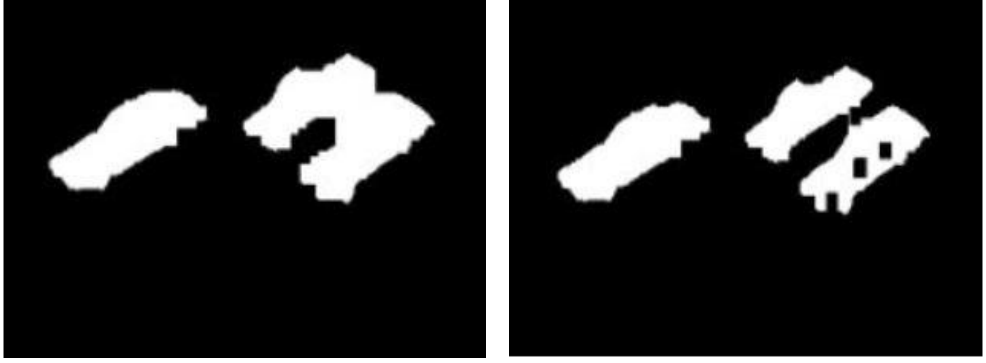


Figure 3.10: Binary image before and after Morphological Erosion Operation

Furthermore, morphological closing function with the closing kernel of 4x4 is further applied to join or close the small holes inside the foreground objects and at the same time minimize two foreground objects touching to each other. The figure below shows the before and after morphological closing is being applied.

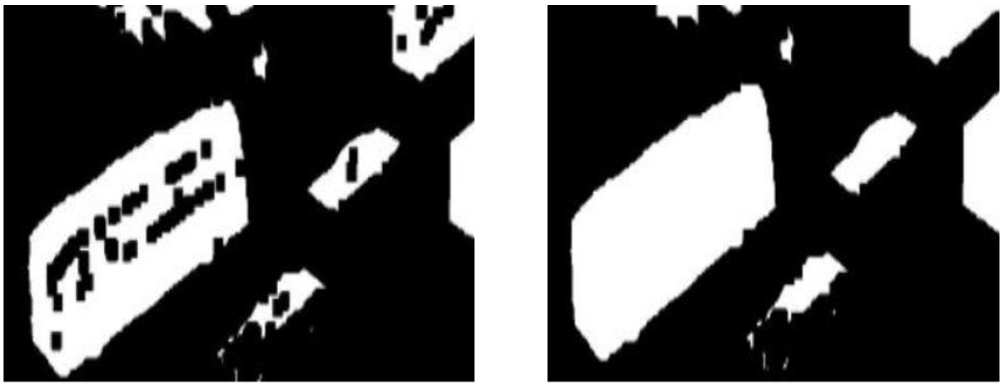


Figure 3.11: The Binary Image before and after Morphological Closing Operation

3.4.7 Vehicle Detection and Counting

After a series of image processing processes, the moving foreground objects representing the moving vehicles can be distinguished and detected easily in the binary image. The centroid for each of detected foreground object is calculated as shown in Figure 3.12 below. The system will estimate the number of vehicles according to the number of centroids detected.

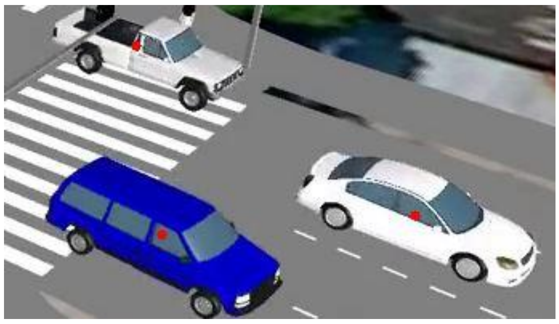


Figure 3.12 : Centroid for Detected Vehicles

3.5 UART- Serial communication

3.5.1 Serial Communication on Altera DE2 board

The implementation of UART serial communication in the Altera DE2 board is divided into three sub-modules which are the receiver module, transmitter module, and baud rate generator which is clearly shown in Figure 3.13 below. The implementation of the UART communication module is the realization of the three sub-modules. Firstly, the baud rate generator is used to produce a local clock signal which is much higher than the baud rate to synchronize the UART reception and transmission. Secondly, the receiver module is used to receive the serial data at RXD and convert them into parallel data. Lastly, the transmit module is used to converts the bytes into serial bits according to the basic frame format and transmits those bits through TXD.

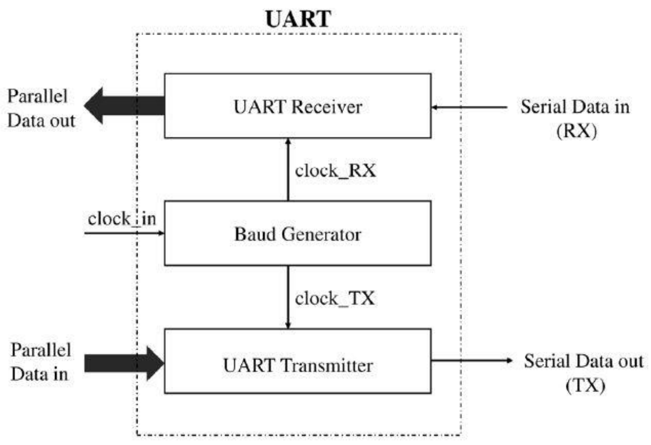


Figure 3.13 : UART Module (Mitu. R, 2017)

3.5.1.1 Baud Rate Generator Module

Baud rate refers to the time taken for bytes of data transmitted in a serial line. Baud rate is expressed in the unit of bits per second (bps). Thus, in the context of serial port, 115200 baud indicates that the serial port is capable to transferring maximum of 115200 bps and this indicates the higher the baud rate, the more data can be transferred in a short time. Since the baud rate is associated with the speed of data transmission, therefore it is important to have an equal baud rate between transmitter and receiver.

In this part, to implement a baud rate generator in the Altera DE2 board, the clock cycle per bit is calculated by using system clock frequency and the baud rate. In the Altera DE2 board, there are two build-in oscillators with a clock frequency of 27MHz and 50MHz respectively can be used. However, in this project, an oscillator with 50MHz clock frequency and baud rate of 115200 bps will be used to implement a baud rate generator because high data transmission speed is required.

To calculate the clock cycle per bit, equation below is used

$$\text{Clock cycle per bit} = \frac{\text{System clock frequency (Hz)}}{\text{Baud rate (bps)}} \qquad (3.1)$$

$$= \frac{50\ MHz}{115200\ bps} = 434 \text{ cycle per bit}$$

From the calculation above, when the clock frequency is 50MHz with the baud rate of 115200 bps, 1 bit of binary data will consist around 434 clock cycles. This calculated value is used by the receiver module to determine the middle point of each serial data bit to sample the received data information, and for the transmitter to transmit 1 bit of data based on the calculated cycle per bit.

3.5.1.2 Receiver Module

As mentioned before, the receiver is basically used to receive the serial data from transmitter and then convert it into 8-bit parallel data. During the UART reception, it is critical to correctly define the start-bit from a frame of serial data since the receiving clock are asynchronous with the transmitting clock. The receiver module will receive serial data from receiving pin (RXD) and if the state of received data transits from logic 1 to logic 0, it can be regarded as the beginning of a data frame (start-bit). When the UART receiver module is idle, it has been waiting for RXD logic state to transit. In the receiver module, after the start-bit has been identified, the receiver will sample the logic state at the middle of each bit serial data by using half of the clock cycle per bit value that is calculated in the baud rate generator module. Each sampled value of the data state is then deposited in the 8-bit register by order. When the count equals to 10 bit (start-bit + 8 bit data + stop-bit), and verified, received serial data are converted into a byte of parallel data and stored. In this project, the UART receiver module is implemented by using the finite state machine.

3.5.1.3 Serial Receive FSM

From Figure 3.14 below, the finite state machine for receiver module consists of five states: IDLE (waiting for start bit), Start (find and check whether the midpoint of start bit is logic 0), Data (sampling the serial data), Stop (waiting and receiving stop bit) and Cleanup (reset some variable).

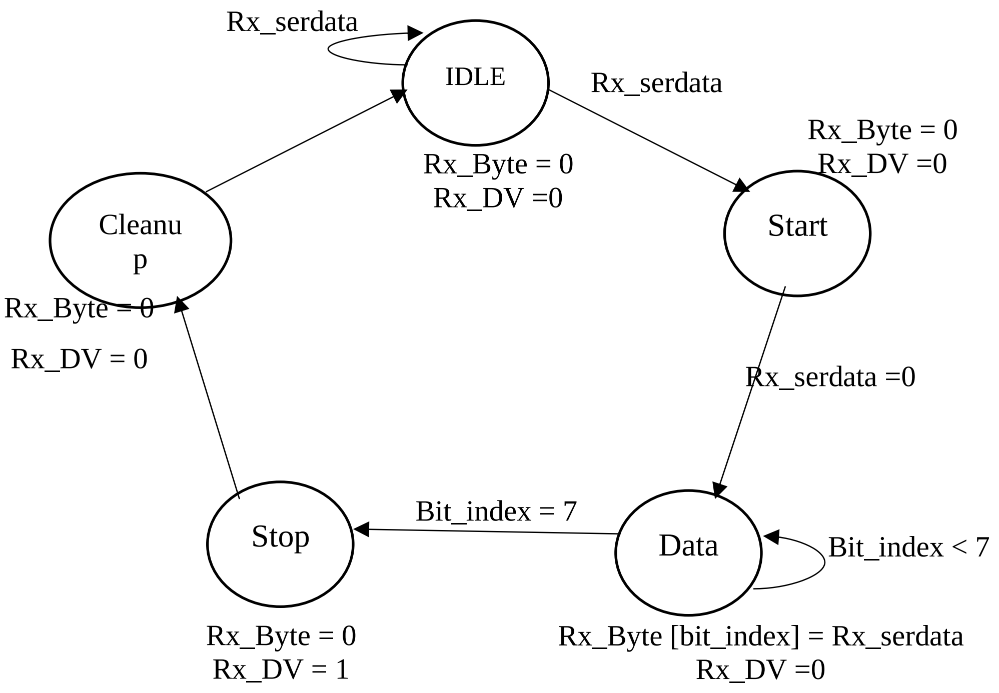


Figure 3.14 : Receiver Module FSM

3.5.1.4 Transmitter Module

The transmitter module serialize the 8-bit parallel binary data and adds a start-bit at the beginning of the data as well as a stop-bit at the end of the data. When the UART transmitter module is enable in IDLE state, the transmitter module will jump to Start state and send a logic 0 to the receiver to inform data is ready to be sent. After this, 8 bit parallel data is read in the 8-bit register and sends it out bit by bit via the TXD line. The data is sent following the order 1 start-bit, 8 data-bits, and 1 stop-bit. In this project, the UART transmitter module is also implemented using the finite state machine.

3.5.1.5 Serial Transmit FSM

From Figure 3.15 below, the finite state machine for transmitter module consists of five states: IDLE (waiting for enable signal), Start (send start bit), Data (transmit 8 bit serial data), Stop (send stop bit) and Cleanup (reset some variable).

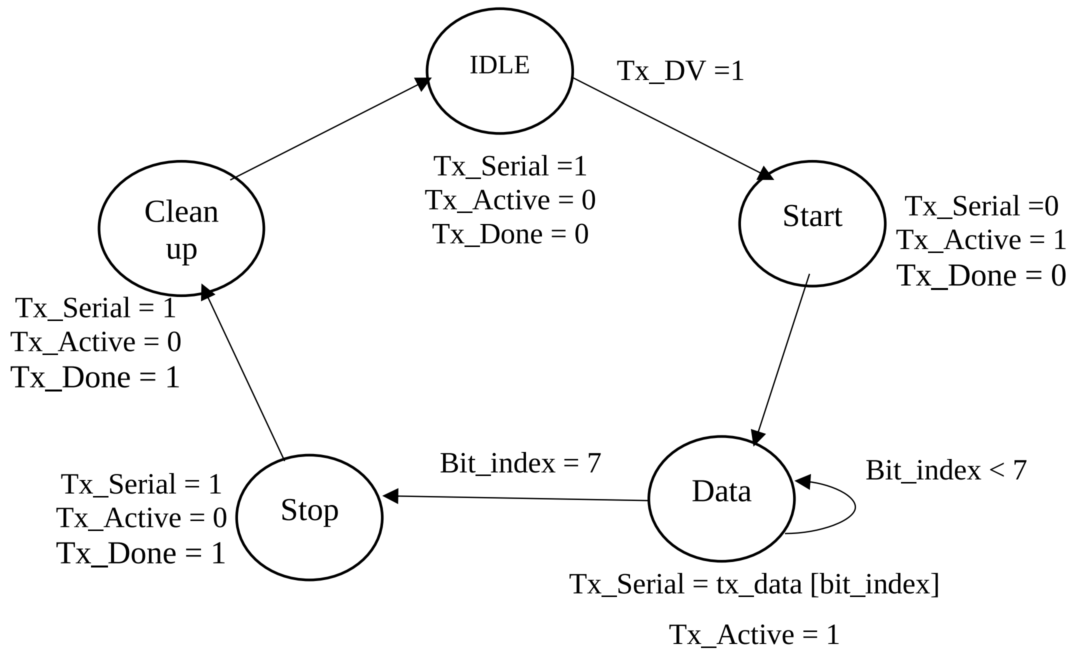
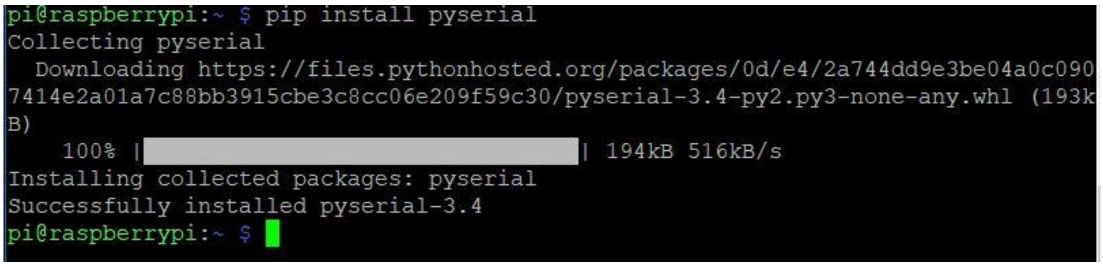


Figure 3.15 : Transmitter Module FSM

3.5.2 Serial Communication on Raspberry Pi 3B+

3.5.2.1 Install PySerial Package

The command “pip install pyserial” was ran in the terminal to download and install python serial communication package to enable the serial communication function in Raspberry Pi.

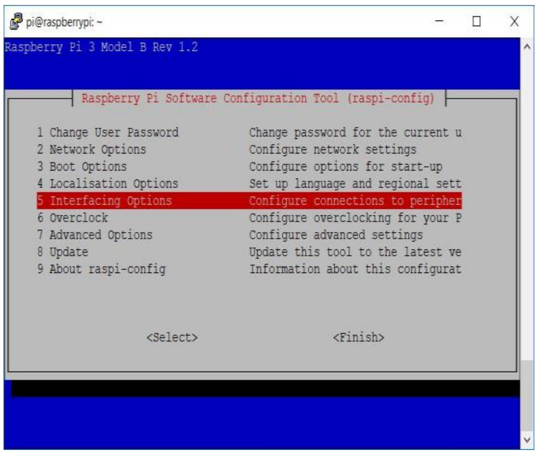


```
pi@raspberrypi:~ $ pip install pyserial
Collecting pyserial
  Downloading https://files.pythonhosted.org/packages/0d/e4/2a744dd9e3be04a0c090
7414e2a01a7c88bb3915cbe3c8cc06e209f59c30/pyserial-3.4-py2.py3-none-any.whl (193k
B)
    100% |████████████████████████████████| 194kB 516kB/s
Installing collected packages: pyserial
Successfully installed pyserial-3.4
pi@raspberrypi:~ $ 
```

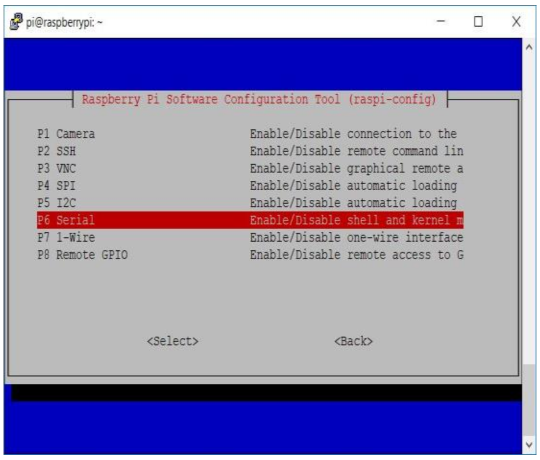
Figure 3.16 : Install Python Serial Package

3.5.2.2 Disable Unused Serial Ports

The configuration command “sudo rasp-config” was ran and the instructions as shown in Figures below were followed to prohibit other serial ports in order to optimize the Raspberry Pi performance by freeing up some resources. After that, the system was rebooted for the configuration to take effect.

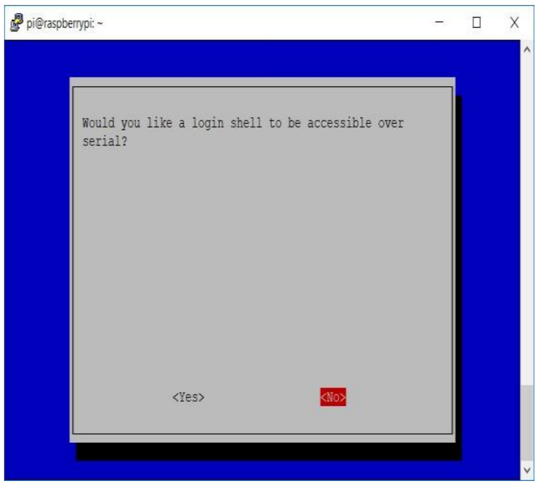


```
pi@raspberrypi: ~
Raspberry Pi 3 Model B Rev 1.2

      Raspberry Pi Software Configuration Tool (raspi-config)

 1 Change User Password          Change password for the current u
 2 Network Options               Configure network settings
 3 Boot Options                  Configure options for start-up
 4 Localisation Options          Set up language and regional sett
 5 Interfacing Options           Configure connections to peripher
 6 Overclock                     Configure overclocking for your P
 7 Advanced Options              Configure advanced settings
 8 Update                        Update this tool to the latest ve
 9 About raspi-config            Information about this configurat

              <Select>                    <Finish>
```



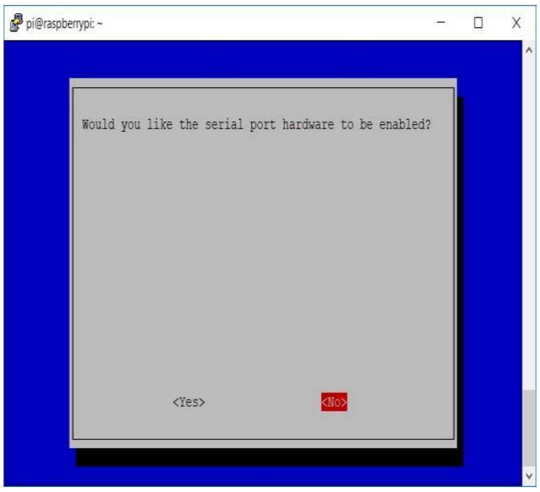
```
pi@raspberrypi: ~

      Raspberry Pi Software Configuration Tool (raspi-config)

 P1 Camera                       Enable/Disable connection to the
 P2 SSH                          Enable/Disable remote command lin
 P3 VNC                          Enable/Disable graphical remote a
 P4 SPI                          Enable/Disable automatic loading
 P5 I2C                          Enable/Disable automatic loading
 P6 Serial                       Enable/Disable shell and kernel m
 P7 1-Wire                       Enable/Disable one-wire interface
 P8 Remote GPIO                  Enable/Disable remote access to G

              <Select>                    <Back>
```

Figure 3.17 : Select Interfacing Options. Figure 3.18 : Choose P6 Serial.



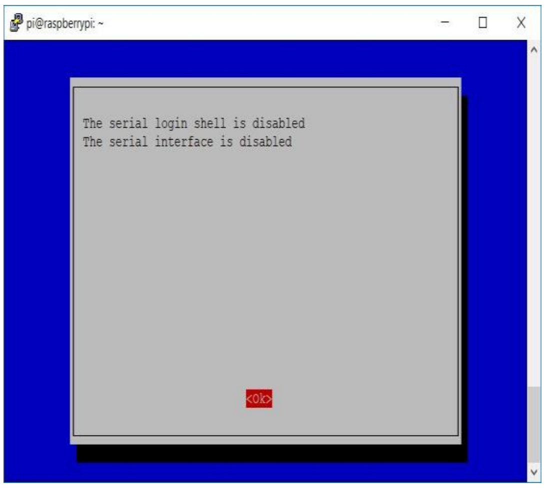
```
Would you like a login shell to be accessible over
serial?

<Yes>                    <No>
```

Figure 3.19 : Disable the Login Shell to Accessible Over Serial.



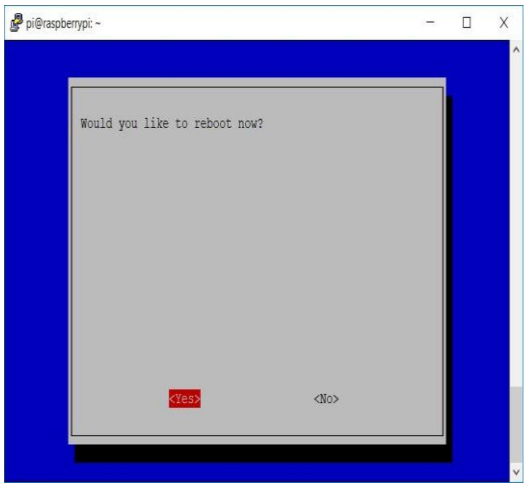
```
Would you like the serial port hardware to be enabled?

<Yes>                    <No>
```

Figure 3.20 : Disable the Serial Port Hardware.



```
The serial login shell is disabled
The serial interface is disabled

<Ok>
```

Figure 3.21 : Confirm the Configurations.

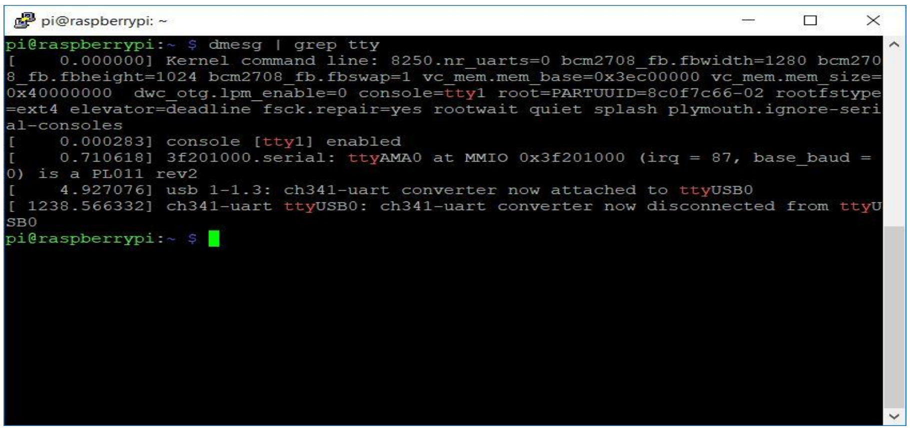


```
Would you like to reboot now?

<Yes>                    <No>
```

Figure 3.22 : Reboot the System.

3.5.2.3 Check ttyUSB port list appearance from the list and ready to function

After the configurations, “ dmesg | grep tty” command is used to search the related configuration files with the term “tty” and display the messages regarding to the status of configuration. From the Figure 3.23 below, it shows that ttyUSB0 port is enabled and ready to be used.



```
pi@raspberrypi:~ $ dmesg | grep tty
[    0.000000] Kernel command line: 8250.nr_uarts=0 bcm2708_fb.fbwidth=1280 bcm270
8_fb.fbheight=1024 bcm2708_fb.fbswap=1 vc_mem.mem_base=0x3ec00000 vc_mem.mem_size=
0x40000000  dwc_otg.lpm_enable=0 console=tty1 root=PARTUUID=8c0f7c66-02 rootfstype
=ext4 elevator=deadline fsck.repair=yes rootwait quiet splash plymouth.ignore-seri
al-consoles
[    0.000283] console [tty1] enabled
[    0.710618] 3f201000.serial: ttyAMA0 at MMIO 0x3f201000 (irq = 87, base_baud =
0) is a PL011 rev2
[    4.927076] usb 1-1.3: ch341-uart converter now attached to ttyUSB0
[ 1238.566332] ch341-uart ttyUSB0: ch341-uart converter now disconnected from ttyU
SB0
pi@raspberrypi:~ $
```

Figure 3.23 : Verify the ttyUSB0

3.5.3 UART- Serial Communication Connection

Figure 3.24 below shows the connections of serial communication for the Raspberry Pi 3B+ and Altera DE2 FPGA. From the figure, USB to serial RS232 converter cable is used to provide connectivity between USB port (Raspberry Pi 3B+) and RS232 serial port (Altera DE2 FPGA). Within the USB to RS232 converter cable, it consists of ch340 chip which function to equalize the voltage level for serial communication between USB and RS232 port, and at the same time interconnect the receiver terminal to transmitter terminal so that duplex communication can be made.

