Course Name: Engineering Software 3
Assessment2 —— Traffic Light Report
Keyi Liu s1703084

Part 1:Introduction

This Project aims to design and implement the traffic lights of the T-junction road based on the BASYS-3 board. The simulated Traffic Lights follow by British Standards. I use LEDs and VGA display to indicate the different states in the sequence of traffic lights, and the 7-segment display on the Board can monitor the time of each state.

Part 2:Instructions

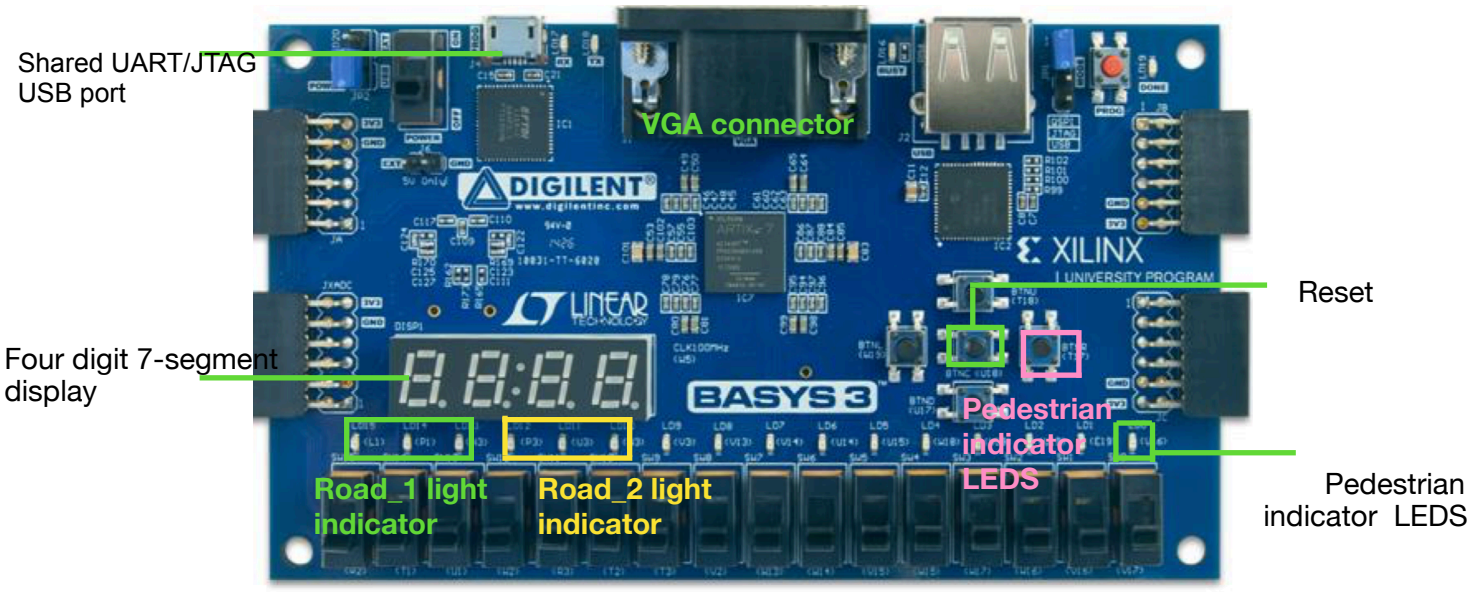

Figure 1: Devices Used For The Design

Region 0 Road-1 RED Light	**Region 3**	**Region 6** Road-2 RED Light	**Region 9**
Region 1 Road-1 YELLOW Light	**Region 4** Pedestrian Light —— RED or GREEN Light	**Region 7** Road-2 YELLOW Light	**Region 10**
Region 2 Road-1 GREEN Light	**Region 5**	**Region 8** Road-2 GREEN Light	**Region 11**

Figure 2: VGA region selection on display

Explanation for Devices used (See Figure1):

- ✦LEDS{L1 P1 N3}: represent what color of the traffic light is displaying on road 1 at current state, which indicates red, yellow and green respectively.
- ✦LEDS{P3 U3 W3}: represent what color of the traffic light is displaying on road 2 at current state, which indicates red, yellow and green respectively.
- ✦LED U16: illuminated if the pedestrian button is pressed, will go off when pedestrians are allowed to cross.
- ✦BTNR: simulates the pedestrian button in real life.
- ✦BTNC: reserved for "Reset".
- ✦Four digital 7-segment display: shows the time in the way of countdown.

Steps to execute the application

I. Turning on the FPGA(BASYS 3), connecting it to PC with the Micro-USB
II. Opening the **Vivado** —> **Export Hardware** (Includes bitstream) —> **Launch SDK**.
III. Connecting board to the monitor with the **VGA cable**, and select alternate monitor display source.
IV. **Program FPGA** to get connection, then the VGA Regions (12 empty white squares) will be displayed on monitor (see Figure 2).
   - ✦Region 0–2: represent the traffic lights: RED, AMBER, GREEN respectively on road 1;
   - ✦Region 4: represents the pedestrian light, either RED or GREEN.
   - ✦Region 6–8: represent the traffic lights: RED, AMBER, GREEN respectively on road 2;
V. Set up **Run Configuration** and run the application.

Traffic lights' operations

The Traffic Lights of T_junction (two roads) is followed by the sequence described below (See Figure 3), starting from Number-1 state to Number-8 state and then go back to Number-1 state as the end, this sequence is known as basic operations. the time duration of each state is 3 seconds.