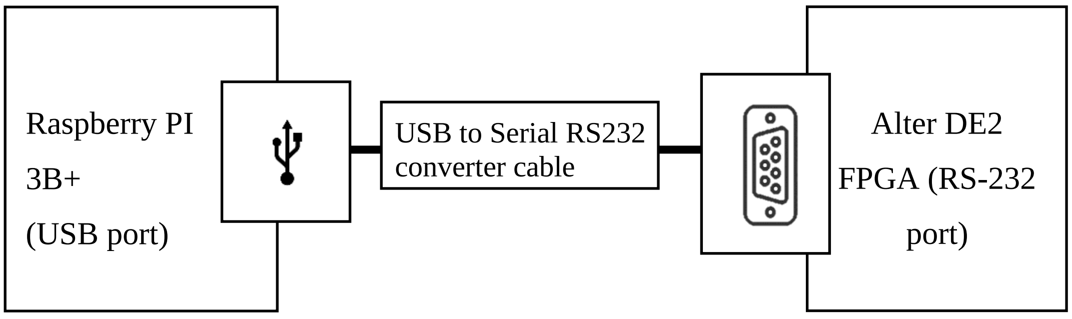


Figure 3.24 : UART Connections

3.6 Traffic light Controller System on Altera DE2 Board

3.6.1 Structure of Traffic Junction

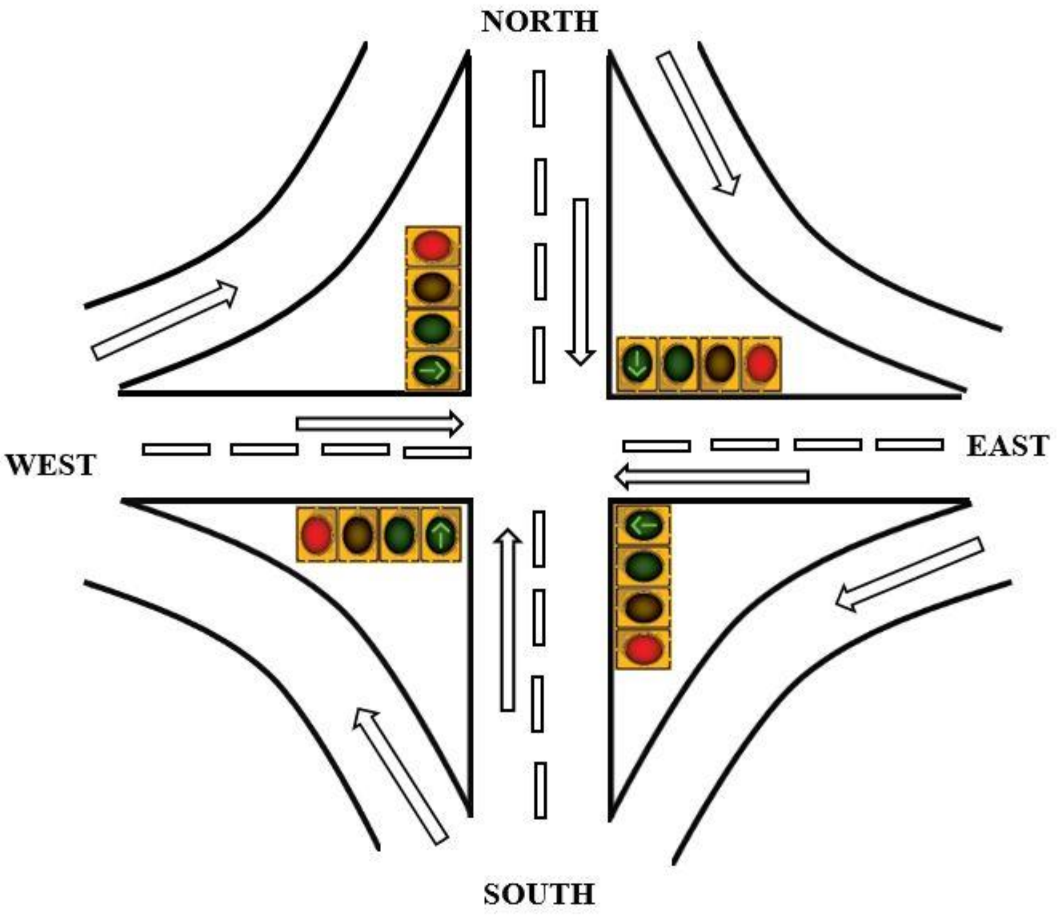


Figure 3.25 : Model of the Traffic Junction

From the Figure 3.25 above, it depicts a general structure of traffic junction which consists of North, East, and West and South directions. Four traffic lights with a set of four lights: two green, yellow and red light are used to control the traffic flow or discharge the vehicles from each direction of the lane. From the model of the traffic junction above, vehicles are free to move from West to North, South to West, East to South, and North to East directions. However, for the vehicles that need to move in straight or right direction, drivers are required to obey the traffic signals with pre-defined sequences or phases.

To design a traffic light controller, traffic phase design is one of the important steps to separate the conflict of vehicle movements in an intersection into various phases. In the proposed system, four-phase signals are used in the implementation of the TLC and the movement of vehicles across the traffic intersection in different phases are shown in the figures below.

3.6.2 4 Phase Traffic Light Signals

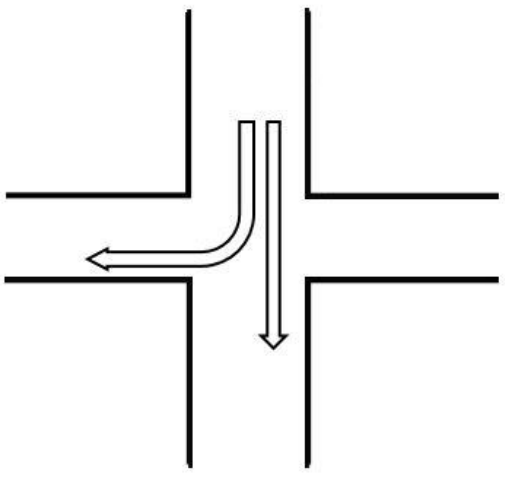


Figure 3.26 : Phase 1 Traffic Signal

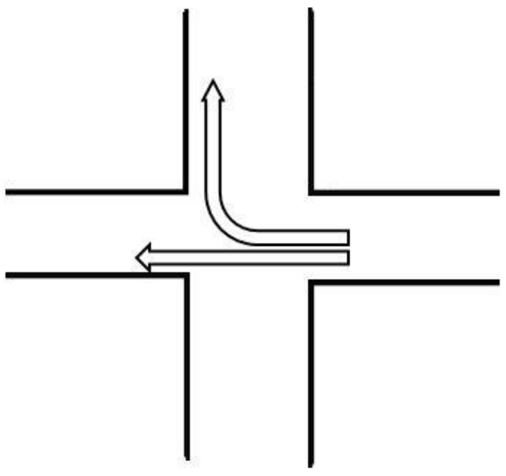


Figure 3.27 : Phase 2 Traffic Signal

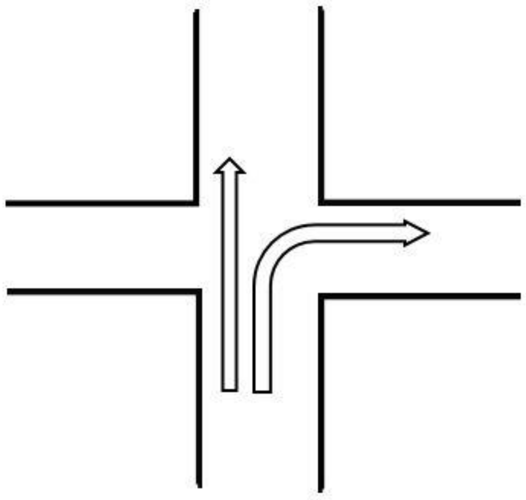


Figure 3.28 : Phase 3 Traffic Signal

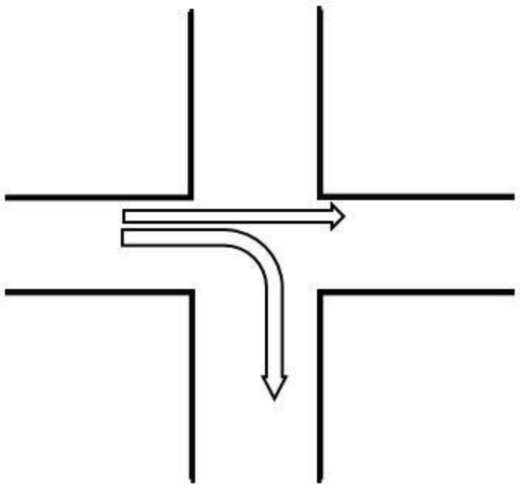


Figure 3.29 : Phase 4 Traffic Signal

3.6.3 State table of the Traffic Light Controller

Table 3.4 : State Table of the Traffic Light Controller System

States				**Direction of lanes**				**Output**
Current State	**Stored State**	**Next State**		**North**	**East**	**South**	**West**	**S_light , W_light, N_light, E_light**
		clr=0	**clr=1**					
Idle (D)	**A**	**I**	**Idle**	Red	Yellow	Red	Red	0001 0001 0001 0010
A	**C**	**B**	**Idle**	**Green**	Red	Red	Red	1100 0001 0001 0001
B		**I**	**Idle**	Yellow	Red	Red	Red	0010 0001 0001 0001
C	**E**	**D**	**Idle**	Red	**Green**	Red	Red	0001 1100 0001 0001
D		**I**	**Idle**	Red	Yellow	Red	Red	0001 0010 0001 0001
E	**G**	**F**	**Idle**	Red	Red	**Green**	Red	0001 0001 1100 0001
F		**I**	**Idle**	Red	Red	Yellow	Red	0001 0001 0010 0001
G	**A**	**H**	**Idle**	Red	Red	Red	**Green**	0001 0001 0001 1100
H		**I**	**Idle**	Red	Red	Red	Yellow	0001 0001 0001 0010
I		**Stored State**	**Idle**	Red	Red	Red	Red	0001 0001 0001 0001

3.6.4 Traffic Light Controller Flowchart

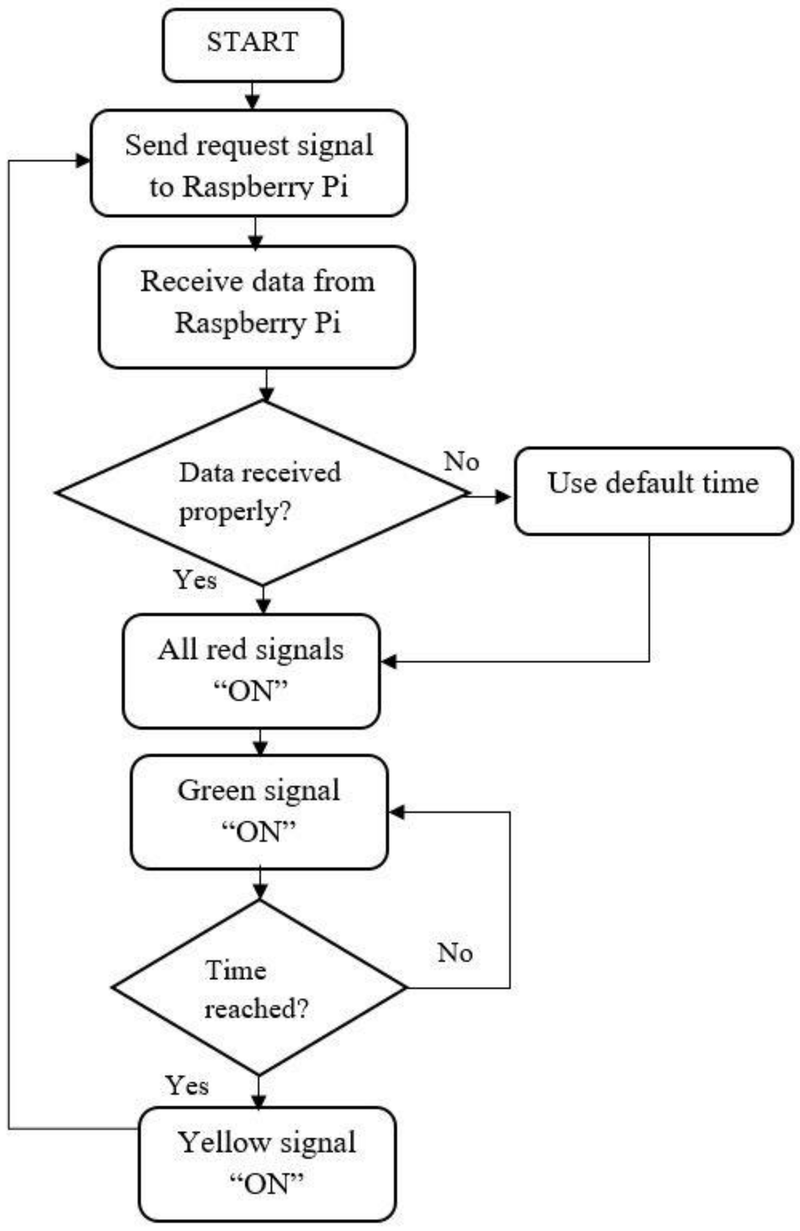


Figure 3.30 : Flowchart of the Traffic Light Controller

Figure 3.30 above illustrates the flowchart of the implemented TLC. At first, the traffic light controller will send a request signal and receive the green time value calculated by the Raspberry Pi. If the data is not received correctly due to connection problem or received value is not within the range, the traffic light controller system will use the default green light time for traffic control. Otherwise, the traffic light will display the green light according to the received green time value from the Raspberry Pi, the system will always compare the received data with counter value to limit the green time period, if the green light period has reached, green light will be turned OFF and yellow light will be turned ON, the system will send a signal to retrieve the green time value for the next phase of traffic signal. This process is repeated continuously.

3.6.5 State Diagram of Traffic Light Controller System

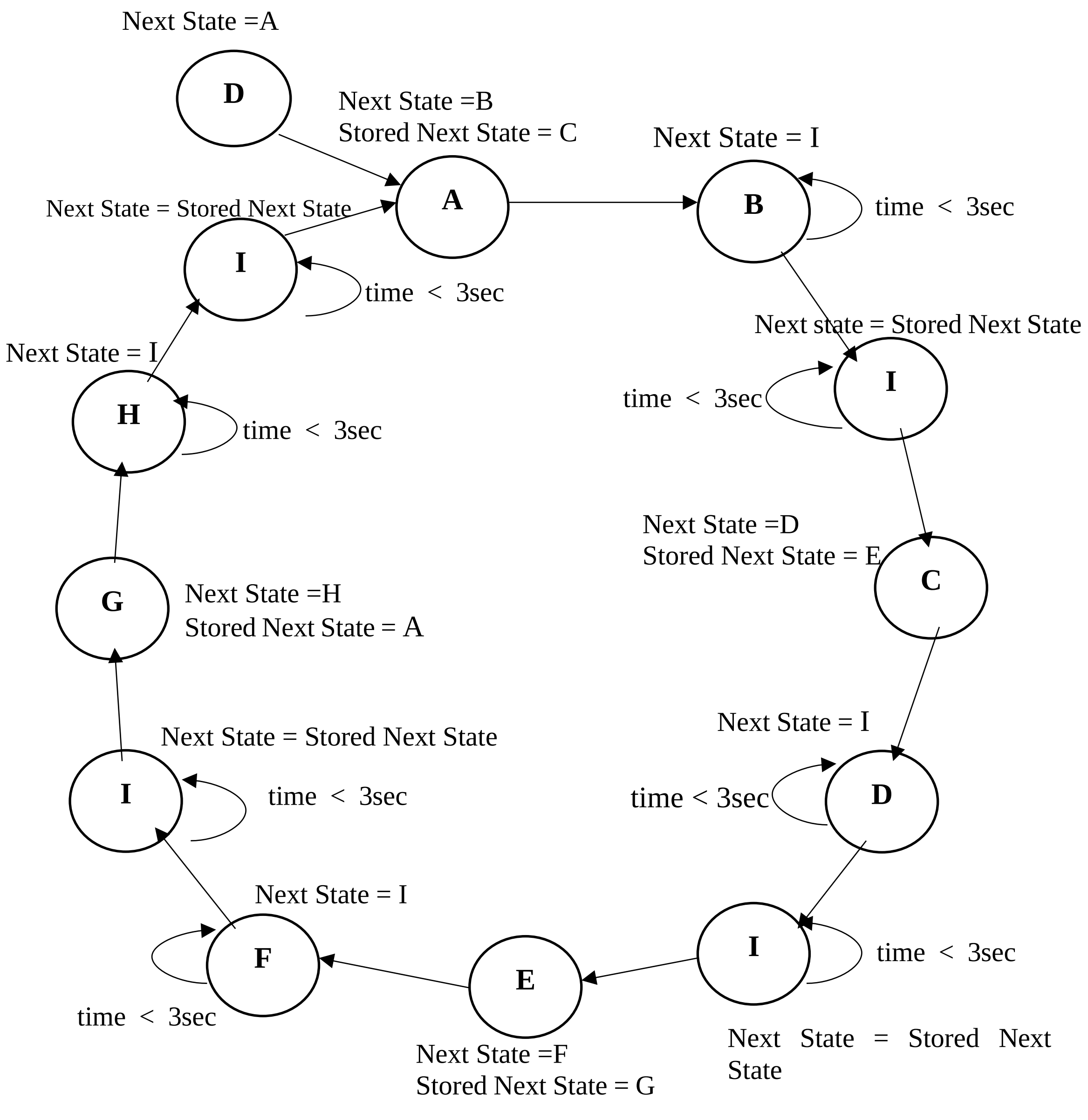


Figure 3.31 : State Diagram of the Traffic Light Controller

3.6.6 Hardware Implementation of Traffic Light Controller

Figure 3.32 below shows the TLC circuit is simulated using the build-in general-purpose input output (GPIO) pins on the Altera DE2 board. From the circuit model, one traffic light circuit consists of 4 resistors, 2 green LEDs, 1 yellow LED, and 1 red LED. Thus, a total of 16 LEDs (4 red LEDs, 4 yellow LEDs, and 8 green LEDs) and resistors are required for all directions of the traffic light circuit.

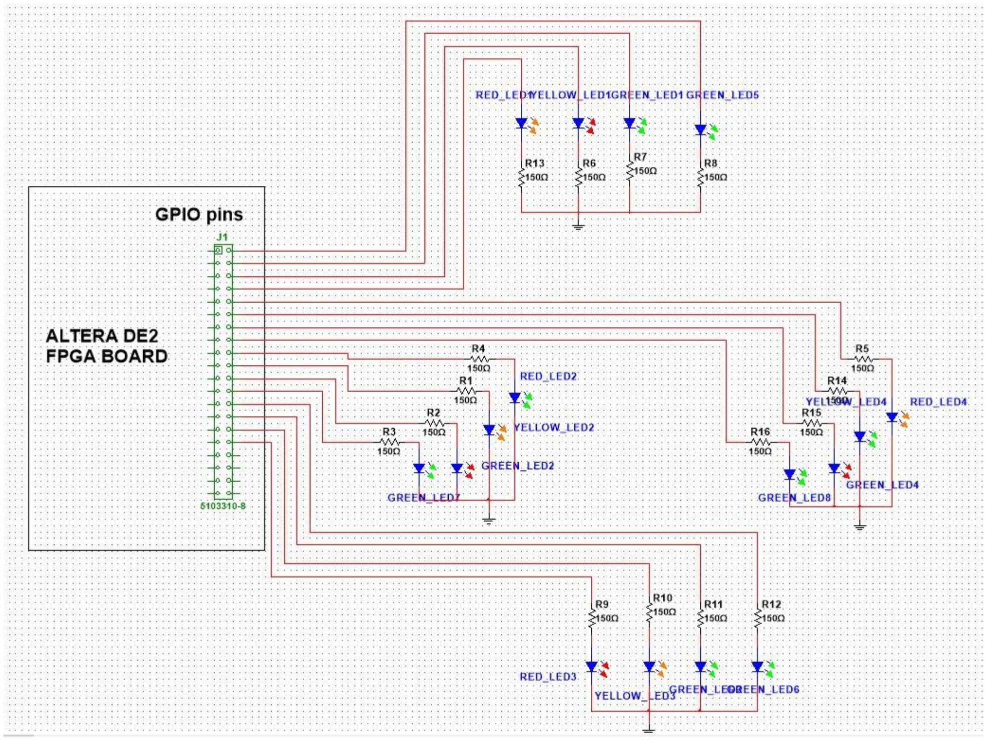


Figure 3.32 : Schematic of the Traffic Light Controller

The precaution needs to be considered during the implementation of the TLC circuit is the maximum current rating for LEDs. In the traffic light circuit, each LED must have a resistor that is connected in series connection in order to prevent the overflow of the current flow through the LED, otherwise the LED will burn out instantly. In this project, 5mm LEDs with a maximum current rating is about 20mA. Therefore, to ensure the reliability of the circuit, current flow with less than 10mA is required. To obtain this current value, Ohm’s Law equation as shown in the equation below is used to calculate the minimum required resistor value.

$$R = \frac{Vs - Vf}{I} \qquad (3.2)$$

$$R = \frac{Vs - Vf}{I} = \frac{3.3V - 1.8V}{10mA}$$

$$R = 150\ \Omega$$

Where,

Vs is voltage supply by GPIO output pin, maximum voltage is 3.3V.

Vf is forward voltage of LED, typical value is 1.8V.

I is current value, assume is 10mA.

R is resistance value in unit ohms, Ω

Based on the calculation above, a simple traffic light circuit consists of four 150 Ω resistors and 4 LEDs which is as shown in the PCB design. From the traffic light PCB circuit below, the circuit consists of 6 connectors, 4 connectors were used for each LED making the connection to the Altera DE2 board through the 150 Ω resistor. Another one connector is the ground terminal of the LEDs.

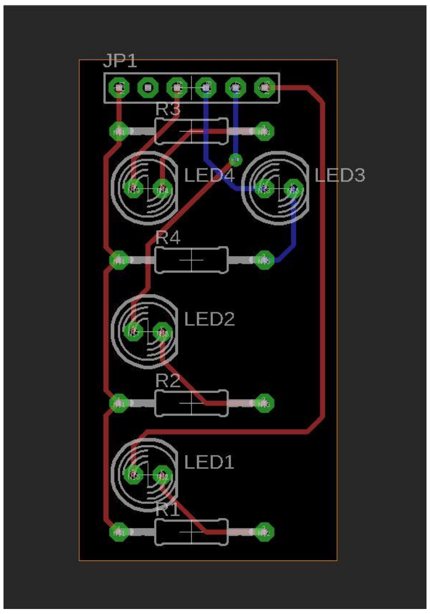


Figure 3.33 : PCB Design of the Traffic Light Circuit

3.7 Timing Algorithms for Traffic Light Signal

In order to make the system more comprehensive, the proposed STLCS comes with a timing algorithm that aims to minimize the queuing time at each intersecting road and make the system more effective in discharging the traffic load.

The timing algorithm is implemented based on Figure 3.34 below. From the figure, the time needed for a vehicle to pass through the junction is referred to as headway. Theoretically, the first headway will be relatively greater than the remaining headway because it includes longer start-up loss time, e_n (driver's reaction time + vehicle acceleration time). The second headway is comparatively shorter because the start-up loss time may partially overlap with the first driver and continuously decline for the remaining headway. After a few vehicles, the headway tends to be more saturated or stable at a constant value which means that the vehicles queuing behind have more time to accelerate to high velocity, and eventually, it achieved stable movement of vehicles across the junction. In real life the average saturation headway, h is around 2.4 s/vehicles, however, to shorten the waiting time during the demonstration, saturation headway, h is assumed to be half of the actual value which is at 1.2 s /vehicle.

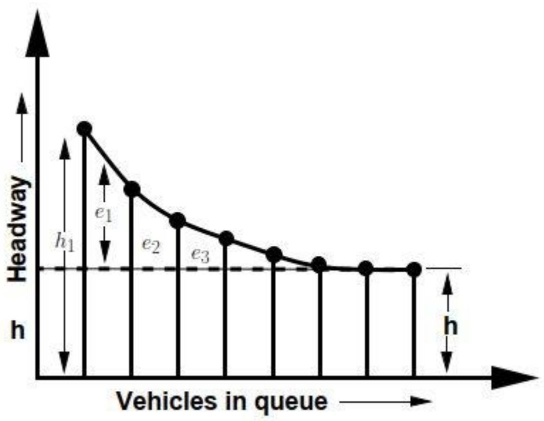


Figure 3.34 : Headways Departing Trend

To make the algorithm more reliable and practical, the total start-up loss time, S_{loss} must be added to the algorithms. Since the start-up loss time, e_n is shown in a negative inverse trend, therefore, the total start-up loss time, S_{loss} can be calculated by using equation,

$$S_{loss} = \sum_{i=1}^{n} sloss[(f^{i-1})] \qquad (3.3)$$

Where,

S_{loss} is total start-up loss time.

sloss is standard start-up loss time (assume 2.5s).

f is decrease factor (assume 0.75).

n is number of detected vehicles.

Furthermore, the green time required to clear N vehicles can be estimated as

$$T = S_{loss} + h*N \qquad (3.4)$$

Where,

T is required green time.

S_{loss} is total start-up loss time

N is number of detected vehicles

h is saturation headway. (assume 1.2s / vehicle)

Lastly, though the system will calculate and adjust the green time depending upon the traffic load, but it still provides a minimum and maximum time limit which are fixed to 5 seconds and 25 seconds respectively, and this is considered as a precautionary measure for vehicle detection error.

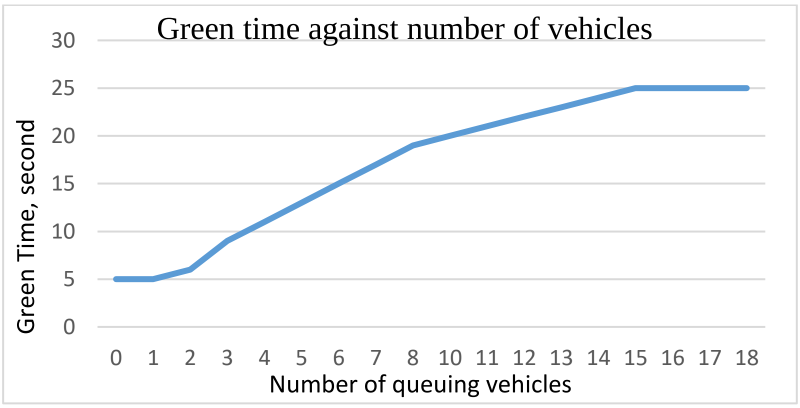


Figure 3.35 : Maximum and Minimum Green Light Period

3.8 Cost Analysis of the Project.

The total expenditure done in favour of the project right from its initiation up to its completion has been tabulated in the Table 3.5 below.

Table 3.5 : Equipment and Components List with Price

No.	**Components**	**Quantity**	**Remarks**	**Price (RM)**
1.	Altera DE2 board (2C35)	1	Acquired from Lab	-
2.	Raspberry Pi 3B+	1	Acquired from Lab	-
3.	OV5647 camera Module - 175 ° view angle	1	Website : Shopee.com	RM49.67
4.	Female to Female jumper wires (40p)	1	Website : Lelong.com	RM 5.80
5.	RS232 to USB serial cable	1	Website : Shopee.com	RM10.10
6.	LEDs (Red)	4	Acquired from Lab	-
7.	LEDs (Yellow)	4	Acquired from Lab	-
8.	LEDs (Green)	8	Acquired from Lab	-
9.	Polystyrene -12x1 inches	6	Website : Shopee.com	RM14.40
10.	Polystyrene -10x1 inches	5	Website : Shopee.com	RM10.00
11.	Toy Cars	1	Giant Kampar	RM 9.91
			Total Cost :	**RM 99.88**

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Software Testing

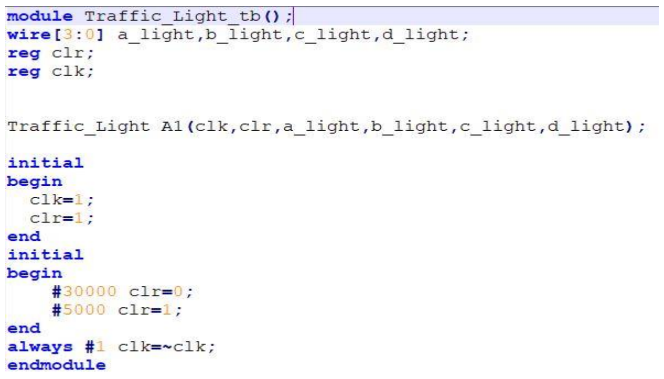
The software testing process involves the utilization of simulation software to test and check the behaviour and functionality of the implemented software and to determine whether the program is able to generate the required outcomes when different inputs or conditions were applied.

4.1.1 Traffic Light Controller Program Simulation

In this project, the traffic light controller program was written in Verilog HDL using the ModelSim 10.1b simulator software. After implementation, a simple test-bench as shown in Figure 4.1 was loaded to simulate the functionality of the FPGA based traffic light controller program. The simulated output waveforms of the TLC are shown in Figure 4.2 and 4.3 below, and the description for related output is shown in Table 4.1.

Table 4.1 : Description for the Simulated Traffic Light Output

Simulated output	**Description**
0001	Red light turn ON
0010	Yellow light turn ON
1100	Two green lights turn ON



```
module Traffic_Light_tb();
wire[3:0] a_light,b_light,c_light,d_light;
reg clr;
reg clk;


Traffic_Light A1(clk,clr,a_light,b_light,c_light,d_light);

initial
begin
  clk=1;
  clr=1;
end
initial
begin
    #30000 clr=0;
    #5000 clr=1;
end
always #1 clk=~clk;
endmodule
```

Figure 4.1 : Test-bench for the Traffic Light Controller

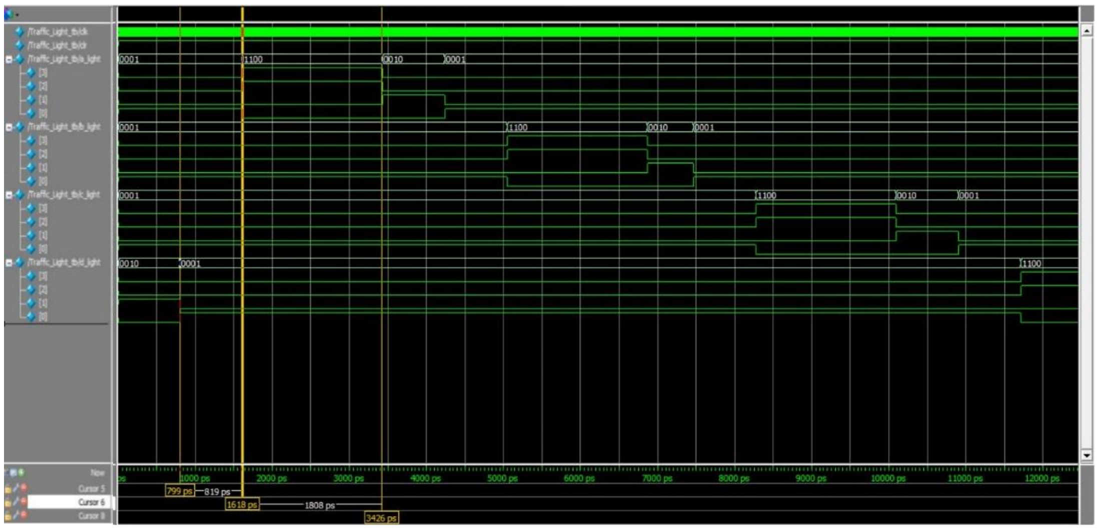


Figure 4.2 : Simulated Output Waveform of the TLC

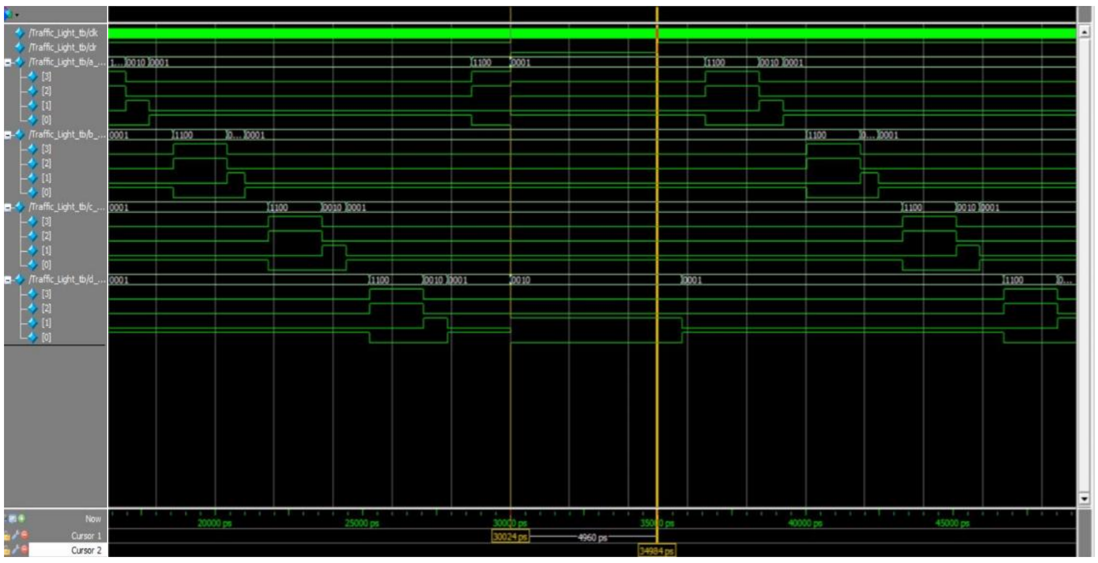
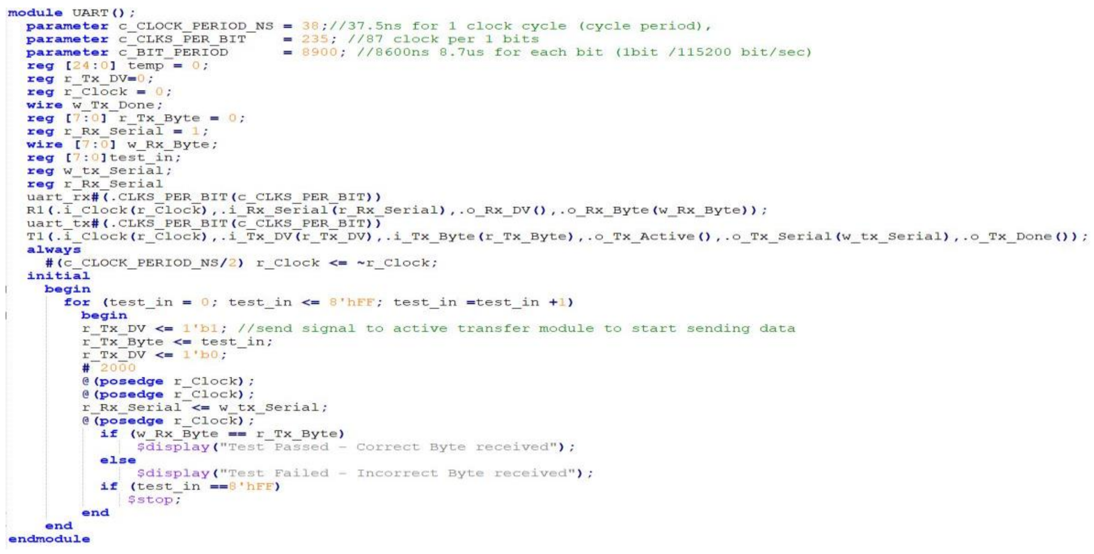


Figure 4.3 : Simulated Output Waveform of the TLC

4.1.2 UART-Serial Communication Simulation

UART serial communication consists of three modules, which are baud rate generator, transmitter, and a receiver module. These modules were simulated and tested using the test-bench as shown in Figure 4.4 below. The output waveforms for each transmitter and receiver module are shown in Figure 4.5, 4.6, 4.7 and 4.8 below.



```
module UART();
  parameter c_CLOCK_PERIOD_NS  = 38;//37.5ns for 1 clock cycle (cycle period),
  parameter c_CLKS_PER_BIT     = 235; //87 clock per 1 bits
  parameter c_BIT_PERIOD       = 8900; //8600ns 8.7us for each bit (1bit /115200 bit/sec)
  reg [24:0] temp = 0;
  reg r_Tx_DV=0;
  reg r_Clock = 0;
  wire w_Tx_Done;
  reg [7:0] r_Tx_Byte = 0;
  reg r_Rx_Serial = 1;
  wire [7:0] w_Rx_Byte;
  reg [7:0]test_in;
  reg w_tx_Serial;
  reg r_Rx_Serial
  uart_rx#(.CLKS_PER_BIT(c_CLKS_PER_BIT))
  R1(.i_Clock(r_Clock),.i_Rx_Serial(r_Rx_Serial),.o_Rx_DV(),.o_Rx_Byte(w_Rx_Byte));
  uart_tx#(.CLKS_PER_BIT(c_CLKS_PER_BIT))
  T1(.i_Clock(r_Clock),.i_Tx_DV(r_Tx_DV),.i_Tx_Byte(r_Tx_Byte),.o_Tx_Active(),.o_Tx_Serial(w_tx_Serial),.o_Tx_Done());
  always
    #(c_CLOCK_PERIOD_NS/2) r_Clock <= ~r_Clock;
  initial
    begin
      for (test_in = 0; test_in <= 8'hFF; test_in =test_in +1)
        begin
        r_Tx_DV <= 1'b1; //send signal to active transfer module to start sending data
        r_Tx_Byte <= test_in;
        r_Tx_DV <= 1'b0;
        # 2000
        @(posedge r_Clock);
        @(posedge r_Clock);
        r_Rx_Serial <= w_tx_Serial;
        @(posedge r_Clock);
          if (w_Rx_Byte == r_Tx_Byte)
              $display("Test Passed - Correct Byte received");
          else
              $display("Test Failed - Incorrect Byte received");
          if (test_in ==8'hFF)
             $stop;
        end
    end
endmodule
```

Figure 4.4 : Test-bench for UART Serial Communication

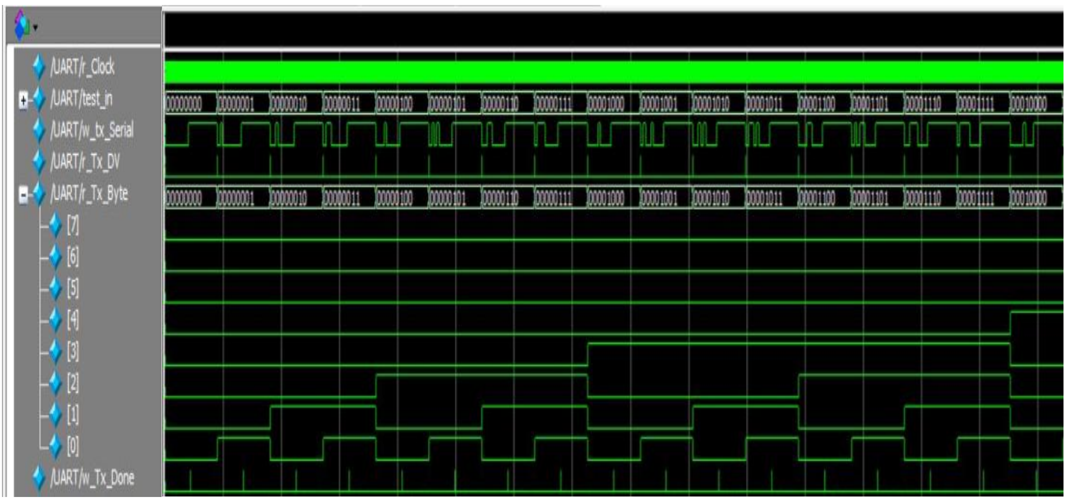


Figure 4.5 : Output Waveform of the Transmitter (0000000 to 0010000)

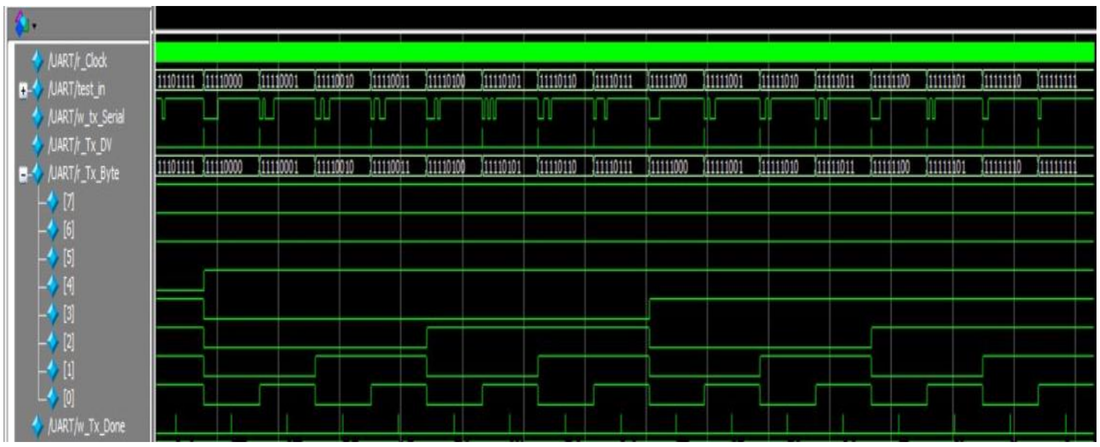


Figure 4.6 : Output Waveform of the Transmitter (11101111 to 11111111)

From Figure 4.5 and 4.6 above, each increment value in variable test_in was assigned into the variable r_Tx_byte in the transmitter module. When w_tx_Serial drops from 1 to 0, the transmitter module serialized and transmitted the value stored in the variable r_Tx_byte to the receiver module.

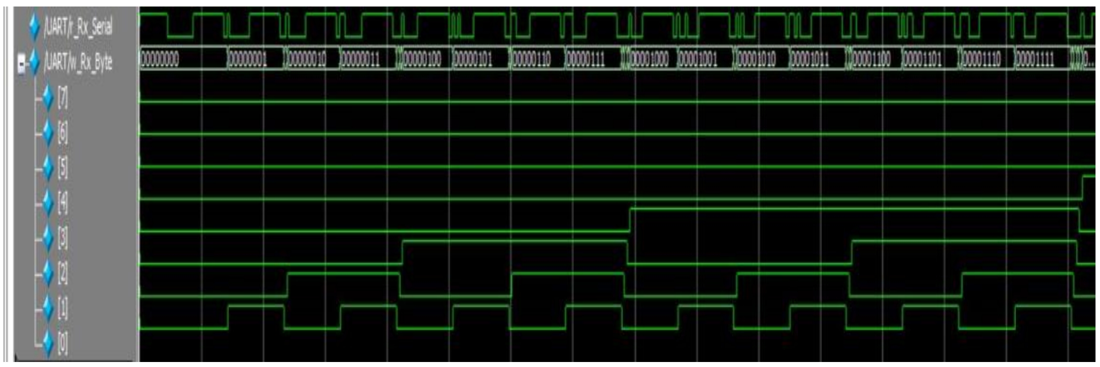


Figure 4.7 : Output Waveform of the Receiver (0000000 to 0010000)

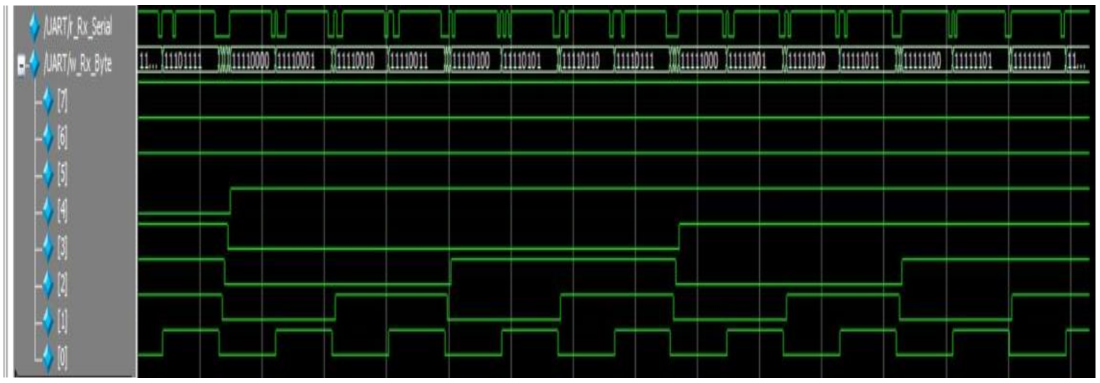


Figure 4.8 : Output Waveform of the Receiver (11101111 to 11111111)

From Figure 4.7 and 4.8 above, each serialized data from the w_tx_Serial was assigned into serial input of receiver module r_Rx_Serial. The receiver module parallelized the received value and stored it in the variable w_Rx_Byte. The binary data stored in w_Rx_Byte is then compared with the binary data in r_Tx_byte, if the data is equal, the binary data in test_in is increasing continuously until it reached 11111111.

4.2 Hardware Testing

During the development process, hardware testing has been done in several phases to ensure the proper operation of the circuit designed as well as hardware device involved.

4.2.1 Basic Circuit Testing

The phase of hardware testing was conducted based on the following tests and evaluations:

i. Short circuit testing between Vcc (3.3V) and ground.

ii. Testing the functionality of Altera DE2 GPIO ports.

iii. Checking the functionality of LEDs and resistors.

iv. Connectivity testing.

v. Testing the data transfer of RS232 to USB converter cable.

4.2.2 Testing the Traffic Light Circuit

The traffic light circuit was tested by connecting the female connector that is connected to the anode of LEDs (traffic signs) via 150Ω resistors to the build-in 3.3V GPIO port on Altera DE2 and connector connected to the cathode of LED to the ground. The circuit was checked whether the certain LEDs were illuminated as required. If the LED was not turned ON, it might be caused by either the short circuit connection, faulty of LED and resistors, or improper connection of components in the circuit.

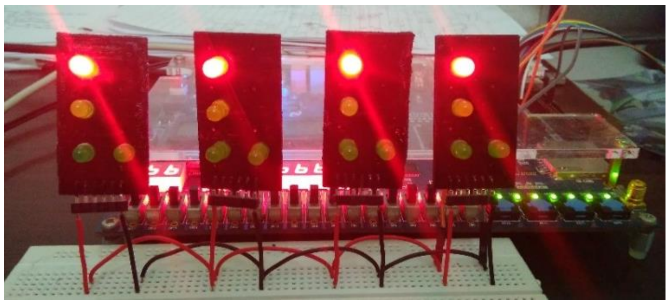


Figure 4.9 : Red light LEDs Testing

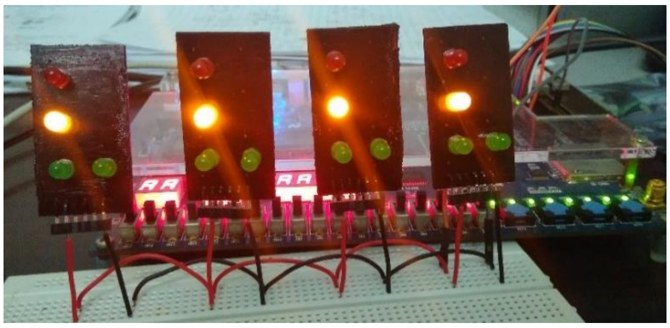


Figure 4.10 : Yellow Light LEDs Testing

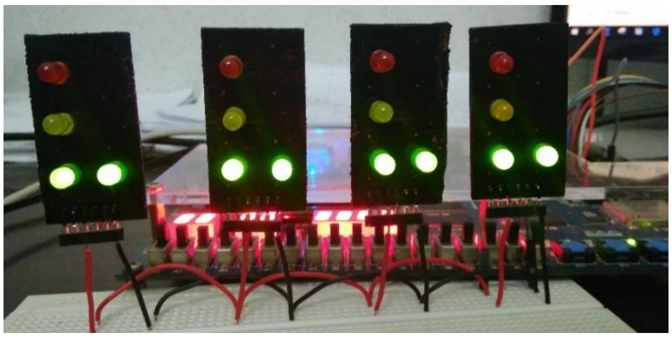


Figure 4.11 : Green Light LEDs Testing

4.2.3 Testing the UART- Serial Communication Interface

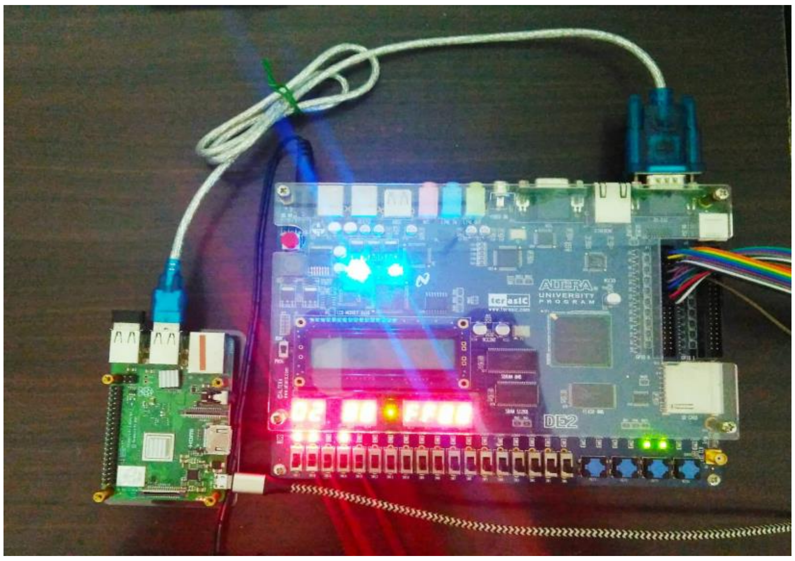
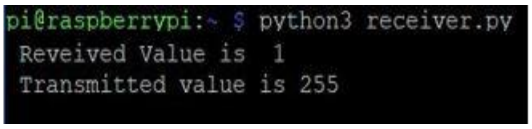
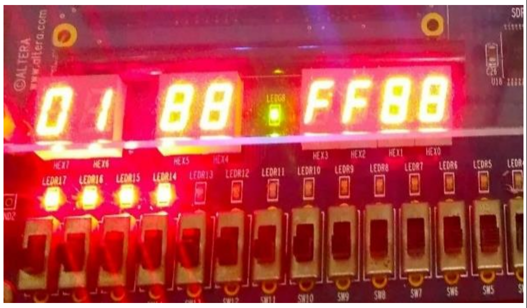


Figure 4.12 : Setup for the UART Interface

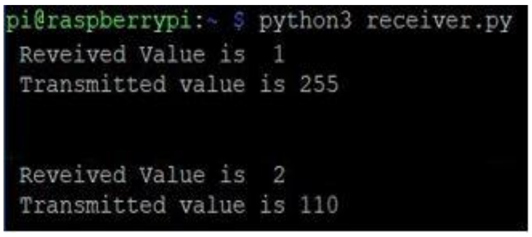
From the Figure 4.1 above, the Raspberry Pi 3B+ and Altera DE2 board were connected using RS232 to USB converter cable. The UART serial communication was programmed into both devices and tested. The data that is transferred from the Raspberry Pi to Altera DE2 board and Altera DE2 board to Raspberry Pi was checked and observed whether the data reached properly, and recorded which is shown in Table 4.1 below.

Table 4.2 : Data Transferred between Raspberry Pi and Altera DE2

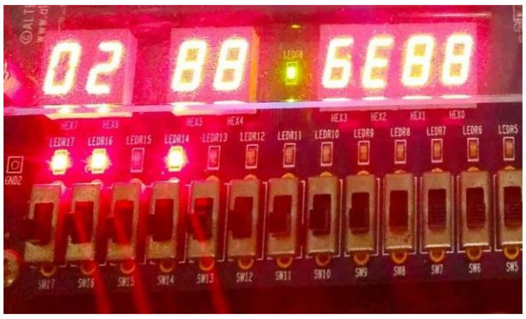
Raspberry Pi Command Prompt	**Altera DE2 Board**


 `pi@raspberrypi:~ $ python3 receiver.py`
 `Reveived Value is  1`
 `Transmitted value is 255`
	

RPI: Received data is 1 and transmitted data is 255. (decimal) **Altera DE2**: Transmitted data is 1 and received data is FF. (hexadecimal)	

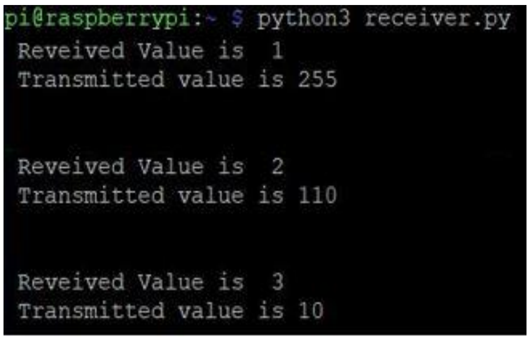


```
pi@raspberrypi:~ $ python3 receiver.py
 Reveived Value is  1
 Transmitted value is 255


 Reveived Value is  2
 Transmitted value is 110
```

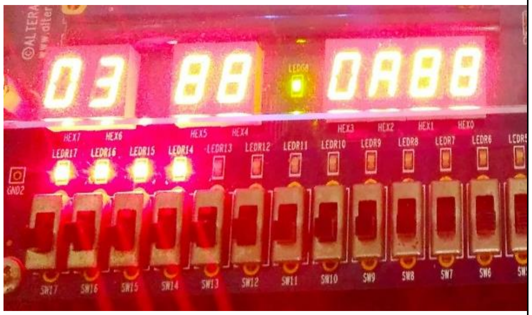


RPI: Received data is 2 and transmitted data is 110. (decimal)

Altera DE2: Transmitted data is 2 and received data is 6E. (hexadecimal)

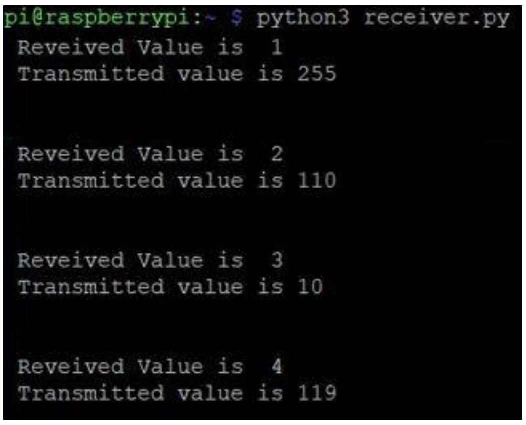


```
pi@raspberrypi:~ $ python3 receiver.py
 Reveived Value is  1
 Transmitted value is 255


 Reveived Value is  2
 Transmitted value is 110


 Reveived Value is  3
 Transmitted value is 10
```

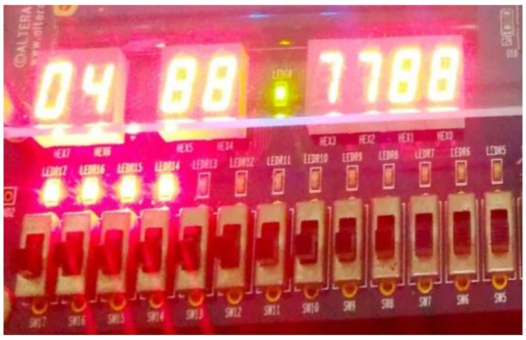


RPI: Received data is 3 and transmitted data is 10. (decimal)

Altera DE2: Transmitted data is 3 and received data is A. (hexadecimal)



```
pi@raspberrypi:~ $ python3 receiver.py
 Reveived Value is  1
 Transmitted value is 255


 Reveived Value is  2
 Transmitted value is 110


 Reveived Value is  3
 Transmitted value is 10


 Reveived Value is  4
 Transmitted value is 119
```



RPI: Received data is 4 and transmitted data is 119. (decimal)

Altera DE2: Transmitted data is 4 and received data is 77. (hexadecimal).

4.3 Prototype Model of Developed Smart Traffic Light Controller

After successfully passing the software and hardware testing phases, a traffic junction model was built using polystyrene and the toy cars were used to imitate the traffic light junction in the real world. Next, every module implemented in the software and hardware, and model were integrated for further functional testing. Figures below show the prototype model of the developed STLCS in different views.

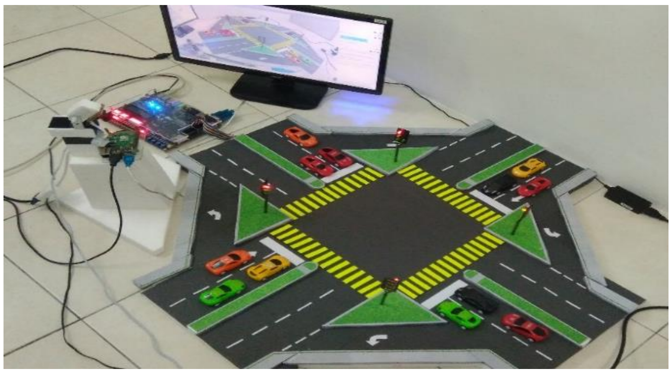


Figure 4.13 : Prototype Model (View 1)

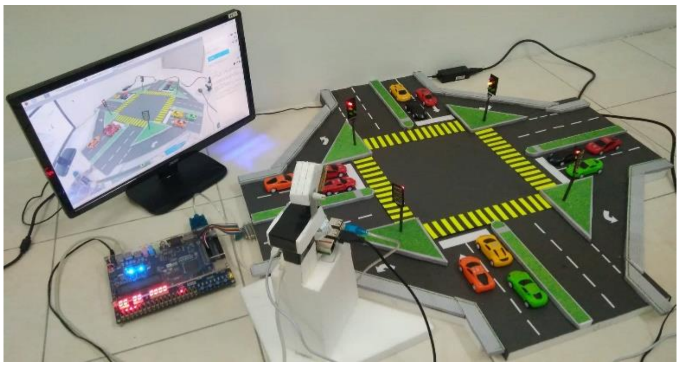


Figure 4.14 : Prototype Model (View 2)

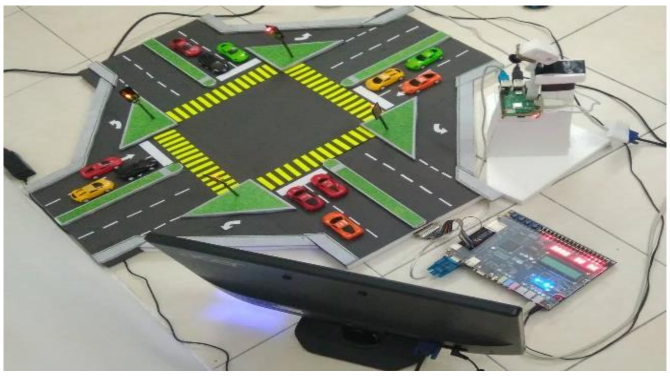


Figure 4.15 : Prototype Model (View 3)

4.4 User Interface of the Developed Smart Traffic Light Controller System

Figure 4.16 below depicts a simple user interface (UI) of the developed system. In the beginning, the UI will display the real-time video captured by the camera and the user needs to draw the ROIs (red, green, blue, pink colour of four-sided polygons) which refer to the boundaries for vehicle detection. After that, some information for instance, number of vehicles detected for four different lanes (near the top left corner) and the calculated green time corresponds to the number of detected vehicle (near the top right corner) will appear automatically after the user input to indicate the system is operating.

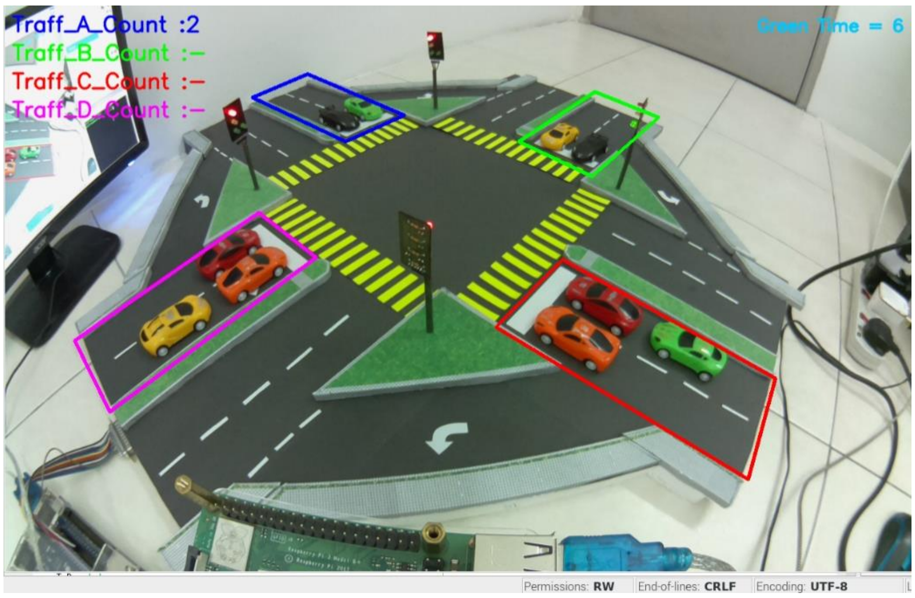


Figure 4.16 : User Interface of the System

4.5 Result from the Developed Smart Traffic Light Controller System

In this phase, few experiments were conducted to validate the effectiveness of the developed system in terms of the principle of work and expected results. From the tables below, different combinations and arrangements of the colour and number of vehicles were assigned into four traffic lanes in the intersection. The actual vehicle number represents the number of actual vehicles in real life, detected vehicles indicate the number of vehicles detected and counted by the computer vision and accuracy represent the percentage error between the actual and measured number of vehicles.

Table 4.3 : The Results Obtained in Test 1

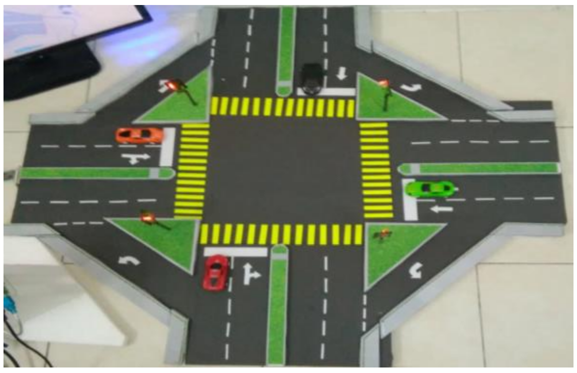
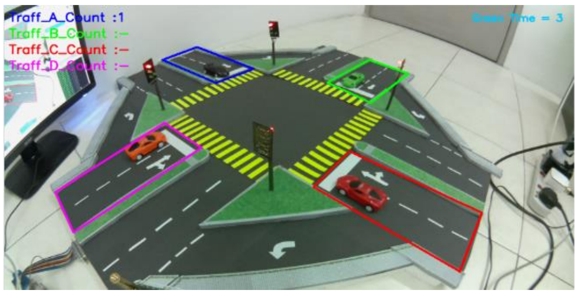
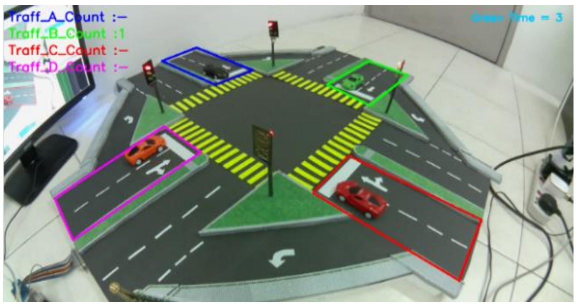
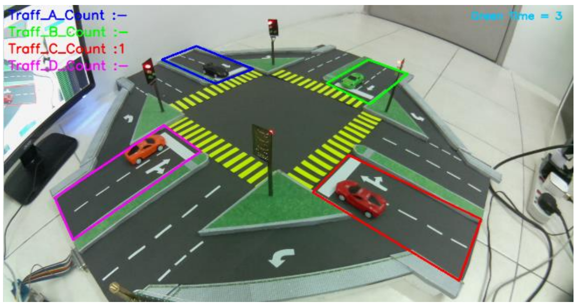
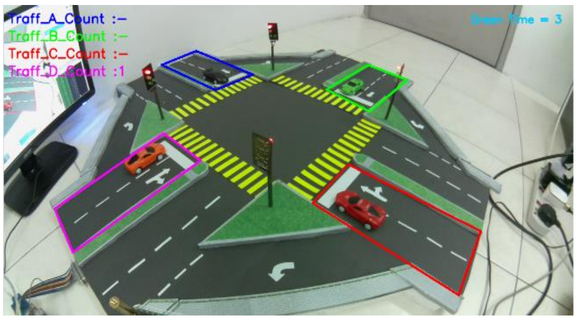
	Actual vehicle number	
	Lane 1: **1** Lane 2: **1** Lane 3: **1** Lane 4: **1**	
	Detected vehicles	**Accuracy(%)**
 Traff_A_Count :1 Traff_B_Count :– Traff_C_Count :– Traff_D_Count :– Green Time = 3	Lane 1: **1** Green time: **5 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :1 Traff_C_Count :– Traff_D_Count :– Green Time = 3	Lane 2: **1** Green time : **5 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :1 Traff_D_Count :– Green Time = 3	Lane 3: **1** Green time : **5 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :– Traff_D_Count :1 Green Time = 3	Lane 4: **1** Green time : **5 sec**	**100%**

Table 4.4 : The Results Obtained in Test 2

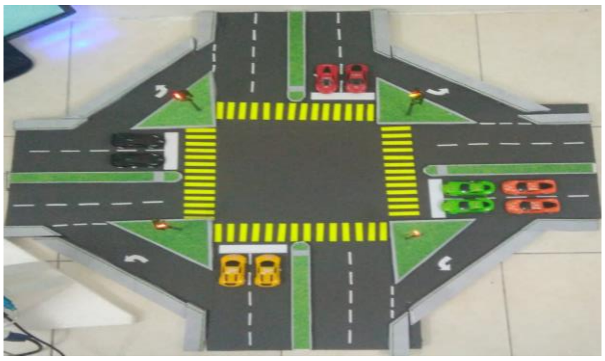
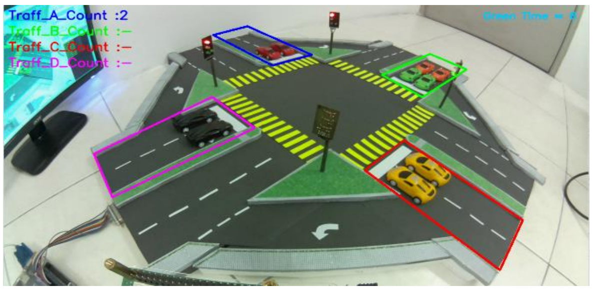
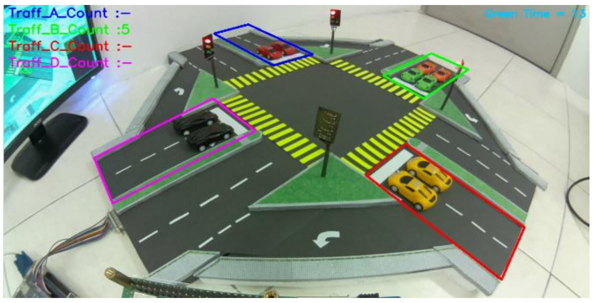
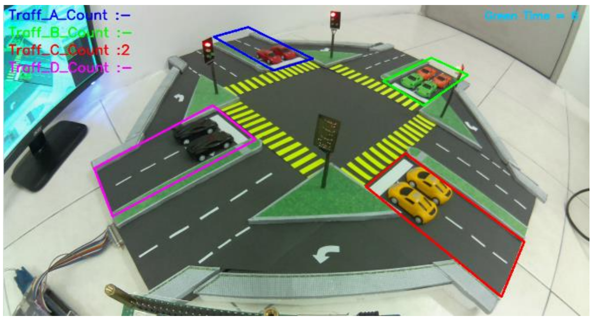
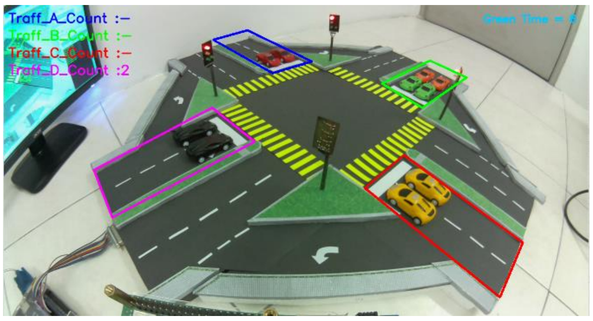
	Actual vehicle number	
	Lane 1: **2** Lane 2: **4** Lane 3: **2** Lane 4: **2**	
	Detected vehicles	**Accuracy(%)**
 Traff_A_Count :2 Traff_B_Count :– Traff_C_Count :– Traff_D_Count :– Green Time = 6	Lane 1: **2** Green time: **6 sec**	**100 %**
 Traff_A_Count :– Traff_B_Count :5 Traff_C_Count :– Traff_D_Count :– Green Time = 13	Lane 2: **5** Green time: **13 sec**	**75 %**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :2 Traff_D_Count :– Green Time = 6	Lane 3: **2** Green time: **6 sec**	**100 %**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :– Traff_D_Count :2 Green Time = 6	Lane 4: **2** Green time: **6 sec**	**100 %**

Table 4.5 : The Results Obtained in Test 3

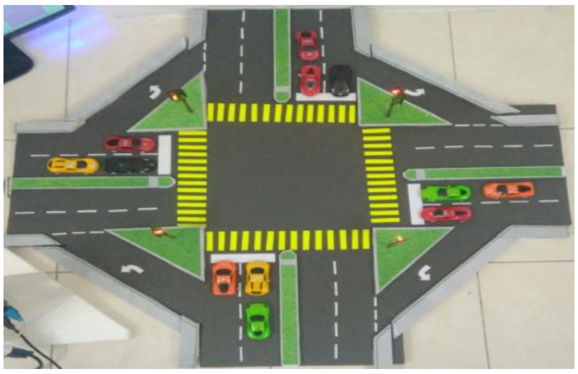
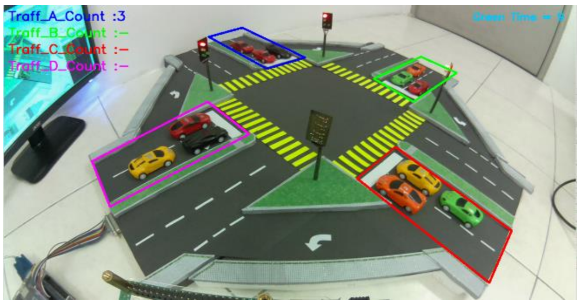
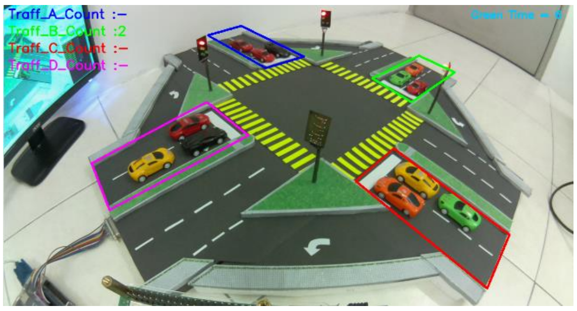
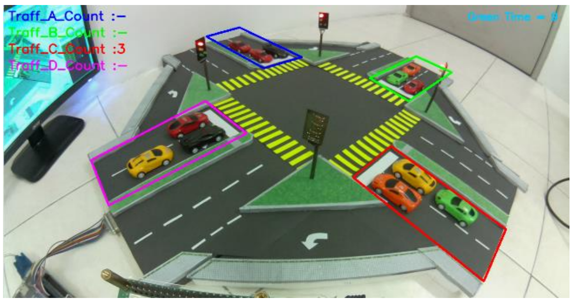
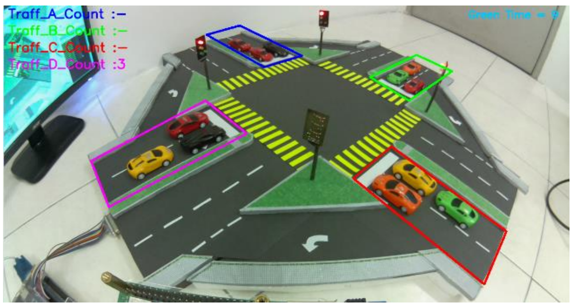
	**Actual vehicle number** Lane 1: **3** Lane 2: **3** Lane 3: **3** Lane 4: **3**	
	Detected vehicles	**Accuracy(%)**
 Traff_A_Count :3 Traff_B_Count :– Traff_C_Count :– Traff_D_Count :– Green Time = 9	Lane 1: **3** Green time: **9 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :2 Traff_C_Count :– Traff_D_Count :–	Lane 2: **2** Green time: **6 sec**	**66.67%**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :3 Traff_D_Count :–	Lane 3: **3** Green time: **9 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :– Traff_D_Count :3	Lane 4: **3** Green time: **9 sec**	**100%**

Table 4.6 : The Results Obtained in Test 4

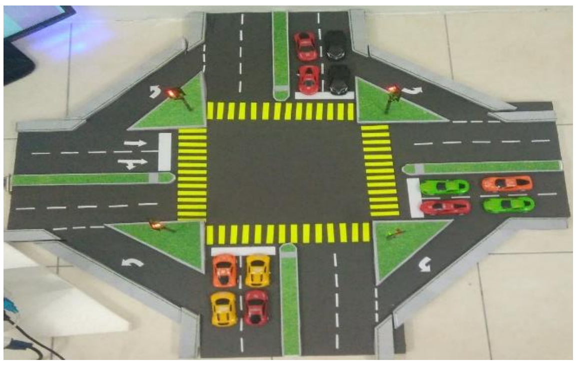
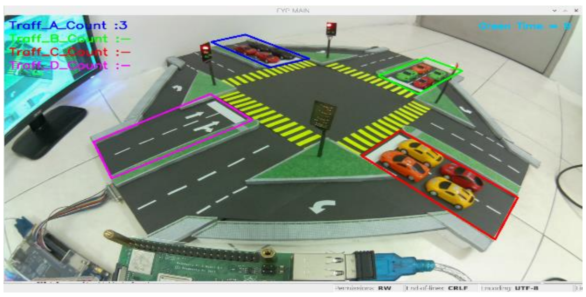
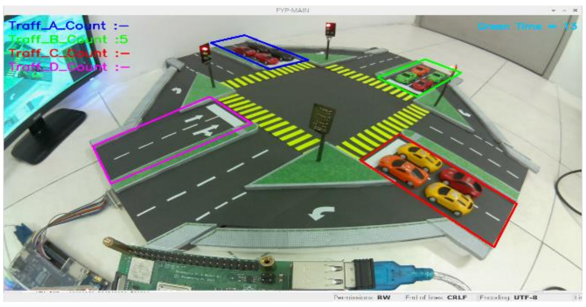
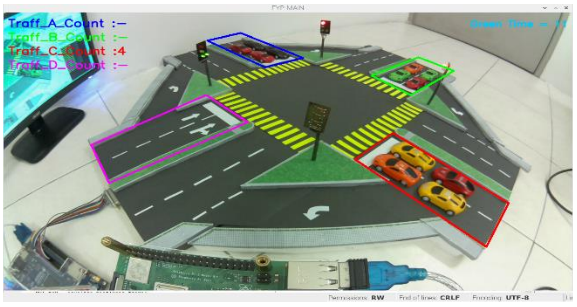
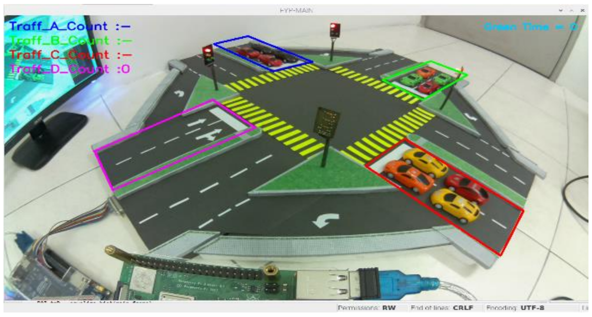
	Actual vehicle number	
	Lane 1: **4** Lane 2: **4** Lane 3: **4** Lane 4: **0**	
	Detected vehicles	**Accuracy(%)**
	Lane 1: **3** Green time: **9 sec**	**75%**
	Lane 2: **5** Green time: **13 sec**	**75%**
	Lane 3: **4** Green time: **11 sec**	**100%**
	Lane 4: **0** Green time: **5 sec**	**100%**

Table 4.7 : The Results Obtained in Test 5

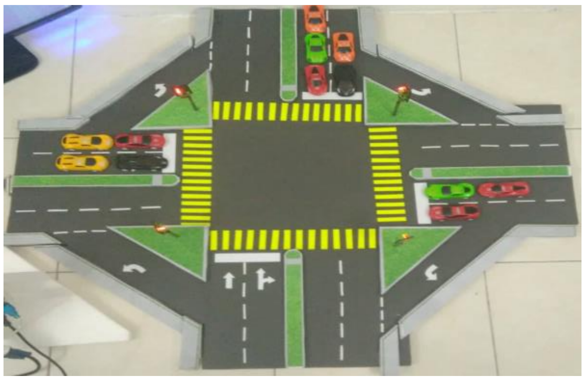
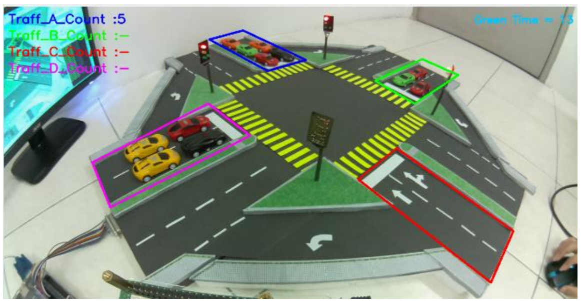
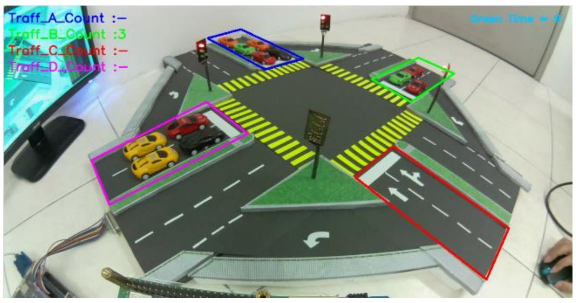
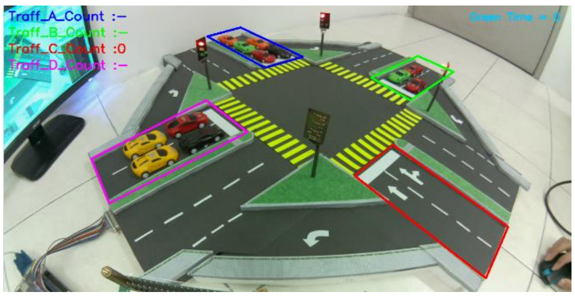
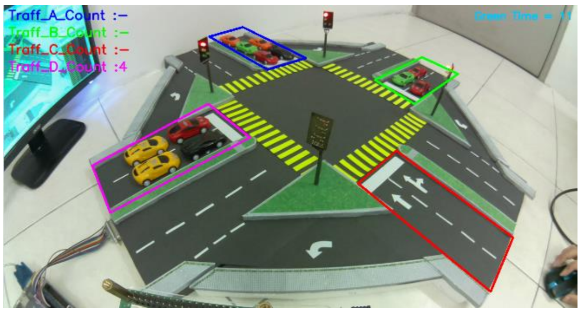
	Actual vehicle number	
	Lane 1: **5** Lane 2: **3** Lane 3: **0** Lane 4: **4**	
	Detected vehicles	**Accuracy(%)**
 Traff_A_Count :5 Traff_B_Count :– Traff_C_Count :– Traff_D_Count :– Green Time = 13	Lane 1: **5** Green time: **13 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :3 Traff_C_Count :– Traff_D_Count :– Green Time = 9	Lane 2: **3** Green time: **9 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :0 Traff_D_Count :– Green Time = 5	Lane 3: **0** Green time: **5 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :– Traff_D_Count :4 Green Time = 11	Lane 4: **4** Green time: **11 sec**	**100%**

Table 4.8 : The Results Obtained in Test 6

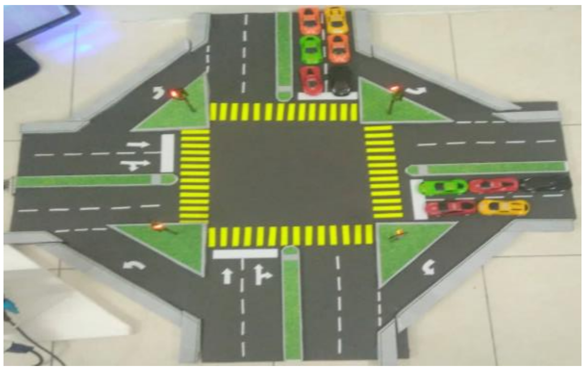
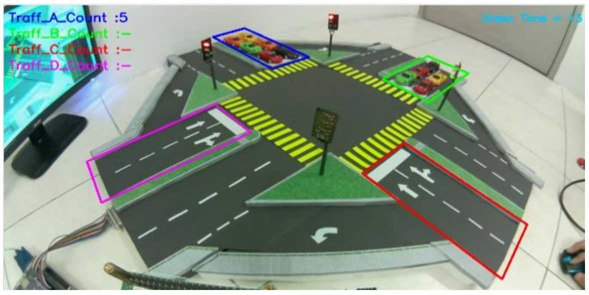
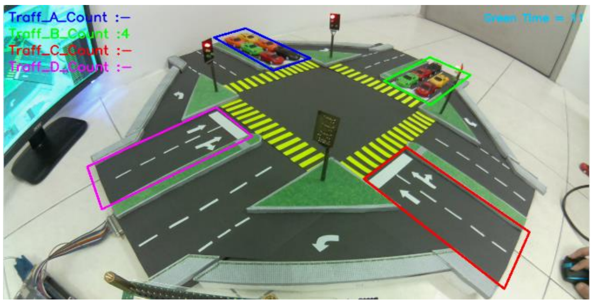
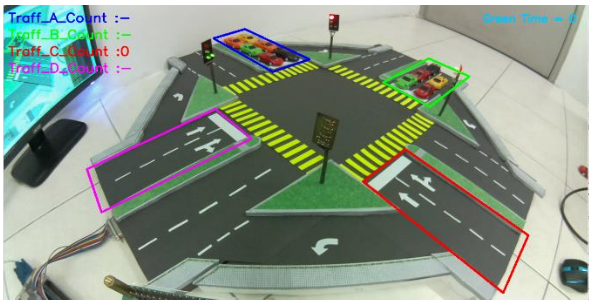
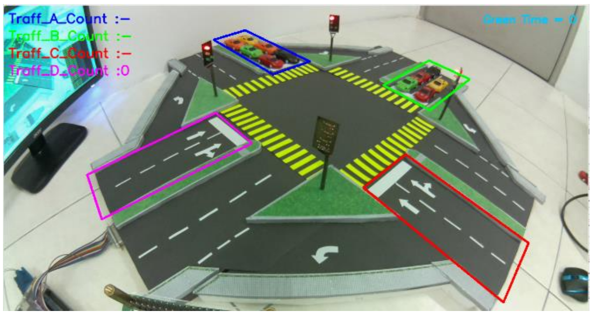
	**Actual vehicle number** Lane 1: **6** Lane 2: **5** Lane 3: **0** Lane 4: **0**	
 Traff_A_Count :5 Traff_B_Count :– Traff_C_Count :– Traff_D_Count :– Green Time = 13	**Detected vehicles** Lane 1: **5** Green time: **13 sec**	**Accuracy(%)** **83.33%**
 Traff_A_Count :– Traff_B_Count :4 Traff_C_Count :– Traff_D_Count :– Green Time = 11	Lane 2: **4** Green time: **11 sec**	**80%**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :0 Traff_D_Count :– Green Time = 5	Lane 3: **0** Green time: **5 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :– Traff_D_Count :0 Green Time = 5	Lane 4: **0** Green time: **5 sec**	**100%**

Table 4.9 : The Results Obtained in Test 7

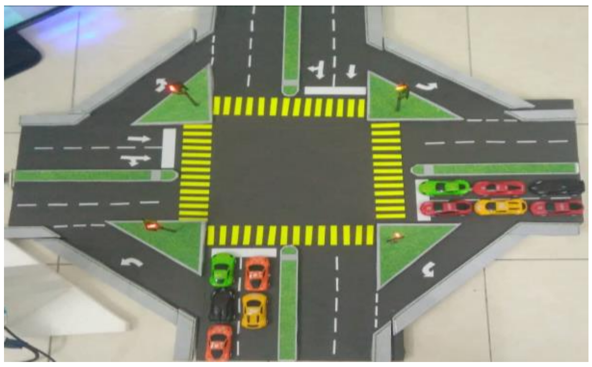
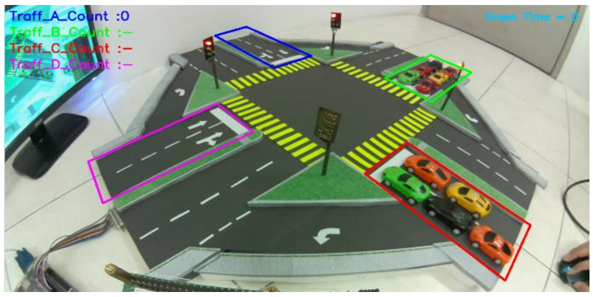
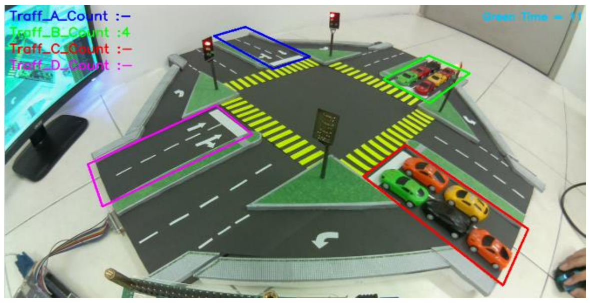
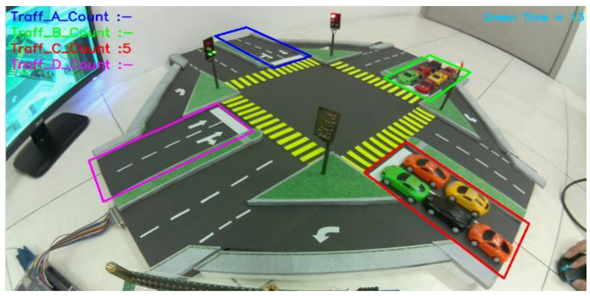
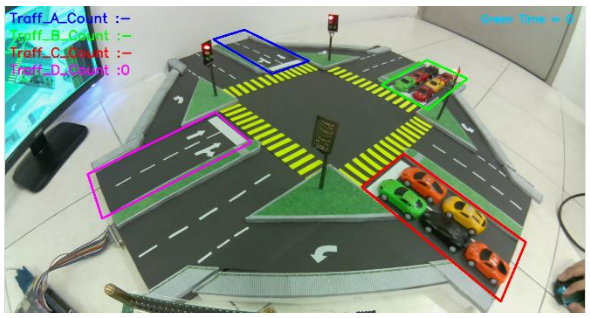
	Actual vehicle number	
	Lane 1: **0** Lane 2: **6** Lane 3: **5** Lane 4: **0**	
	Detected vehicles	**Accuracy(%)**
	Lane 1: **0** Green time: **5 sec**	**100%**
	Lane 2: **4** Green time: **11sec**	**66.67%**
	Lane 3: **5** Green time: **13 sec**	**100%**
	Lane 4: **0** Green time: **5 sec**	**100%**

Table 4.10 : The Results Obtained in Test 8

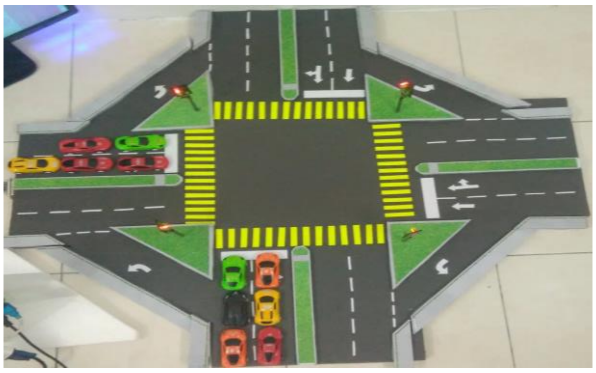
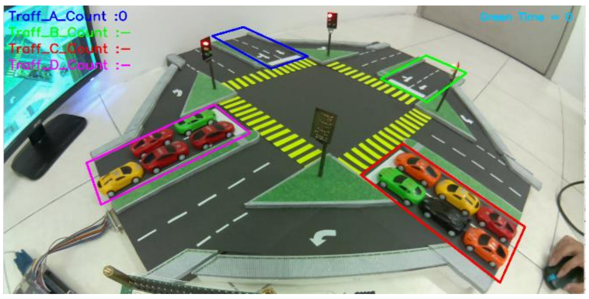
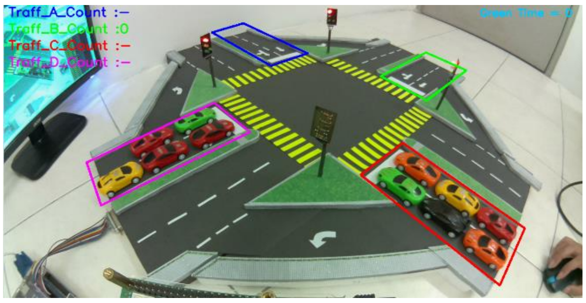
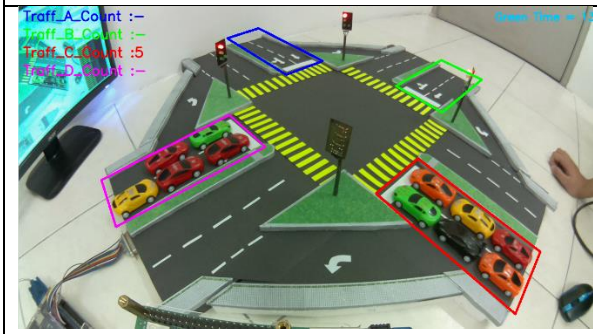
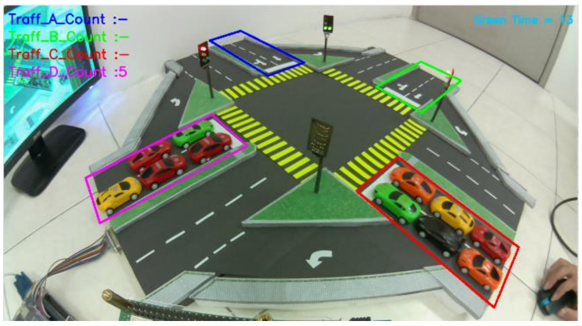
	Actual vehicle number	
	Lane 1: **0** Lane 2: **0** Lane 3: **6** Lane 4: **5**	
	Detected vehicles	**Accuracy(%)**
 Traff_A_Count :0 Traff_B_Count :– Traff_C_Count :– Traff_D_Count :–	Lane 1: **0** Green time: **5 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :0 Traff_C_Count :– Traff_D_Count :–	Lane 2: **0** Green time: **5 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :5 Traff_D_Count :–	Lane 3: **5** Green time: **13 sec**	**83.33%**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :– Traff_D_Count :5	Lane 4: **5** Green time: **13 sec**	**100%**

Table 4.11 : The Results Obtained in Test 9

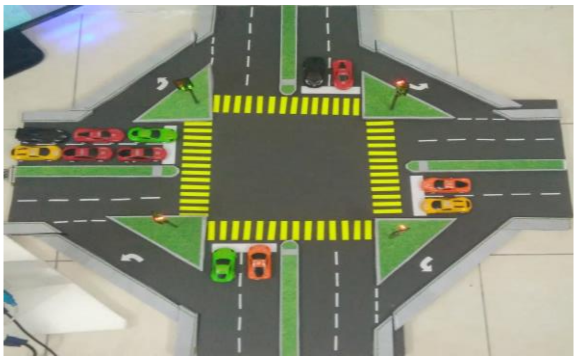
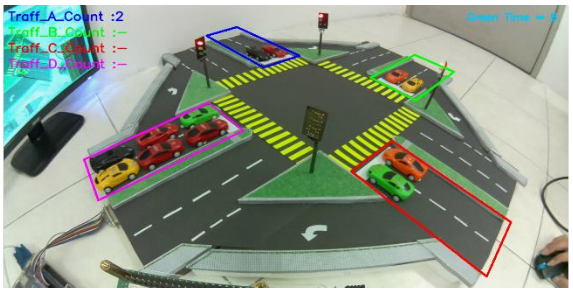
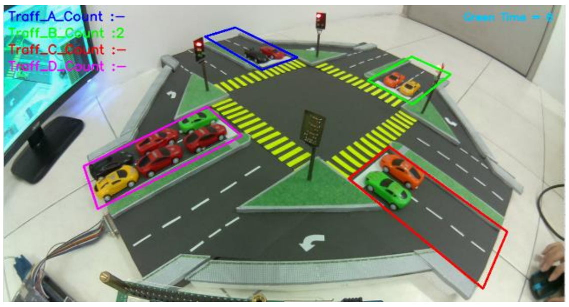
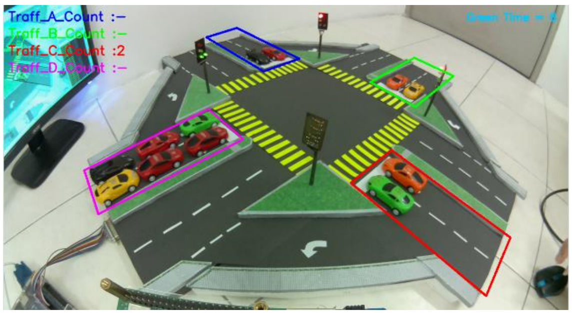
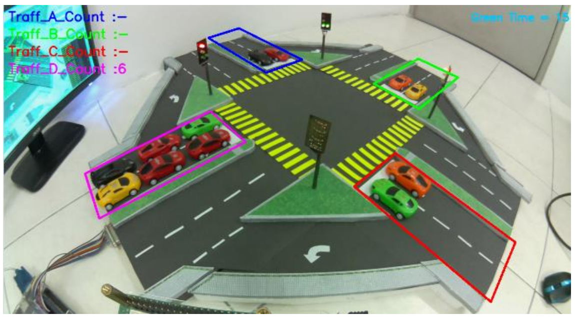
	Actual vehicle number	
	Lane 1: **2** Lane 2: **2** Lane 3: **2** Lane 4: **6**	
	Detected vehicles	**Accuracy(%)**
 Traff_A_Count :2 Traff_B_Count :– Traff_C_Count :– Traff_D_Count :– Green Time = 6	Lane 1: **2** Green time: **6 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :2 Traff_C_Count :– Traff_D_Count :– Green Time = 6	Lane 2: **2** Green time: **6 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :2 Traff_D_Count :– Green Time = 6	Lane 3: **2** Green time: **6 sec**	**100%**
 Traff_A_Count :– Traff_B_Count :– Traff_C_Count :– Traff_D_Count :6 Green Time = 15	Lane 4: **6** Green time: **15 sec**	**100%**

Table 4.12 : Summary of the Results Obtained

Test	Lanes	Actual number of vehicles	Detected number of vehicles	Accuracy (%)	Calculated green time, s
1	1	1	1	100	5
	2	1	1	100	5
	3	1	1	100	5
	4	1	1	100	5
2	1	2	2	100	6
	2	4	5	75	13
	3	2	2	100	6
	4	2	2	100	6
3	1	3	3	100	9
	2	3	2	66.67	6
	3	3	3	100	9
	4	3	3	100	9
4	1	4	3	75	9
	2	4	5	75	13
	3	4	4	100	11
	4	0	0	100	5
5	1	5	5	100	13
	2	3	3	100	9
	3	0	0	100	5
	4	4	4	100	11
6	1	6	5	83.33	13
	2	5	4	80	11
	3	0	0	100	5
	4	0	0	100	5
7	1	0	0	100	5
	2	6	4	66.67	11
	3	5	5	100	13
	4	0	0	100	5

8	1	0	0	100	5
	2	0	0	100	5
	3	6	5	83.33	13
	4	5	5	100	13
9	1	2	2	100	6
	2	2	2	100	6
	3	2	2	100	6
	4	6	6	100	15
Total		**95**	**90**	**94.74**	

Based on the results obtained in the summary table above, the frequency of the miscounting is high in lane 1 and lane 2, this issue is most likely caused by the morphological operations and the segmentation algorithms used in the vehicle detection process. In the testing process, if the vehicles with common color properties are placed compactly together, morphological closing tends to join the binary objects together and may cause the segmentation algorithms over-segment the objects and this eventually leads the system counting the vehicles more than the actual number. Moreover, if two vehicles that have different color properties but queuing in series, are close enough together and are far away from the camera, some portion of the detection area for the vehicle queuing behind is blocked by vehicle in front, if the detection area is small, the system will consider it as a noise and ignore it in the counting process. In general, the developed system is able to generate acceptable results in a few tests, the overall vehicle detection accuracy is more than 90% which proved the effectiveness of the methodology used in the system.

4.6 Limitations of the Developed System

Despite the fact that some of the problems found in the existing traffic light controller system has been solved successfully in this project; some limitations still exists in the developed STLCS. The limitations of this system are as follows:

i. When the vehicles are too close to each other, the system tends to miscount the vehicles, this will affect the accuracy of the result and calculate an undesirable green time to discharge the vehicles.

ii. In this system, although the image enhancement technique is applied to enhance the brightness and recover some details in low illumination image, but this approach is just able to improve the image to some extent, this caused the system to not respond well under unfavourable lighting conditions for instance during night time, rainy days, or in poorly illuminated environment.

iii. In this project, the high-resolution video (1280 x 800) is used to obtain decent detection accuracy. But at the same time, high-resolution video processing requires more computational resources and caused the Raspberry Pi tends to be fully loaded easily. As a consequence, the frame rate obtained is relatively low (around 5 fps).

iv. Since the detection algorithms used are not vigorous enough to perfectly count the vehicle in day time and night time. Therefore, the implemented STLCS is still unable to skip the lane for now just in case of detection error.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The project “Development of FPGA based Smart Traffic Light Controller System with Image Processing” which aims to improve the efficiency of existing automatic traffic light signalling system by reducing the queuing time of each lane of the vehicles and maximize the discharge of vehicles across an intersection was successfully designed and developed.

The development of STLCS is divided into three major parts which are the automatic traffic light control system, vehicle detection system and the UART communication system. The implementation of the vehicle detection system was done by using the Raspberry Pi 3B+ and written in Python programming language, the vehicle detection approach emphasizes on the segmentation that is based on colour properties of the object and the overall accuracy of this approach is 94.74%. Furthermore, a full-duplex UART communication system was constructed on both the Altera FPGA board and the Raspberry Pi to allow the communication of multiple designed systems. Moreover, an automatic traffic light control system based on FPGA was constructed successfully on the Altera FPGA board using the Mealy finite-state machine design, the system can communicate with the vehicle detection system to enhance the flexibility in traffic signal control.

Although the developed system has not been tested in the actual field, the performance of the system was evaluated and satisfactory results were obtained. As mentioned above, some limitations still exist in the system and those limitations need to be addressed and a few other modifications as well as enhancements can be made in the future so that the system can be applied in the actual field.

5.2 Recommendations

There are several enhancements that can be made in future to address the limitations as mentioned previously and improve the robustness of the STLCS to cope with unpredictable environment. The recommendations for the enhancements are as follows:

i. A powerful single board computer could be used so that the machine learning techniques such as K-means clustering, Deep Neural Network (DNNs), Convolutional Neural Network (CNN), etc. can be used for more robust vehicle detection.

ii. Using a more advanced image processing algorithms and libraries so that the system is able to operate properly regardless of the unfavourable weather and illuminating conditions.

iii. Use multi-camera to monitor each lane in the junction instead of single camera monitor on all traffic lanes. This approach will solve the detection error due to the blockage of the rear vehicle by the front vehicle and improve the performance of vehicle detection and counting.

iv. Utilize the deep learning method to train a highly complex and advanced traffic management model that is able to control the traffic flow in the traffic junction in a more efficient and effective manner.

REFERENCES

Agho, O., Faisal, S. B., and Ganiyu, B. (2015). *Field Programmable Gate Array Based Intelligent Traffic Light System*. Abubakar Tafawa Balewa University, Department of Mechatronic and Systems Engineering, 4(11), pp,10-16.

Andrey. (2014). *What is a color model?* [online] Available at: https://www.script-tutorials.com/what-is-a-color-model/ [Accessed: 25 June 2019]

Alexander, M. and Abid Rahman K. (2014). *Introduction to OpenCV-Python Tutorials.* [online] Available at: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_setup /py_intro /py_intro.html#intro [Accessed: 4 March. 2019].

Alemu, A. (2015). *Automatic Traffic Light Control System Using Image Processing In Ethiopia*. [online] Available at: https://www.academia.edu/23228882/Automatic_Traffic_Light_Control_System_Using_Image_Processing_In_Ethiopia. [Accessed: 25 February. 2019].

Almawagani, A.H.M. (2018). *DESIGN OF REAL TIME SMART TRAFFIC LIGHT CONTROL SYSTEM.* Najran University, Department of Electrical Engineering. pp.51-55.

Bilal, G., Khaled, E., Khaled, C., and Mohamad Kherfan. (2016). *Smart Traffic Light Control System*. Faculty of Sciences IV Lebanese University and School of Engineering Lebanese International University, pp.161-166.

Dilip, B., Alekhya, Y. and Divya, P.B. (2012). *FPGA Implementation of Advanced Traffic Light Controller using Verilog HDL*. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), 1(7), pp.1-6.

Gholamreza Anbarjafari (2014) *Introduction to image processing*. [online] Available at: https://sisu.ut.ee/imageprocessing/book/1 [Accessed: 4 March. 2019].

Ivan Culjak - *IEEE Conference Publication*. (2012). *A brief Introduction to OpenCV*. [online] Available at: https://ieeexplore.ieee.org/document/6240859 [Accessed 3 March. 2019].

Mitu, R., 2017. *UART Receiver Synchronization: Investing the Maximum Tolerable Clock Frequency Deviation.* Indian Journal of Science and Technology. Kerala University, Department of Electronics and Communication, Electronic Research and Development Centre of India, 10 (25), pp. 1-5

Neha R. Laddha and Prof A.P. Thakare (2013). *Implementation of serial communication using UART with configurable baud rate*. International Journal of Recent and Innovation Trends in Computing and Communication, 1(4), pp. 263-268.

Nidhi, D. A. and Amit, S. (2015). *Intelligent Real Time Traffic Controller Using Image Processing*. Raison College of Engineering and Technology, Amravati, India, 4(4), pp.2980-2983.

Philipp, C., Christian, K. and Maher, Y. (2019). *DE2 Board Cyclone II FPGA Development Kit*. [online] Available at: https://www.secs.oakland.edu/~ ganesan/ ece576f10project/index_files/Page361.htm [Accessed: 12 March. 2019].

Parichita, B. and Ramandeep, K. (2016). *Intelligent Traffic Controller System using Image Processing*. Guru Gobind Signh Indraprastha University, Bharati Vidyapeeth's College of Engineering, Department of Information Technology. International Journal of Science and Research (IJSR), 5(8), pp.1396-1398.

Parthasarathi, V., Surya1, M., Akshay, B., Murali Siva, K. and Shriram, K. Vasudevan. (2015). *Smart Control of Traffic Signal System using Image Processing*. Amrita Vishwa Vidyapeetham University, Department of Electrical and Electronics Engineering and Department of Computer Science and Engineering, 8(16), pp.1-5.

Rakesh, V.S. and Shaithya, V. (2007). A Traffic Control System Using Inductive Loop Detector. International Journal of Advanced Research in Electrical and Electronics and Instrumentation Engineering, 4(5), pp.4590-4593.

Sinhmar, P. (2012). *Intelligent traffic light and density control using IR sensors and microcontroller*. International journal of advanced technology & engineering research (IJATER), 2(2), pp. 30-35.

Terasic Technologies (2012) *Terasic - Phased Out - Main Boards - Altera DE2 Board*. [online] Available at: https://www.terasic.com.tw/cgi-bin/page/archive.pl?No=30 [Accessed: 5 March. 2019].

The 4 important color models for presentation design (2016) [online]. Available at: http://presentitude.com/color-theory-part-iii/ [Accessed: 25 Jun. 2019].

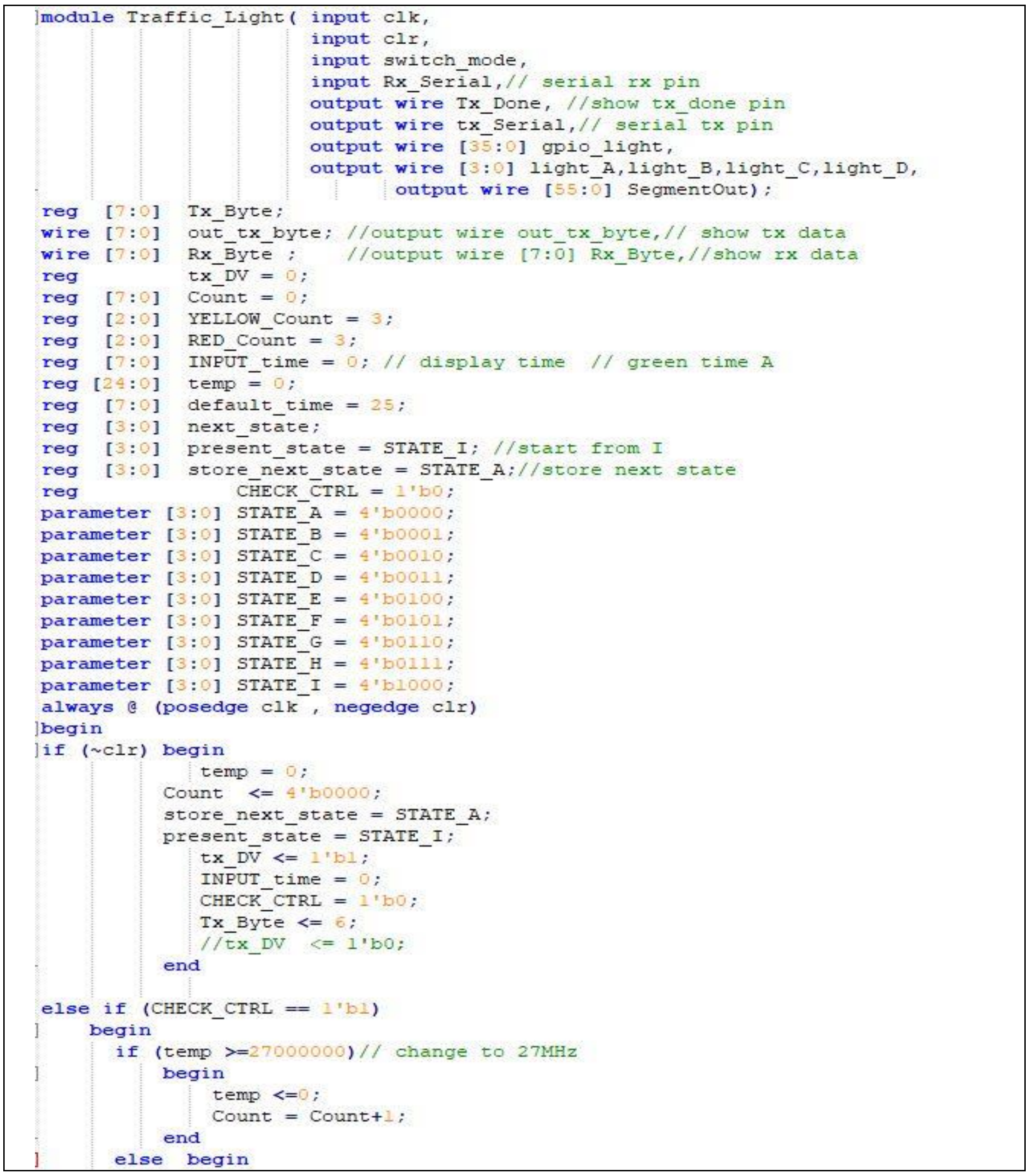
Tom, H. (2001). *How Red-light Cameras Work*. [online]. Available at: https://auto.howstuffworks.com/car-driving-safety/safety-regulatory-devices/red-light-camera1.htm [Accessed 23 March. 2019].

UART Communication with Spartan3an FPGA Project Kit (2017) [online] Available at: https://www.pantechsolutions.net/fpga-tutorials/uart-communication-with-spartan3an-fpga-project-kit [Accessed: 5 March. 2019].

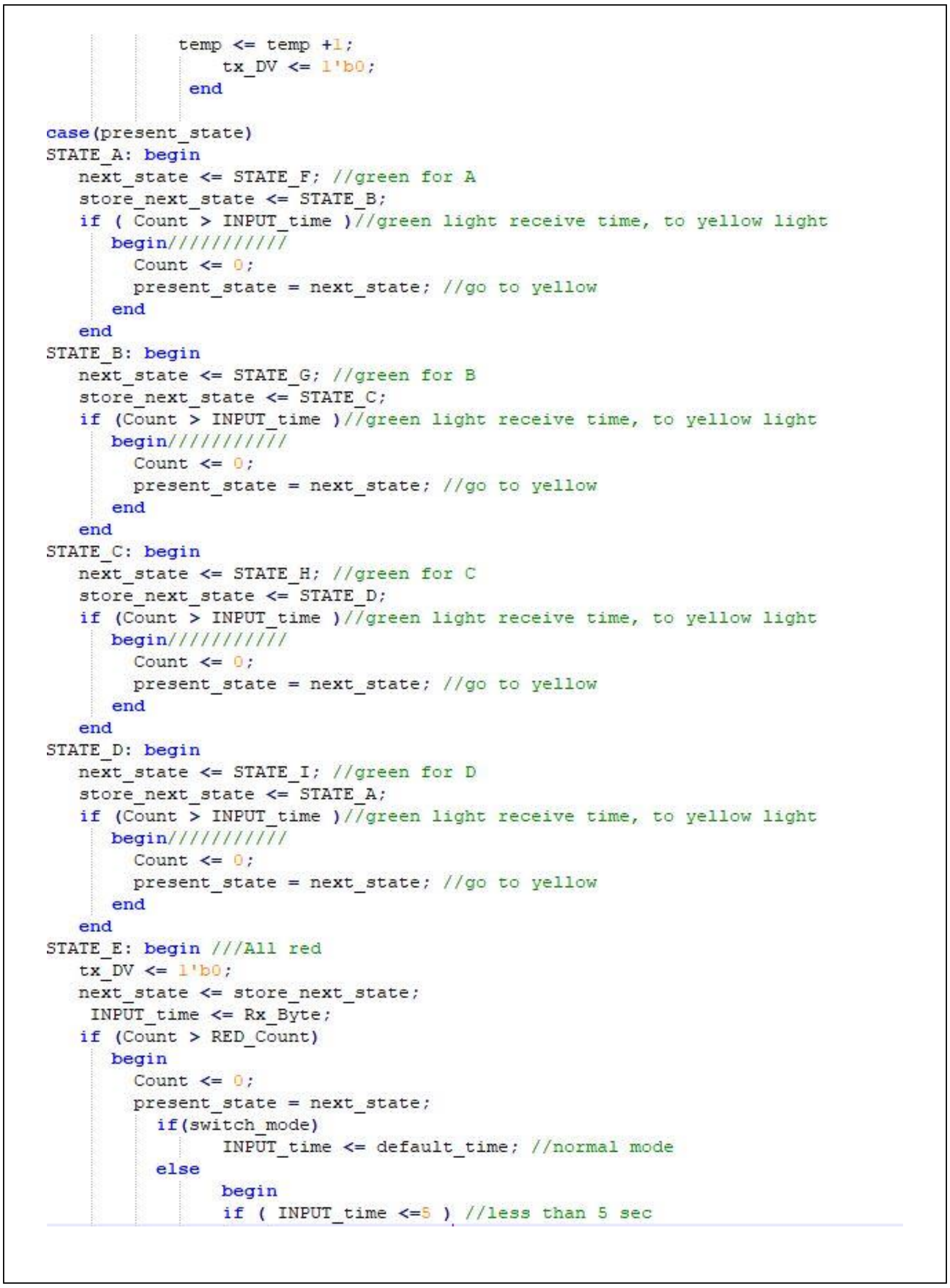
Umesh Lokhande (2017) *Introduction of Raspberry Pi 3 Model B* [online]. Available at: https://binaryupdates.com/introduction-of-raspberry-pi-3-model-b/ [Accessed: 10 March. 2019].

APPENDICES

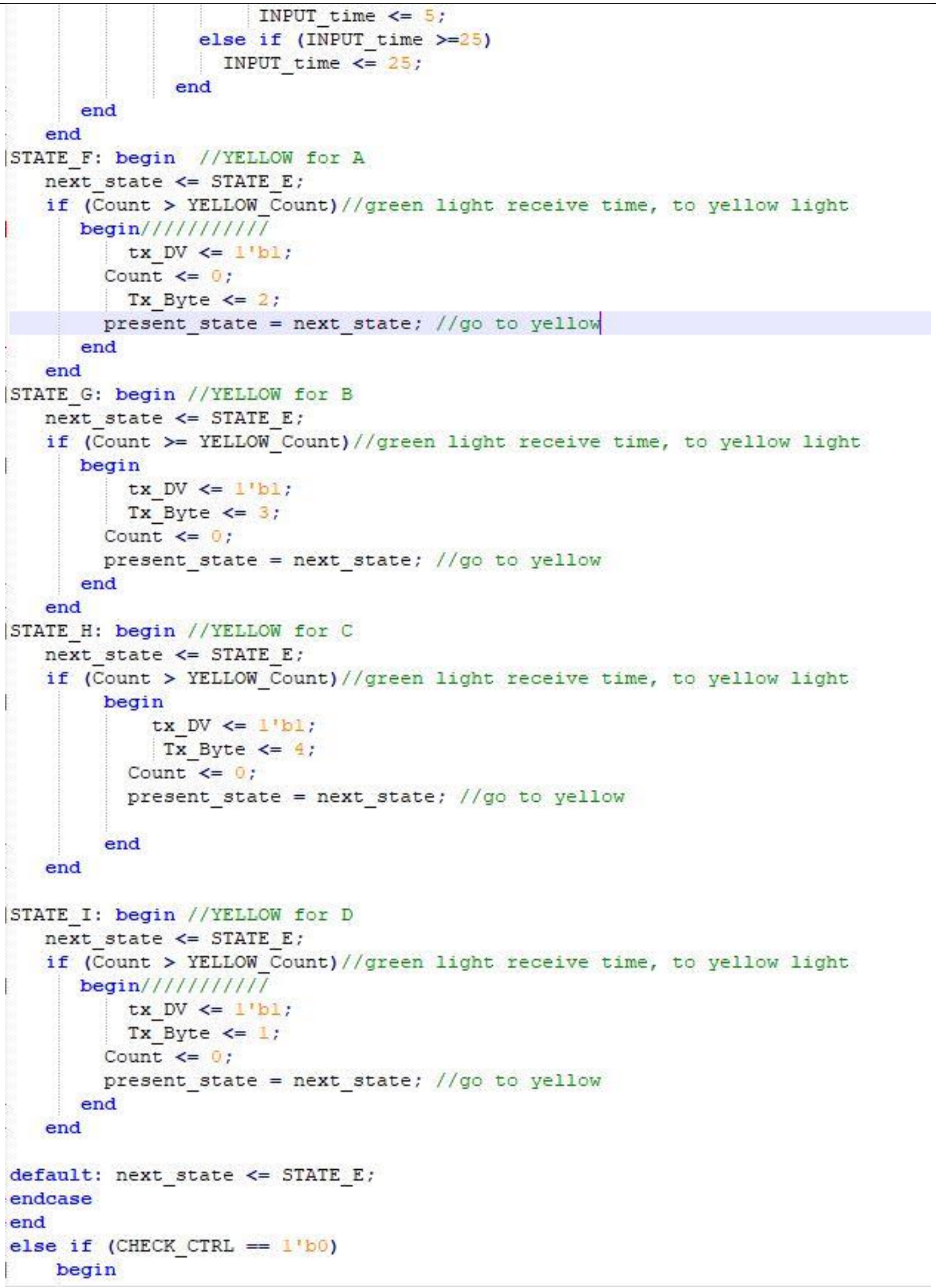
Appendix A: Source code of Traffic Light Controller



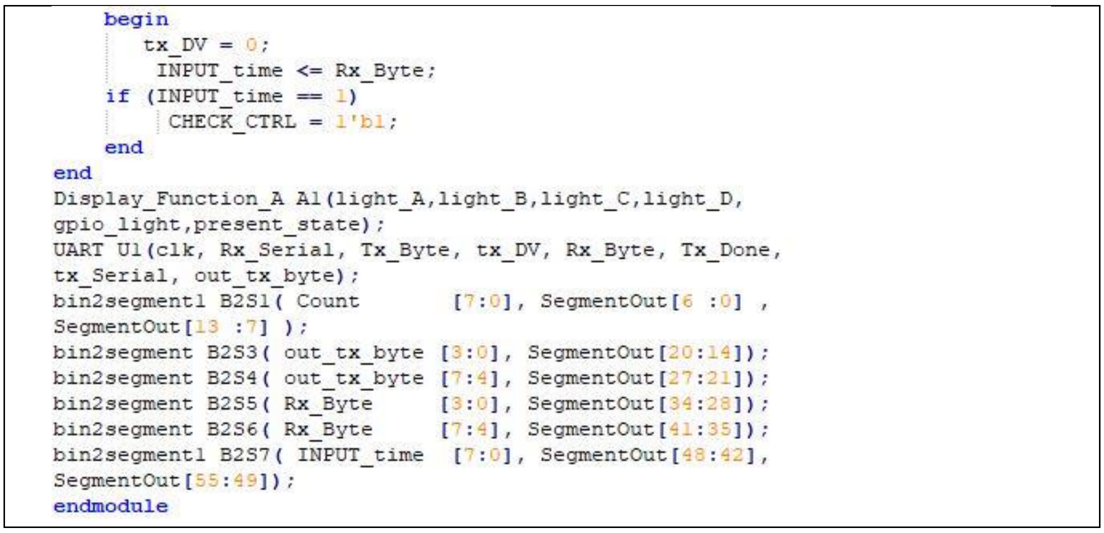
```
module Traffic_Light( input clk,
                      input clr,
                      input switch_mode,
                      input Rx_Serial,// serial rx pin
                      output wire Tx_Done, //show tx_done pin
                      output wire tx_Serial,// serial tx pin
                      output wire [35:0] gpio_light,
                      output wire [3:0] light_A,light_B,light_C,light_D,
                             output wire [55:0] SegmentOut);
reg  [7:0]  Tx_Byte;
wire [7:0]  out_tx_byte; //output wire out_tx_byte,// show tx data
wire [7:0]  Rx_Byte ;    //output wire [7:0] Rx_Byte,//show rx data
reg         tx_DV = 0;
reg  [7:0]  Count = 0;
reg  [2:0]  YELLOW_Count = 3;
reg  [2:0]  RED_Count = 3;
reg  [7:0]  INPUT_time = 0; // display time  // green time A
reg [24:0]  temp = 0;
reg  [7:0]  default_time = 25;
reg  [3:0]  next_state;
reg  [3:0]  present_state = STATE_I; //start from I
reg  [3:0]  store_next_state = STATE_A;//store next state
reg             CHECK_CTRL = 1'b0;
parameter [3:0] STATE_A = 4'b0000;
parameter [3:0] STATE_B = 4'b0001;
parameter [3:0] STATE_C = 4'b0010;
parameter [3:0] STATE_D = 4'b0011;
parameter [3:0] STATE_E = 4'b0100;
parameter [3:0] STATE_F = 4'b0101;
parameter [3:0] STATE_G = 4'b0110;
parameter [3:0] STATE_H = 4'b0111;
parameter [3:0] STATE_I = 4'b1000;
always @ (posedge clk , negedge clr)
begin
if (~clr) begin
             temp = 0;
          Count  <= 4'b0000;
          store_next_state = STATE_A;
          present_state = STATE_I;
             tx_DV <= 1'b1;
             INPUT_time = 0;
             CHECK_CTRL = 1'b0;
             Tx_Byte <= 6;
             //tx_DV  <= 1'b0;
          end

else if (CHECK_CTRL == 1'b1)
    begin
      if (temp >=27000000)// change to 27MHz
          begin
              temp <=0;
              Count = Count+1;
          end
      else  begin
```



```
            temp <= temp +1;
                tx_DV <= 1'b0;
             end

case(present_state)
STATE_A: begin
   next_state <= STATE_F; //green for A
   store_next_state <= STATE_B;
   if ( Count > INPUT_time )//green light receive time, to yellow light
      begin//////////
        Count <= 0;
        present_state = next_state; //go to yellow
      end
   end
STATE_B: begin
   next_state <= STATE_G; //green for B
   store_next_state <= STATE_C;
   if (Count > INPUT_time )//green light receive time, to yellow light
      begin//////////
        Count <= 0;
        present_state = next_state; //go to yellow
      end
   end
STATE_C: begin
   next_state <= STATE_H; //green for C
   store_next_state <= STATE_D;
   if (Count > INPUT_time )//green light receive time, to yellow light
      begin//////////
        Count <= 0;
        present_state = next_state; //go to yellow
      end
   end
STATE_D: begin
   next_state <= STATE_I; //green for D
   store_next_state <= STATE_A;
   if (Count > INPUT_time )//green light receive time, to yellow light
      begin//////////
        Count <= 0;
        present_state = next_state; //go to yellow
      end
   end
STATE_E: begin ///All red
   tx_DV <= 1'b0;
   next_state <= store_next_state;
    INPUT_time <= Rx_Byte;
   if (Count > RED_Count)
      begin
        Count <= 0;
        present_state = next_state;
          if(switch_mode)
                INPUT_time <= default_time; //normal mode
          else
                begin
                if ( INPUT_time <=5 ) //less than 5 sec
```

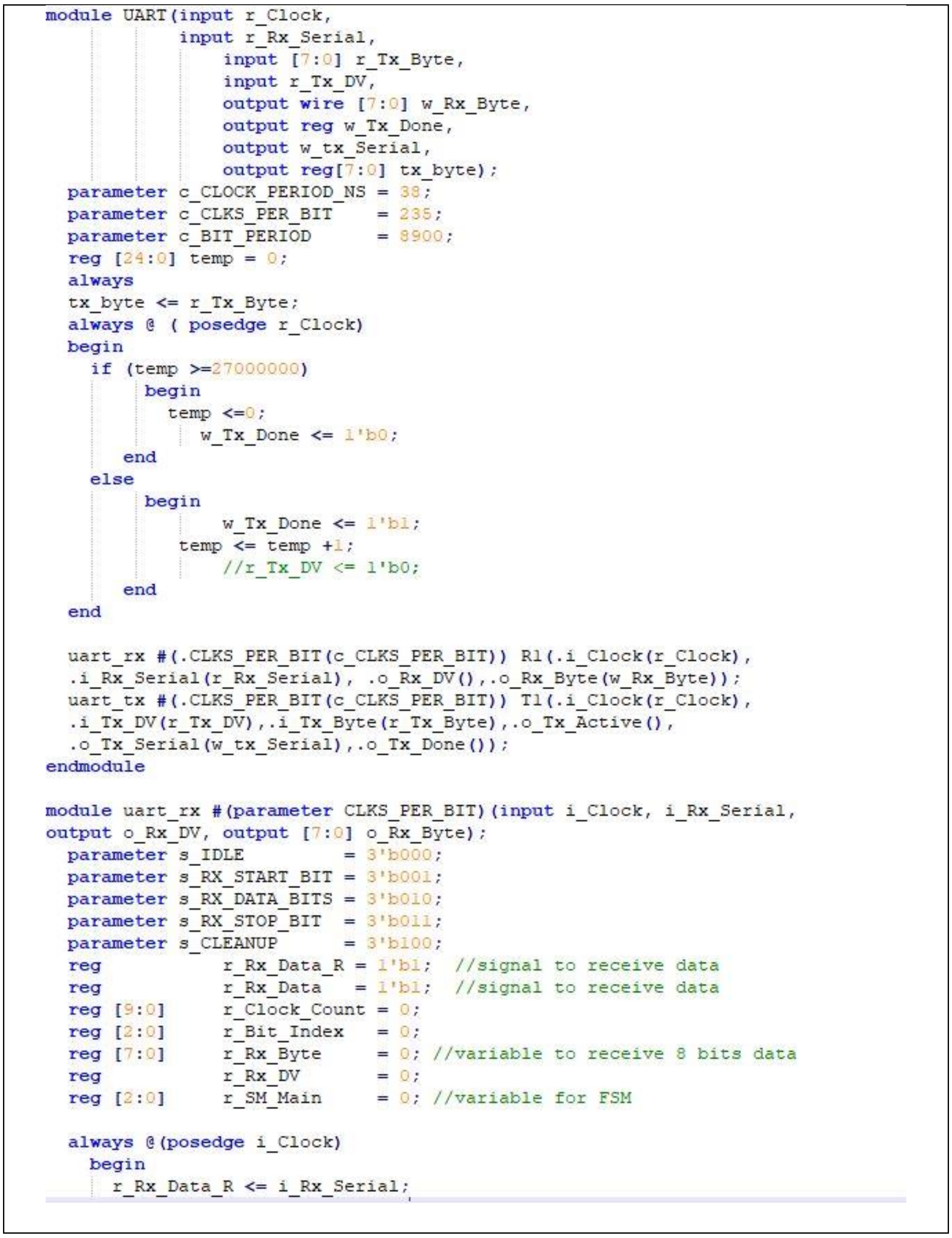


```
                    INPUT_time <= 5;
                else if (INPUT_time >=25)
                  INPUT_time <= 25;
              end
       end
    end
STATE_F: begin  //YELLOW for A
    next_state <= STATE_E;
    if (Count > YELLOW_Count)//green light receive time, to yellow light
       begin//////////
           tx_DV <= 1'b1;
         Count <= 0;
           Tx_Byte <= 2;
         present_state = next_state; //go to yellow
       end
    end
STATE_G: begin //YELLOW for B
    next_state <= STATE_E;
    if (Count >= YELLOW_Count)//green light receive time, to yellow light
       begin
           tx_DV <= 1'b1;
           Tx_Byte <= 3;
         Count <= 0;
         present_state = next_state; //go to yellow
       end
    end
STATE_H: begin //YELLOW for C
    next_state <= STATE_E;
    if (Count > YELLOW_Count)//green light receive time, to yellow light
         begin
             tx_DV <= 1'b1;
              Tx_Byte <= 4;
           Count <= 0;
           present_state = next_state; //go to yellow

         end
    end

STATE_I: begin //YELLOW for D
    next_state <= STATE_E;
    if (Count > YELLOW_Count)//green light receive time, to yellow light
       begin//////////
           tx_DV <= 1'b1;
           Tx_Byte <= 1;
         Count <= 0;
         present_state = next_state; //go to yellow
       end
    end

default: next_state <= STATE_E;
endcase
end
else if (CHECK_CTRL == 1'b0)
    begin
```

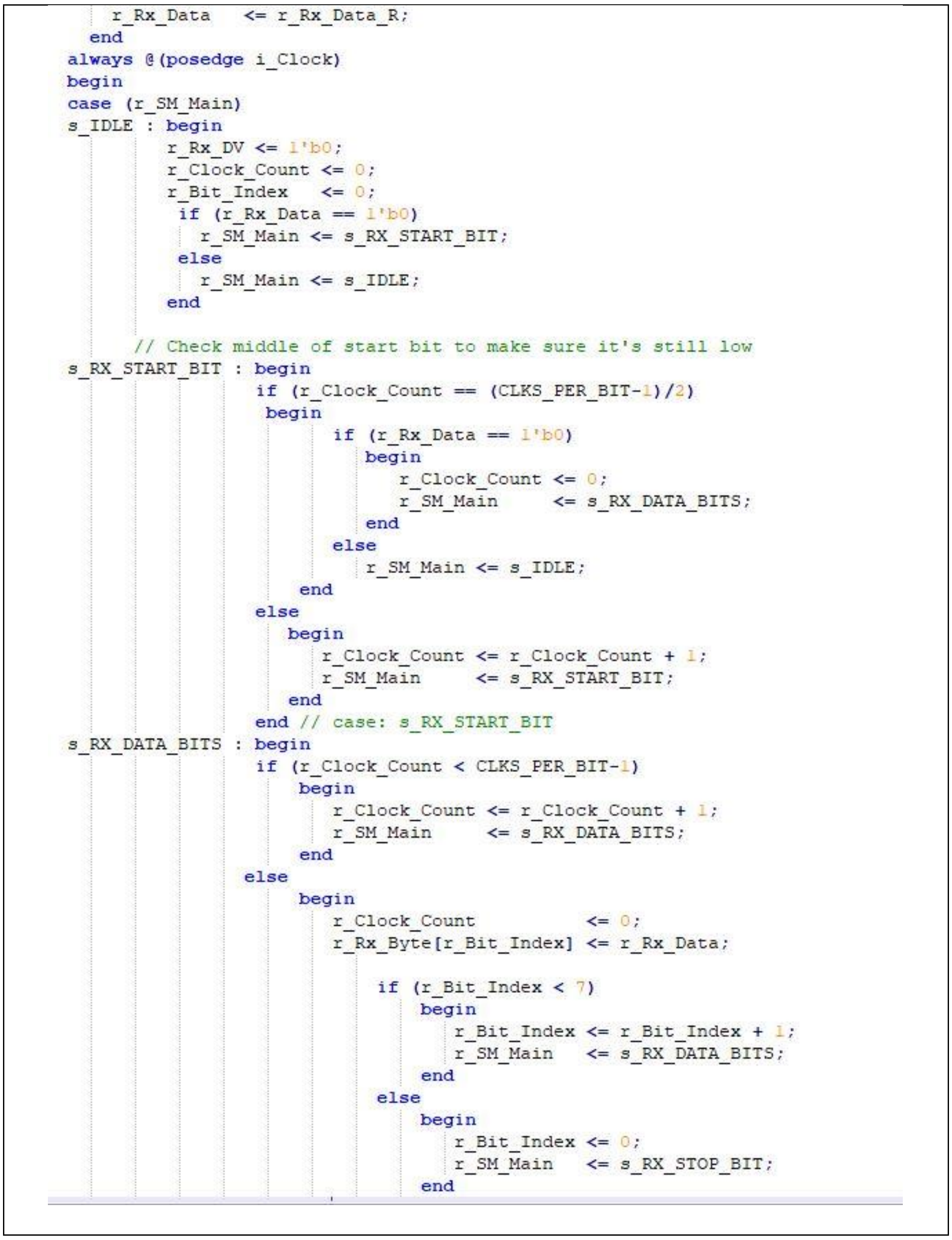


```
        begin
           tx_DV = 0;
            INPUT_time <= Rx_Byte;
        if (INPUT_time == 1)
             CHECK_CTRL = 1'b1;
        end
    end
    Display_Function_A A1(light_A,light_B,light_C,light_D,
    gpio_light,present_state);
    UART U1(clk, Rx_Serial, Tx_Byte, tx_DV, Rx_Byte, Tx_Done,
    tx_Serial, out_tx_byte);
    bin2segment1 B2S1( Count       [7:0], SegmentOut[6 :0] ,
    SegmentOut[13 :7] );
    bin2segment B2S3( out_tx_byte [3:0], SegmentOut[20:14]);
    bin2segment B2S4( out_tx_byte [7:4], SegmentOut[27:21]);
    bin2segment B2S5( Rx_Byte     [3:0], SegmentOut[34:28]);
    bin2segment B2S6( Rx_Byte     [7:4], SegmentOut[41:35]);
    bin2segment1 B2S7( INPUT_time  [7:0], SegmentOut[48:42],
    SegmentOut[55:49]);
    endmodule
```

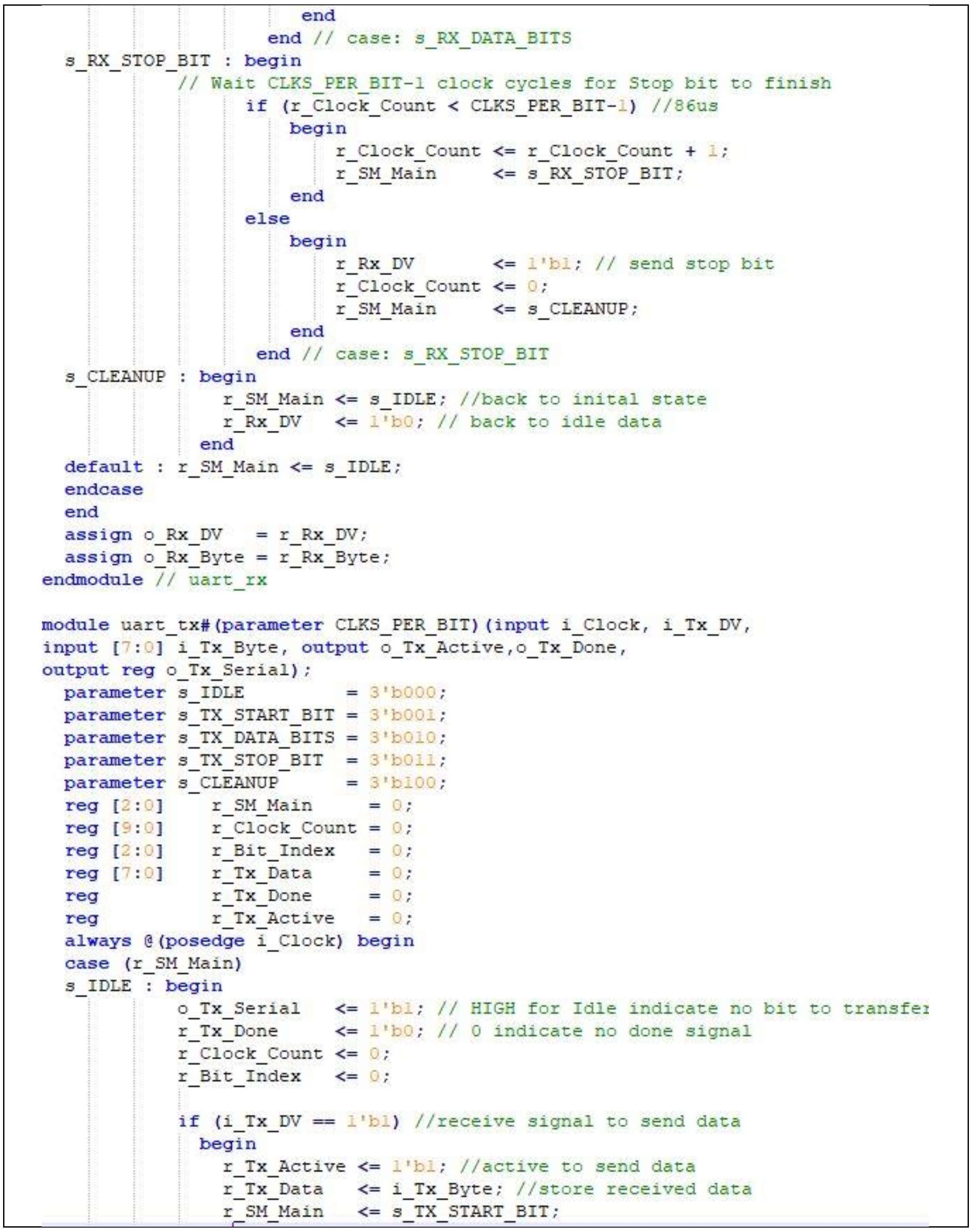
Appendix B: Source code of UART Communication on Altera DE2 board



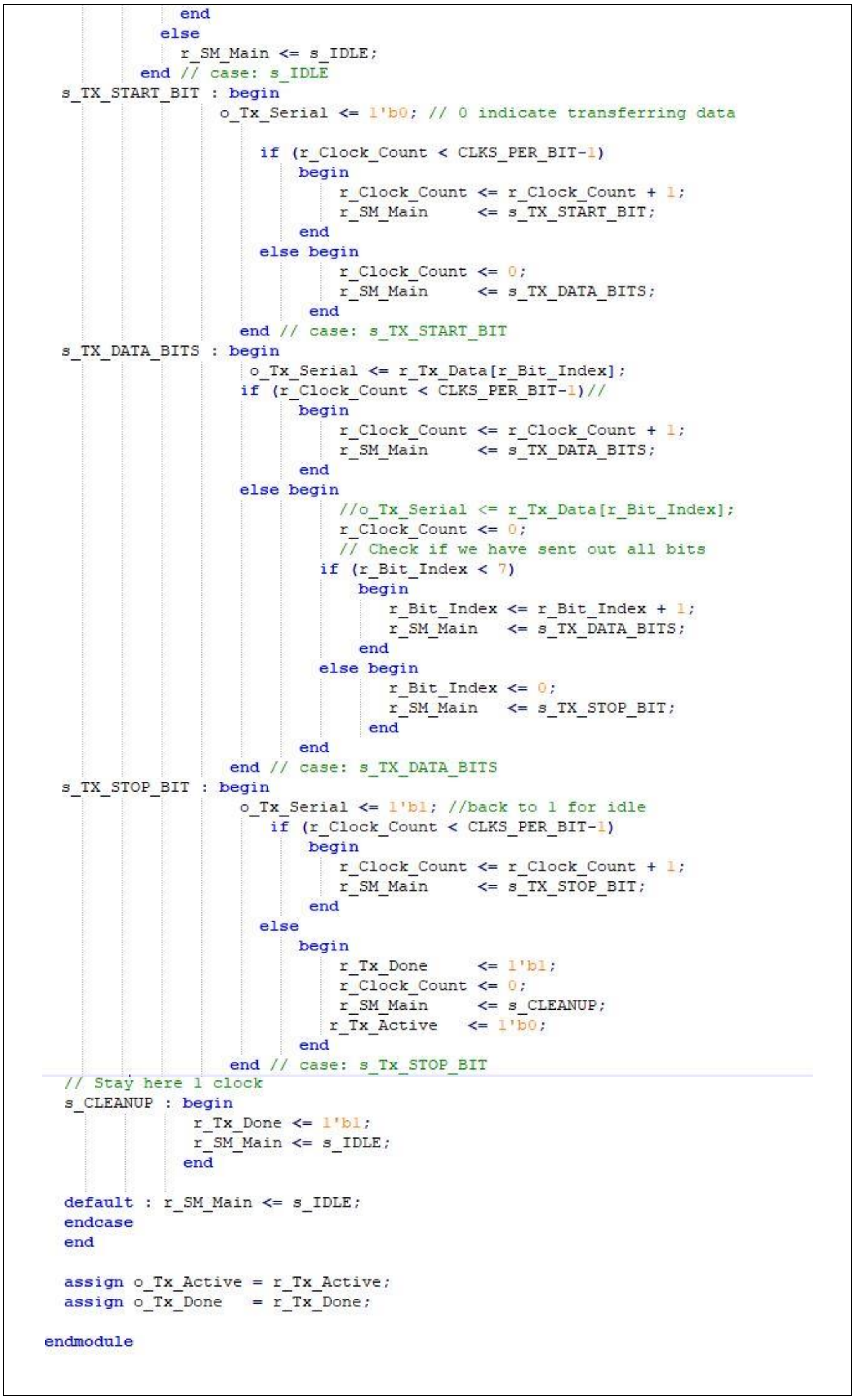
```
module UART(input r_Clock,
            input r_Rx_Serial,
                input [7:0] r_Tx_Byte,
                input r_Tx_DV,
                output wire [7:0] w_Rx_Byte,
                output reg w_Tx_Done,
                output w_tx_Serial,
                output reg[7:0] tx_byte);
  parameter c_CLOCK_PERIOD_NS = 38;
  parameter c_CLKS_PER_BIT    = 235;
  parameter c_BIT_PERIOD      = 8900;
  reg [24:0] temp = 0;
  always
  tx_byte <= r_Tx_Byte;
  always @ ( posedge r_Clock)
  begin
    if (temp >=27000000)
         begin
           temp <=0;
              w_Tx_Done <= 1'b0;
        end
    else
         begin
                w_Tx_Done <= 1'b1;
             temp <= temp +1;
                //r_Tx_DV <= 1'b0;
        end
  end

  uart_rx #(.CLKS_PER_BIT(c_CLKS_PER_BIT)) R1(.i_Clock(r_Clock),
  .i_Rx_Serial(r_Rx_Serial), .o_Rx_DV(),.o_Rx_Byte(w_Rx_Byte));
  uart_tx #(.CLKS_PER_BIT(c_CLKS_PER_BIT)) T1(.i_Clock(r_Clock),
  .i_Tx_DV(r_Tx_DV),.i_Tx_Byte(r_Tx_Byte),.o_Tx_Active(),
  .o_Tx_Serial(w_tx_Serial),.o_Tx_Done());
endmodule

module uart_rx #(parameter CLKS_PER_BIT)(input i_Clock, i_Rx_Serial,
output o_Rx_DV, output [7:0] o_Rx_Byte);
  parameter s_IDLE         = 3'b000;
  parameter s_RX_START_BIT = 3'b001;
  parameter s_RX_DATA_BITS = 3'b010;
  parameter s_RX_STOP_BIT  = 3'b011;
  parameter s_CLEANUP      = 3'b100;
  reg           r_Rx_Data_R = 1'b1;  //signal to receive data
  reg           r_Rx_Data   = 1'b1;  //signal to receive data
  reg [9:0]     r_Clock_Count = 0;
  reg [2:0]     r_Bit_Index   = 0;
  reg [7:0]     r_Rx_Byte     = 0; //variable to receive 8 bits data
  reg           r_Rx_DV       = 0;
  reg [2:0]     r_SM_Main     = 0; //variable for FSM

  always @(posedge i_Clock)
    begin
      r_Rx_Data_R <= i_Rx_Serial;
```



```
      r_Rx_Data    <= r_Rx_Data_R;
    end
  always @(posedge i_Clock)
  begin
  case (r_SM_Main)
  s_IDLE : begin
           r_Rx_DV <= 1'b0;
           r_Clock_Count <= 0;
           r_Bit_Index   <= 0;
            if (r_Rx_Data == 1'b0)
              r_SM_Main <= s_RX_START_BIT;
            else
              r_SM_Main <= s_IDLE;
           end

        // Check middle of start bit to make sure it's still low
  s_RX_START_BIT : begin
                   if (r_Clock_Count == (CLKS_PER_BIT-1)/2)
                    begin
                          if (r_Rx_Data == 1'b0)
                             begin
                                r_Clock_Count <= 0;
                                r_SM_Main     <= s_RX_DATA_BITS;
                             end
                          else
                             r_SM_Main <= s_IDLE;
                      end
                   else
                      begin
                         r_Clock_Count <= r_Clock_Count + 1;
                         r_SM_Main     <= s_RX_START_BIT;
                      end
                   end // case: s_RX_START_BIT
  s_RX_DATA_BITS : begin
                   if (r_Clock_Count < CLKS_PER_BIT-1)
                       begin
                          r_Clock_Count <= r_Clock_Count + 1;
                          r_SM_Main     <= s_RX_DATA_BITS;
                       end
                  else
                       begin
                          r_Clock_Count            <= 0;
                          r_Rx_Byte[r_Bit_Index] <= r_Rx_Data;

                              if (r_Bit_Index < 7)
                                  begin
                                     r_Bit_Index <= r_Bit_Index + 1;
                                     r_SM_Main   <= s_RX_DATA_BITS;
                                  end
                              else
                                  begin
                                     r_Bit_Index <= 0;
                                     r_SM_Main   <= s_RX_STOP_BIT;
                                  end
```

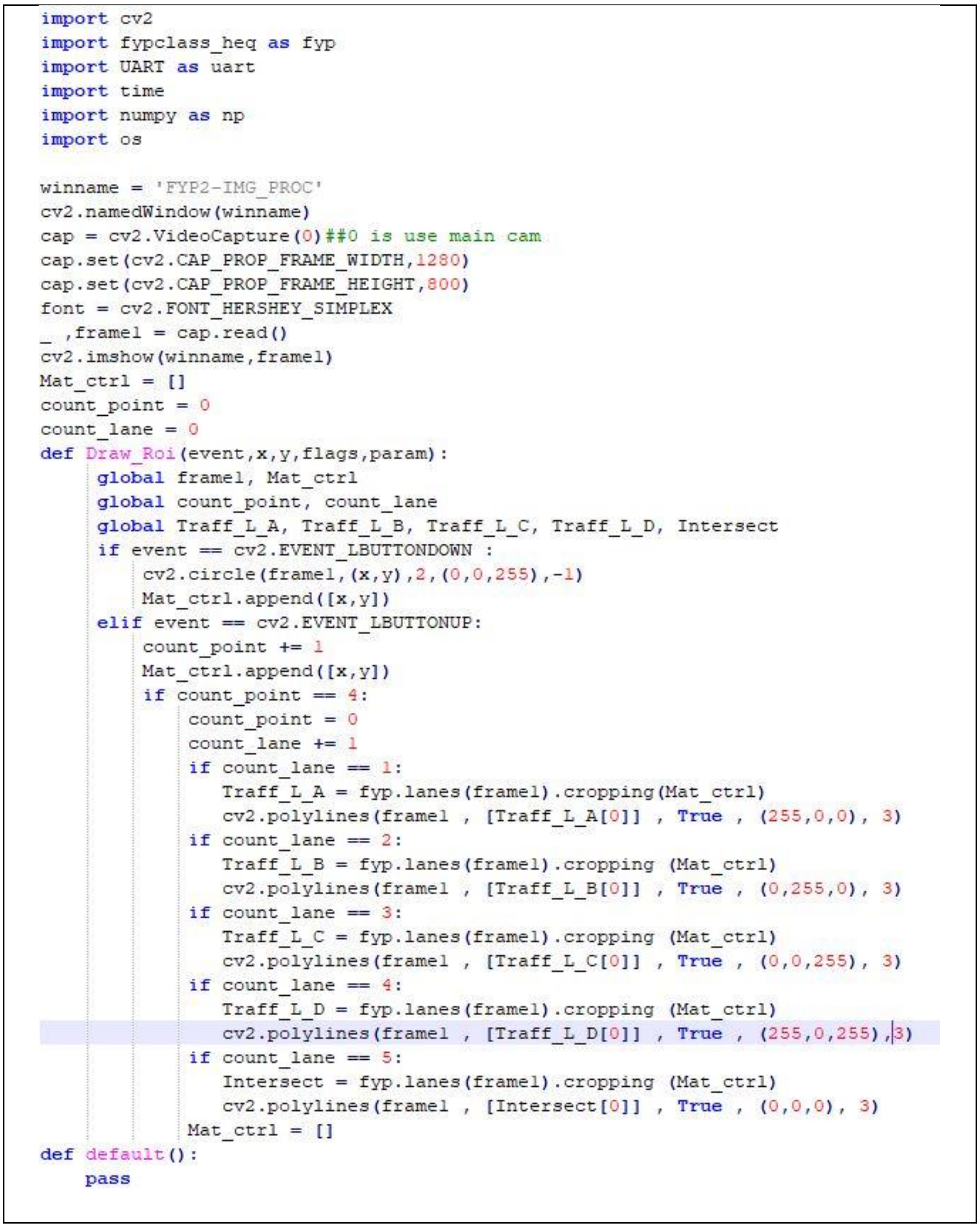


```
                    end
                end // case: s_RX_DATA_BITS
  s_RX_STOP_BIT : begin
            // Wait CLKS_PER_BIT-1 clock cycles for Stop bit to finish
                  if (r_Clock_Count < CLKS_PER_BIT-1) //86us
                      begin
                          r_Clock_Count <= r_Clock_Count + 1;
                          r_SM_Main     <= s_RX_STOP_BIT;
                      end
                  else
                      begin
                          r_Rx_DV       <= 1'b1; // send stop bit
                          r_Clock_Count <= 0;
                          r_SM_Main     <= s_CLEANUP;
                      end
               end // case: s_RX_STOP_BIT
  s_CLEANUP : begin
                r_SM_Main <= s_IDLE; //back to inital state
                r_Rx_DV   <= 1'b0; // back to idle data
              end
  default : r_SM_Main <= s_IDLE;
  endcase
  end
  assign o_Rx_DV   = r_Rx_DV;
  assign o_Rx_Byte = r_Rx_Byte;
endmodule // uart_rx

module uart_tx#(parameter CLKS_PER_BIT)(input i_Clock, i_Tx_DV,
input [7:0] i_Tx_Byte, output o_Tx_Active,o_Tx_Done,
output reg o_Tx_Serial);
  parameter s_IDLE         = 3'b000;
  parameter s_TX_START_BIT = 3'b001;
  parameter s_TX_DATA_BITS = 3'b010;
  parameter s_TX_STOP_BIT  = 3'b011;
  parameter s_CLEANUP      = 3'b100;
  reg [2:0]    r_SM_Main     = 0;
  reg [9:0]    r_Clock_Count = 0;
  reg [2:0]    r_Bit_Index   = 0;
  reg [7:0]    r_Tx_Data     = 0;
  reg          r_Tx_Done     = 0;
  reg          r_Tx_Active   = 0;
  always @(posedge i_Clock) begin
  case (r_SM_Main)
  s_IDLE : begin
          o_Tx_Serial   <= 1'b1; // HIGH for Idle indicate no bit to transfe
          r_Tx_Done     <= 1'b0; // 0 indicate no done signal
          r_Clock_Count <= 0;
          r_Bit_Index   <= 0;

          if (i_Tx_DV == 1'b1) //receive signal to send data
            begin
              r_Tx_Active <= 1'b1; //active to send data
              r_Tx_Data   <= i_Tx_Byte; //store received data
              r_SM_Main   <= s_TX_START_BIT;
```

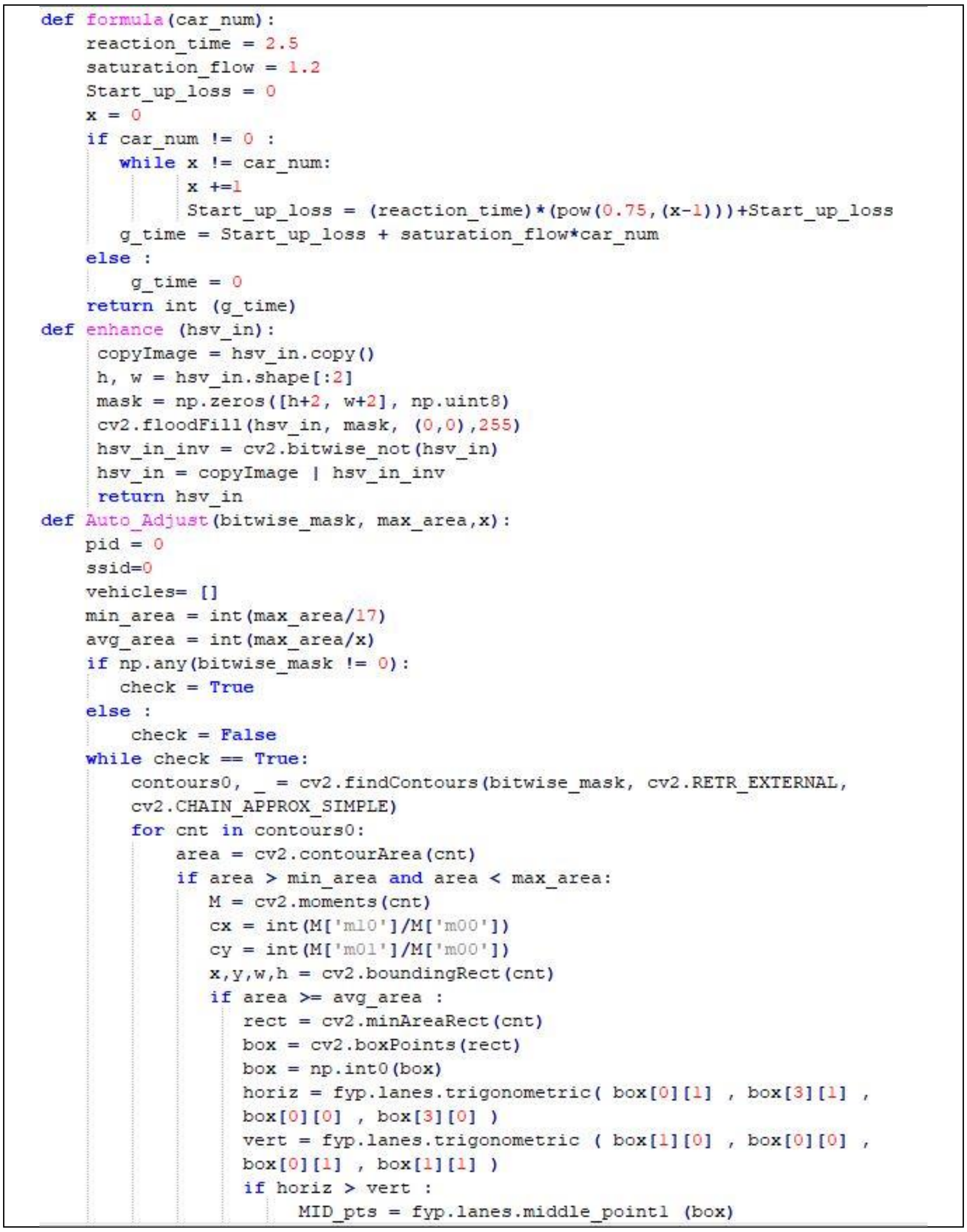


```
                   end
                 else
                   r_SM_Main <= s_IDLE;
               end // case: s_IDLE
  s_TX_START_BIT : begin
                         o_Tx_Serial <= 1'b0; // 0 indicate transferring data

                             if (r_Clock_Count < CLKS_PER_BIT-1)
                                 begin
                                     r_Clock_Count <= r_Clock_Count + 1;
                                     r_SM_Main     <= s_TX_START_BIT;
                                 end
                             else begin
                                     r_Clock_Count <= 0;
                                     r_SM_Main     <= s_TX_DATA_BITS;
                                  end
                           end // case: s_TX_START_BIT
  s_TX_DATA_BITS : begin
                            o_Tx_Serial <= r_Tx_Data[r_Bit_Index];
                           if (r_Clock_Count < CLKS_PER_BIT-1)//
                                 begin
                                     r_Clock_Count <= r_Clock_Count + 1;
                                     r_SM_Main     <= s_TX_DATA_BITS;
                                 end
                           else begin
                                     //o_Tx_Serial <= r_Tx_Data[r_Bit_Index];
                                     r_Clock_Count <= 0;
                                     // Check if we have sent out all bits
                                   if (r_Bit_Index < 7)
                                       begin
                                          r_Bit_Index <= r_Bit_Index + 1;
                                          r_SM_Main   <= s_TX_DATA_BITS;
                                       end
                                   else begin
                                          r_Bit_Index <= 0;
                                          r_SM_Main   <= s_TX_STOP_BIT;
                                        end
                                 end
                          end // case: s_TX_DATA_BITS
  s_TX_STOP_BIT  : begin
                           o_Tx_Serial <= 1'b1; //back to 1 for idle
                              if (r_Clock_Count < CLKS_PER_BIT-1)
                                  begin
                                     r_Clock_Count <= r_Clock_Count + 1;
                                     r_SM_Main     <= s_TX_STOP_BIT;
                                  end
                             else
                                 begin
                                     r_Tx_Done     <= 1'b1;
                                     r_Clock_Count <= 0;
                                     r_SM_Main     <= s_CLEANUP;
                                    r_Tx_Active    <= 1'b0;
                                 end
                          end // case: s_Tx_STOP_BIT
  // Stay here 1 clock
  s_CLEANUP : begin
                 r_Tx_Done <= 1'b1;
                 r_SM_Main <= s_IDLE;
               end

  default : r_SM_Main <= s_IDLE;
  endcase
  end

  assign o_Tx_Active = r_Tx_Active;
  assign o_Tx_Done   = r_Tx_Done;

endmodule
```

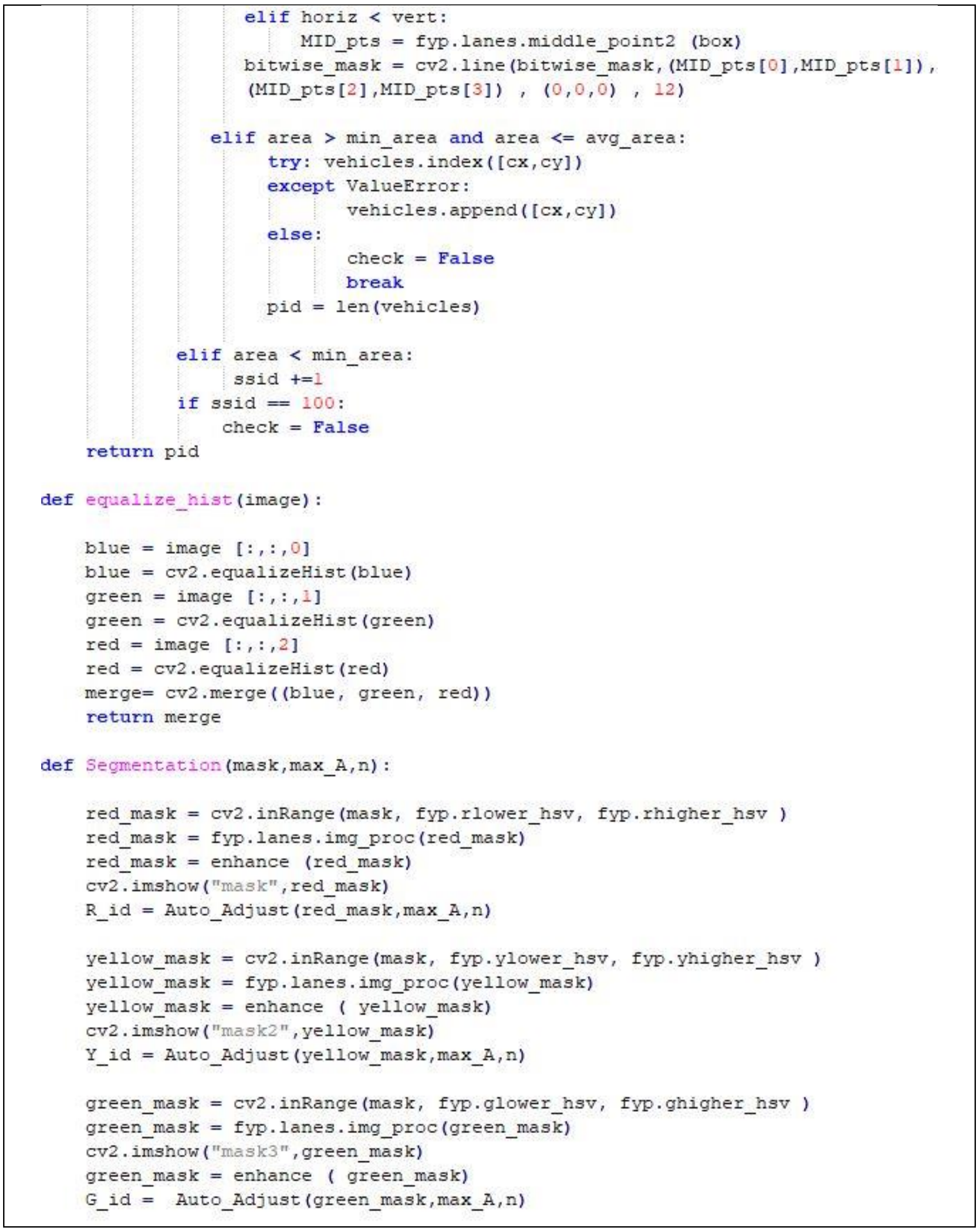
Appendix C : Source code of Image Processing Software



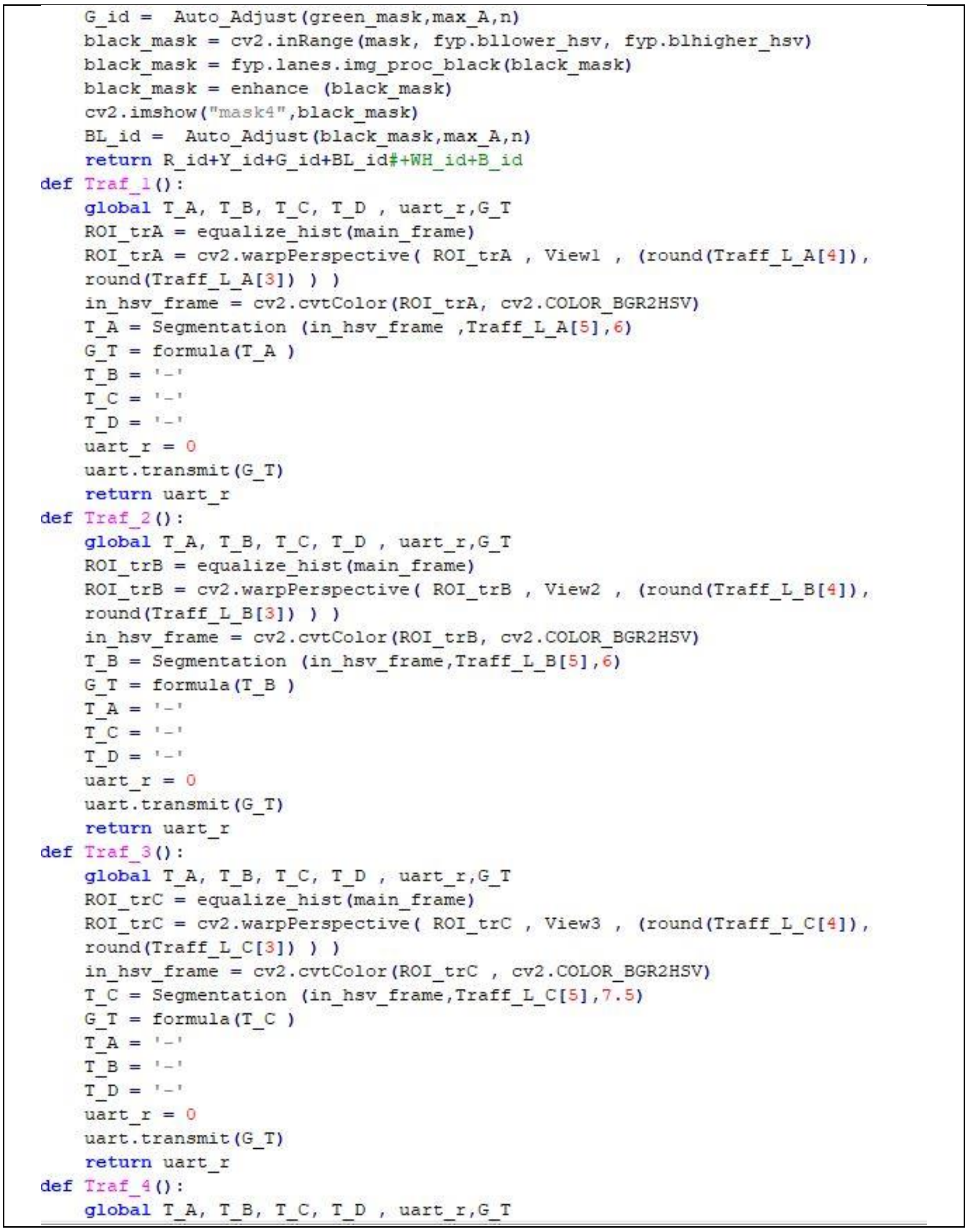
```
import cv2
import fypclass_heq as fyp
import UART as uart
import time
import numpy as np
import os

winname = 'FYP2-IMG_PROC'
cv2.namedWindow(winname)
cap = cv2.VideoCapture(0)##0 is use main cam
cap.set(cv2.CAP_PROP_FRAME_WIDTH,1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT,800)
font = cv2.FONT_HERSHEY_SIMPLEX
_ ,frame1 = cap.read()
cv2.imshow(winname,frame1)
Mat_ctrl = []
count_point = 0
count_lane = 0
def Draw_Roi(event,x,y,flags,param):
    global frame1, Mat_ctrl
    global count_point, count_lane
    global Traff_L_A, Traff_L_B, Traff_L_C, Traff_L_D, Intersect
    if event == cv2.EVENT_LBUTTONDOWN :
        cv2.circle(frame1,(x,y),2,(0,0,255),-1)
        Mat_ctrl.append([x,y])
    elif event == cv2.EVENT_LBUTTONUP:
        count_point += 1
        Mat_ctrl.append([x,y])
        if count_point == 4:
            count_point = 0
            count_lane += 1
            if count_lane == 1:
               Traff_L_A = fyp.lanes(frame1).cropping(Mat_ctrl)
               cv2.polylines(frame1 , [Traff_L_A[0]] , True , (255,0,0), 3)
            if count_lane == 2:
               Traff_L_B = fyp.lanes(frame1).cropping (Mat_ctrl)
               cv2.polylines(frame1 , [Traff_L_B[0]] , True , (0,255,0), 3)
            if count_lane == 3:
               Traff_L_C = fyp.lanes(frame1).cropping (Mat_ctrl)
               cv2.polylines(frame1 , [Traff_L_C[0]] , True , (0,0,255), 3)
            if count_lane == 4:
               Traff_L_D = fyp.lanes(frame1).cropping (Mat_ctrl)
               cv2.polylines(frame1 , [Traff_L_D[0]] , True , (255,0,255),3)
            if count_lane == 5:
               Intersect = fyp.lanes(frame1).cropping (Mat_ctrl)
               cv2.polylines(frame1 , [Intersect[0]] , True , (0,0,0), 3)
            Mat_ctrl = []
def default():
    pass
```



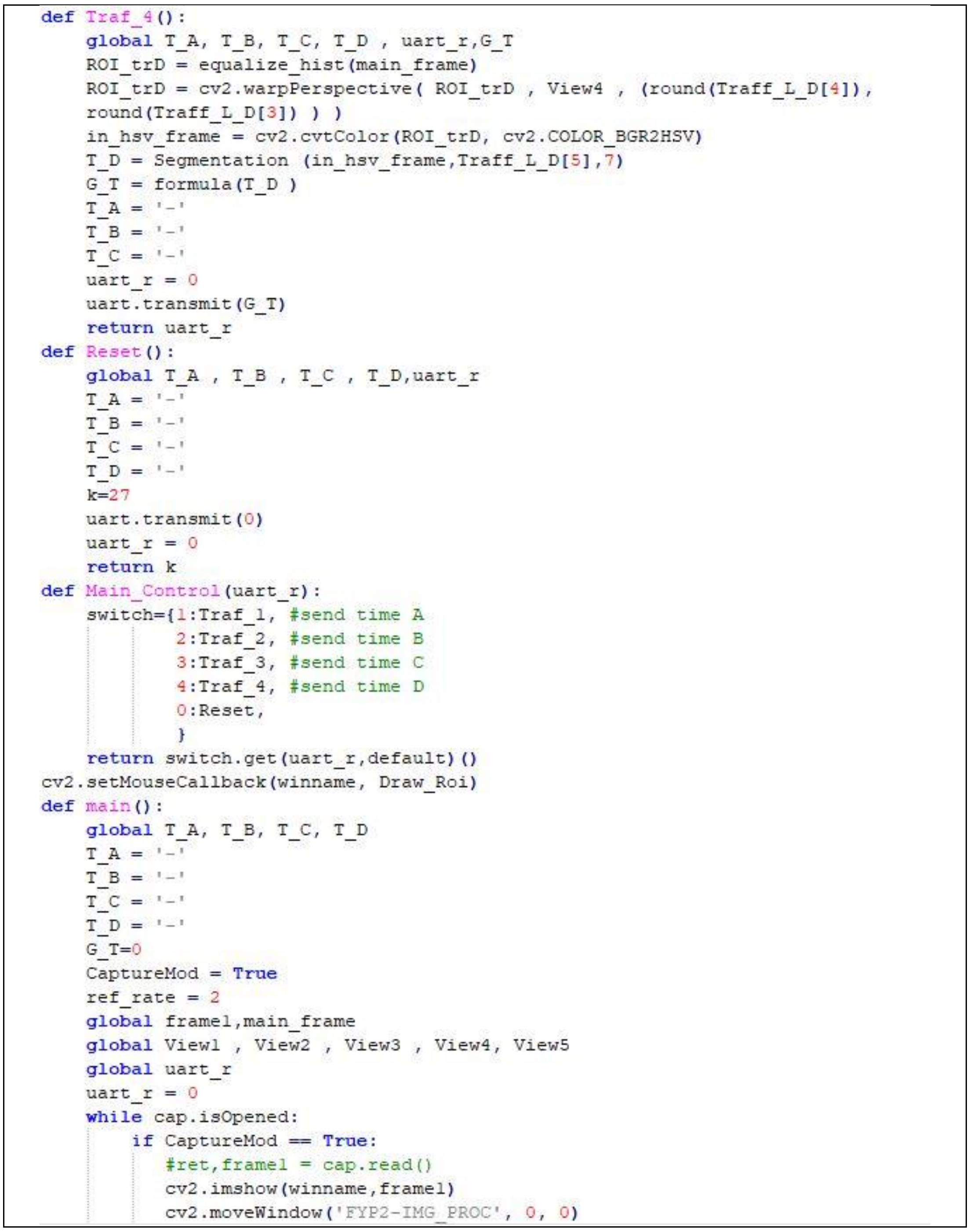
```
def formula(car_num):
    reaction_time = 2.5
    saturation_flow = 1.2
    Start_up_loss = 0
    x = 0
    if car_num != 0 :
        while x != car_num:
                x +=1
                Start_up_loss = (reaction_time)*(pow(0.75,(x-1)))+Start_up_loss
        g_time = Start_up_loss + saturation_flow*car_num
    else :
        g_time = 0
    return int (g_time)
def enhance (hsv_in):
     copyImage = hsv_in.copy()
     h, w = hsv_in.shape[:2]
     mask = np.zeros([h+2, w+2], np.uint8)
     cv2.floodFill(hsv_in, mask, (0,0),255)
     hsv_in_inv = cv2.bitwise_not(hsv_in)
     hsv_in = copyImage | hsv_in_inv
     return hsv_in
def Auto_Adjust(bitwise_mask, max_area,x):
    pid = 0
    ssid=0
    vehicles= []
    min_area = int(max_area/17)
    avg_area = int(max_area/x)
    if np.any(bitwise_mask != 0):
        check = True
    else :
        check = False
    while check == True:
        contours0, _ = cv2.findContours(bitwise_mask, cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
        for cnt in contours0:
            area = cv2.contourArea(cnt)
            if area > min_area and area < max_area:
               M = cv2.moments(cnt)
               cx = int(M['m10']/M['m00'])
               cy = int(M['m01']/M['m00'])
               x,y,w,h = cv2.boundingRect(cnt)
               if area >= avg_area :
                  rect = cv2.minAreaRect(cnt)
                  box = cv2.boxPoints(rect)
                  box = np.int0(box)
                  horiz = fyp.lanes.trigonometric( box[0][1] , box[3][1] ,
                  box[0][0] , box[3][0] )
                  vert = fyp.lanes.trigonometric ( box[1][0] , box[0][0] ,
                  box[0][1] , box[1][1] )
                  if horiz > vert :
                       MID_pts = fyp.lanes.middle_point1 (box)
```



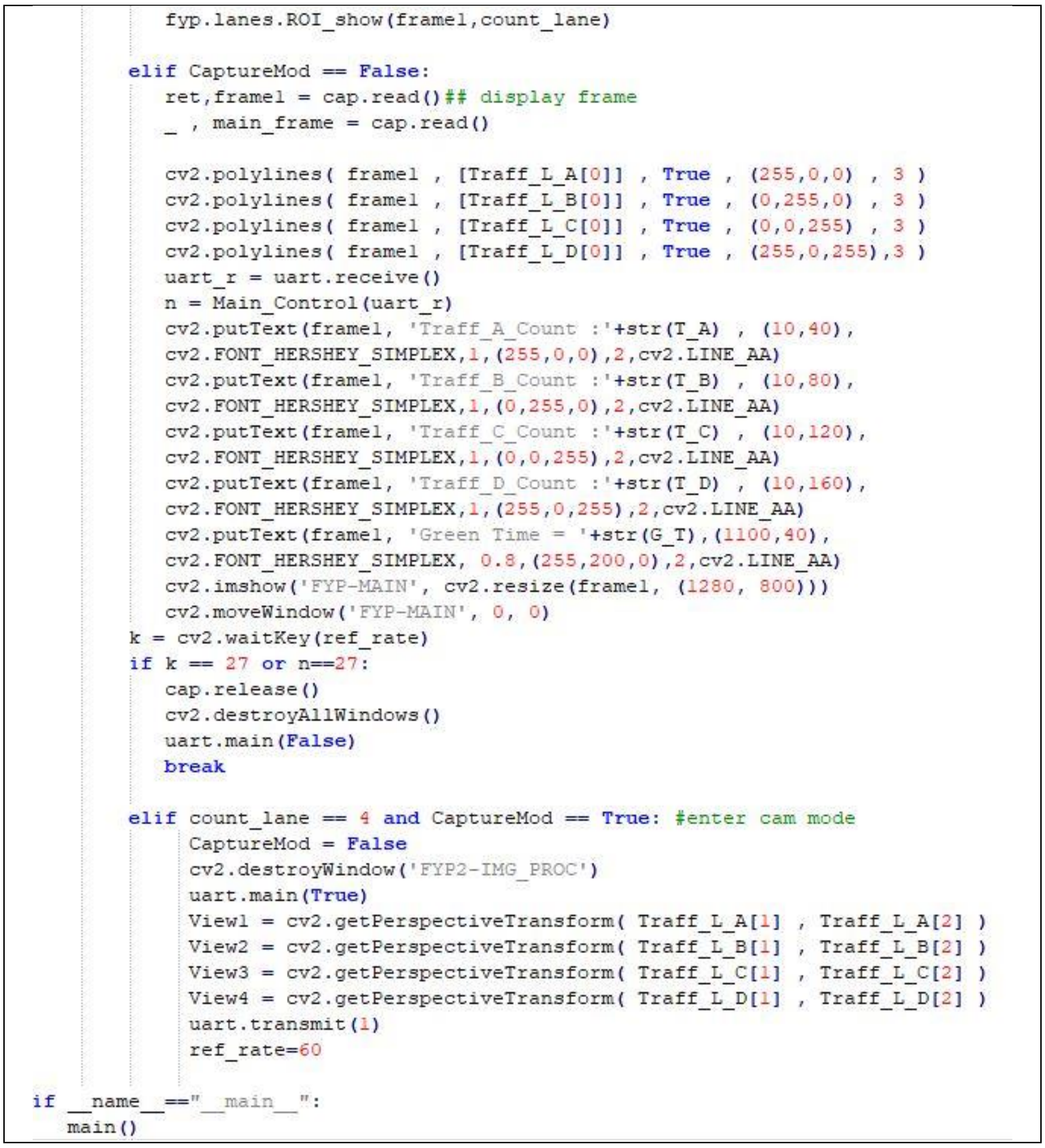
```
                  elif horiz < vert:
                       MID_pts = fyp.lanes.middle_point2 (box)
                  bitwise_mask = cv2.line(bitwise_mask,(MID_pts[0],MID_pts[1]),
                  (MID_pts[2],MID_pts[3]) , (0,0,0) , 12)

               elif area > min_area and area <= avg_area:
                    try: vehicles.index([cx,cy])
                    except ValueError:
                           vehicles.append([cx,cy])
                    else:
                           check = False
                           break
                    pid = len(vehicles)

            elif area < min_area:
                 ssid +=1
            if ssid == 100:
                check = False
    return pid

def equalize_hist(image):

    blue = image [:,:,0]
    blue = cv2.equalizeHist(blue)
    green = image [:,:,1]
    green = cv2.equalizeHist(green)
    red = image [:,:,2]
    red = cv2.equalizeHist(red)
    merge= cv2.merge((blue, green, red))
    return merge

def Segmentation(mask,max_A,n):

    red_mask = cv2.inRange(mask, fyp.rlower_hsv, fyp.rhigher_hsv )
    red_mask = fyp.lanes.img_proc(red_mask)
    red_mask = enhance (red_mask)
    cv2.imshow("mask",red_mask)
    R_id = Auto_Adjust(red_mask,max_A,n)

    yellow_mask = cv2.inRange(mask, fyp.ylower_hsv, fyp.yhigher_hsv )
    yellow_mask = fyp.lanes.img_proc(yellow_mask)
    yellow_mask = enhance ( yellow_mask)
    cv2.imshow("mask2",yellow_mask)
    Y_id = Auto_Adjust(yellow_mask,max_A,n)

    green_mask = cv2.inRange(mask, fyp.glower_hsv, fyp.ghigher_hsv )
    green_mask = fyp.lanes.img_proc(green_mask)
    cv2.imshow("mask3",green_mask)
    green_mask = enhance ( green_mask)
    G_id =  Auto_Adjust(green_mask,max_A,n)
```



```
    G_id =  Auto_Adjust(green_mask,max_A,n)
    black_mask = cv2.inRange(mask, fyp.bllower_hsv, fyp.blhigher_hsv)
    black_mask = fyp.lanes.img_proc_black(black_mask)
    black_mask = enhance (black_mask)
    cv2.imshow("mask4",black_mask)
    BL_id =  Auto_Adjust(black_mask,max_A,n)
    return R_id+Y_id+G_id+BL_id#+WH_id+B_id
def Traf_1():
    global T_A, T_B, T_C, T_D , uart_r,G_T
    ROI_trA = equalize_hist(main_frame)
    ROI_trA = cv2.warpPerspective( ROI_trA , View1 , (round(Traff_L_A[4]),
    round(Traff_L_A[3]) ) )
    in_hsv_frame = cv2.cvtColor(ROI_trA, cv2.COLOR_BGR2HSV)
    T_A = Segmentation (in_hsv_frame ,Traff_L_A[5],6)
    G_T = formula(T_A )
    T_B = '-'
    T_C = '-'
    T_D = '-'
    uart_r = 0
    uart.transmit(G_T)
    return uart_r
def Traf_2():
    global T_A, T_B, T_C, T_D , uart_r,G_T
    ROI_trB = equalize_hist(main_frame)
    ROI_trB = cv2.warpPerspective( ROI_trB , View2 , (round(Traff_L_B[4]),
    round(Traff_L_B[3]) ) )
    in_hsv_frame = cv2.cvtColor(ROI_trB, cv2.COLOR_BGR2HSV)
    T_B = Segmentation (in_hsv_frame,Traff_L_B[5],6)
    G_T = formula(T_B )
    T_A = '-'
    T_C = '-'
    T_D = '-'
    uart_r = 0
    uart.transmit(G_T)
    return uart_r
def Traf_3():
    global T_A, T_B, T_C, T_D , uart_r,G_T
    ROI_trC = equalize_hist(main_frame)
    ROI_trC = cv2.warpPerspective( ROI_trC , View3 , (round(Traff_L_C[4]),
    round(Traff_L_C[3]) ) )
    in_hsv_frame = cv2.cvtColor(ROI_trC , cv2.COLOR_BGR2HSV)
    T_C = Segmentation (in_hsv_frame,Traff_L_C[5],7.5)
    G_T = formula(T_C )
    T_A = '-'
    T_B = '-'
    T_D = '-'
    uart_r = 0
    uart.transmit(G_T)
    return uart_r
def Traf_4():
    global T_A, T_B, T_C, T_D , uart_r,G_T
```

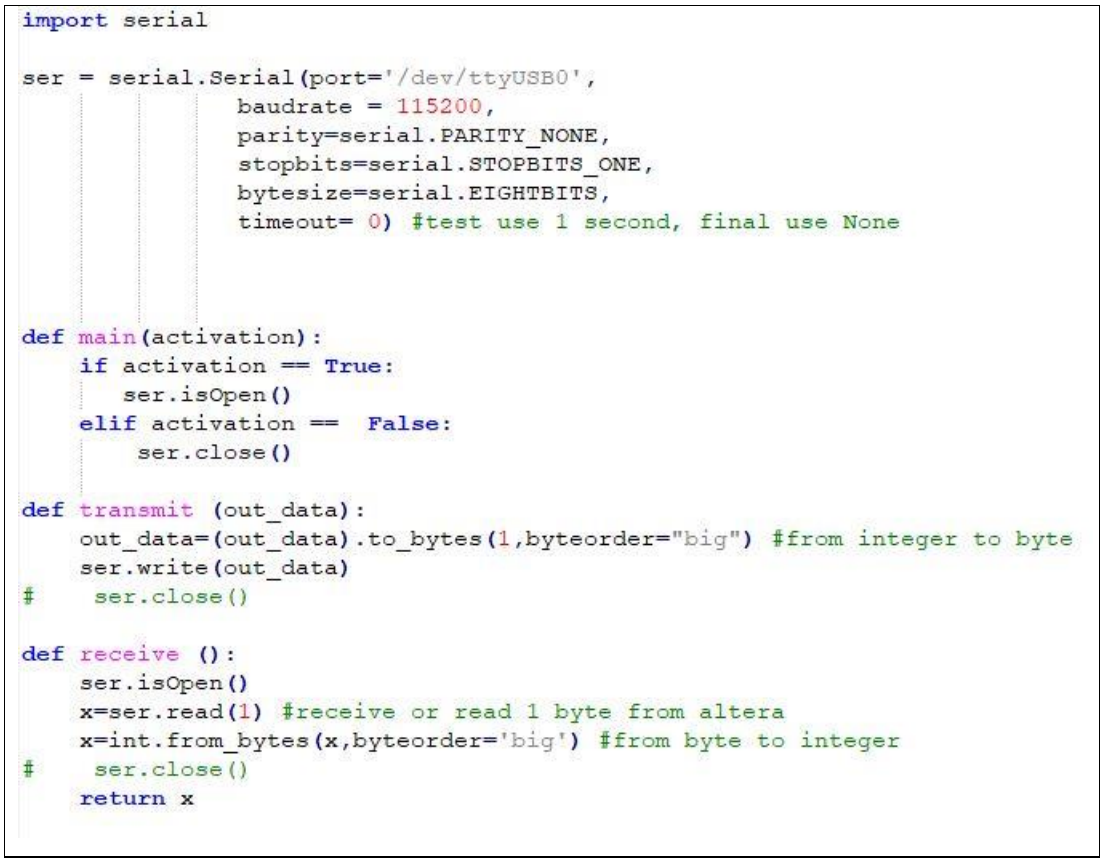


```
def Traf_4():
    global T_A, T_B, T_C, T_D , uart_r,G_T
    ROI_trD = equalize_hist(main_frame)
    ROI_trD = cv2.warpPerspective( ROI_trD , View4 , (round(Traff_L_D[4]),
    round(Traff_L_D[3]) ) )
    in_hsv_frame = cv2.cvtColor(ROI_trD, cv2.COLOR_BGR2HSV)
    T_D = Segmentation (in_hsv_frame,Traff_L_D[5],7)
    G_T = formula(T_D )
    T_A = '-'
    T_B = '-'
    T_C = '-'
    uart_r = 0
    uart.transmit(G_T)
    return uart_r
def Reset():
    global T_A , T_B , T_C , T_D,uart_r
    T_A = '-'
    T_B = '-'
    T_C = '-'
    T_D = '-'
    k=27
    uart.transmit(0)
    uart_r = 0
    return k
def Main_Control(uart_r):
    switch={1:Traf_1, #send time A
            2:Traf_2, #send time B
            3:Traf_3, #send time C
            4:Traf_4, #send time D
            0:Reset,
            }
    return switch.get(uart_r,default)()
cv2.setMouseCallback(winname, Draw_Roi)
def main():
    global T_A, T_B, T_C, T_D
    T_A = '-'
    T_B = '-'
    T_C = '-'
    T_D = '-'
    G_T=0
    CaptureMod = True
    ref_rate = 2
    global frame1,main_frame
    global View1 , View2 , View3 , View4, View5
    global uart_r
    uart_r = 0
    while cap.isOpened:
        if CaptureMod == True:
           #ret,frame1 = cap.read()
           cv2.imshow(winname,frame1)
           cv2.moveWindow('FYP2-IMG_PROC', 0, 0)
```



```
            fyp.lanes.ROI_show(frame1,count_lane)

        elif CaptureMod == False:
            ret,frame1 = cap.read()## display frame
            _ , main_frame = cap.read()

            cv2.polylines( frame1 , [Traff_L_A[0]] , True , (255,0,0) , 3 )
            cv2.polylines( frame1 , [Traff_L_B[0]] , True , (0,255,0) , 3 )
            cv2.polylines( frame1 , [Traff_L_C[0]] , True , (0,0,255) , 3 )
            cv2.polylines( frame1 , [Traff_L_D[0]] , True , (255,0,255),3 )
            uart_r = uart.receive()
            n = Main_Control(uart_r)
            cv2.putText(frame1, 'Traff_A_Count :'+str(T_A) , (10,40),
            cv2.FONT_HERSHEY_SIMPLEX,1,(255,0,0),2,cv2.LINE_AA)
            cv2.putText(frame1, 'Traff_B_Count :'+str(T_B) , (10,80),
            cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2,cv2.LINE_AA)
            cv2.putText(frame1, 'Traff_C_Count :'+str(T_C) , (10,120),
            cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2,cv2.LINE_AA)
            cv2.putText(frame1, 'Traff_D_Count :'+str(T_D) , (10,160),
            cv2.FONT_HERSHEY_SIMPLEX,1,(255,0,255),2,cv2.LINE_AA)
            cv2.putText(frame1, 'Green Time = '+str(G_T),(1100,40),
            cv2.FONT_HERSHEY_SIMPLEX, 0.8,(255,200,0),2,cv2.LINE_AA)
            cv2.imshow('FYP-MAIN', cv2.resize(frame1, (1280, 800)))
            cv2.moveWindow('FYP-MAIN', 0, 0)
        k = cv2.waitKey(ref_rate)
        if k == 27 or n==27:
            cap.release()
            cv2.destroyAllWindows()
            uart.main(False)
            break

        elif count_lane == 4 and CaptureMod == True: #enter cam mode
              CaptureMod = False
              cv2.destroyWindow('FYP2-IMG_PROC')
              uart.main(True)
              View1 = cv2.getPerspectiveTransform( Traff_L_A[1] , Traff_L_A[2] )
              View2 = cv2.getPerspectiveTransform( Traff_L_B[1] , Traff_L_B[2] )
              View3 = cv2.getPerspectiveTransform( Traff_L_C[1] , Traff_L_C[2] )
              View4 = cv2.getPerspectiveTransform( Traff_L_D[1] , Traff_L_D[2] )
              uart.transmit(1)
              ref_rate=60

if __name__=="__main__":
    main()
```

Appendix D: Source code of UART Communication on Raspberry Pi 3B+



```
import serial

ser = serial.Serial(port='/dev/ttyUSB0',
                    baudrate = 115200,
                    parity=serial.PARITY_NONE,
                    stopbits=serial.STOPBITS_ONE,
                    bytesize=serial.EIGHTBITS,
                    timeout= 0) #test use 1 second, final use None



def main(activation):
    if activation == True:
       ser.isOpen()
    elif activation ==  False:
        ser.close()

def transmit (out_data):
    out_data=(out_data).to_bytes(1,byteorder="big") #from integer to byte
    ser.write(out_data)
#     ser.close()

def receive ():
    ser.isOpen()
    x=ser.read(1) #receive or read 1 byte from altera
    x=int.from_bytes(x,byteorder='big') #from byte to integer
#     ser.close()
    return x
```