If the pedestrian button is pressed during this sequence at any states, the RED light of pedestrian must continue to be held until the sequence finish the last state. Then the GREEN light will replace the RED on region 4, the 7-segment will countdown from 6, moreover, at the last remaining 2 seconds, the Green light will blink 5 times to give warning to the people. The basic operations will be cycling after the pedestrian state.

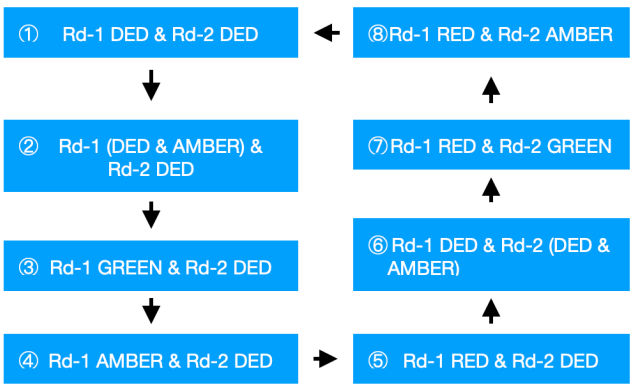

Figure 3: Basic Operations

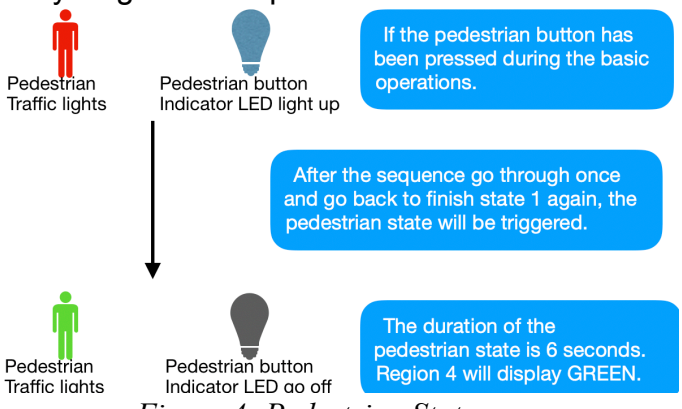

Figure 4: Pedestrian State

Part 3: Programmer's Guide

Code Structure

The core codes includes int main( ) and void hwTimerISR(void *CallbackRef) in the traffic light.c file. In the main function, to describe basic operations (See Figure 3) and Pedestrian state (See Figure 4), I used the Switch-Case, including what kind of colour will be displayed on each VGA region and how the status of the LEDs will be demonstrated. And ISR is used as a controller, to define the duration of each state and refresh the countdown timer while finishing one state. The most important thing is to define the order to execute different states which I only declare different scenarios in main function.

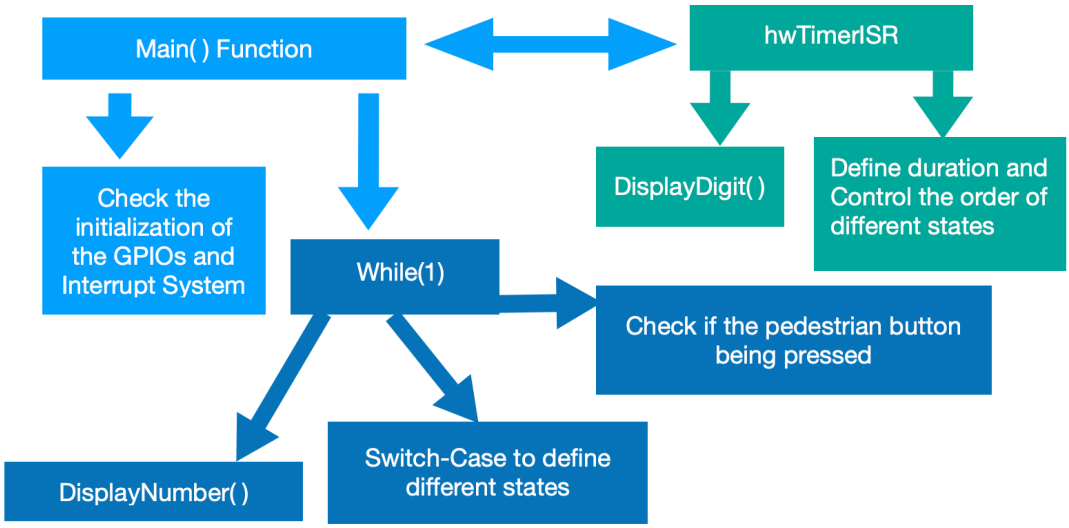

Figure 5: Design Flow Chart

Files included in the src folder of the project

— **definition.h**: definitions of constants are used for this project.

— **seg7_display.h**: definitions of 7-segment BCD codes and digit selection codes.

— **gpio_init.h**: declaration of all the devices on the FPGA and REGION_COLOUR for VGA display will be used for this project.

— **platform.h**: containing functions of init_platform() and cleanup_platform().

— **platform_config.h**: definition of hardware configuration.

— **platform.c**: functions of init_platform() and cleanup_platform().

— **gpio_init.c**: initialization of all the variables which are declared in gpio_init.h.

— **seg7_display.c**: functions of **displayNumber()** and **displayDigits()**.

— **xinterruptES3.c**: a design example using the Interrupt Controller both with a PowerPC and MicroBlaze processor, including **setUpInterruptSystem()** function.

Short explanation for hardware implementation

The VGA driver has already been implemented by using the Verilog HDL to describe hardware. The pixel location of 12 square regions has been written in module named **VGAControl**, the 12-bit colour_in signal of the each region are defined in same module and this can be controlled from software by using the **XGpio** object VGA_COLOUR.

●*Declaration and initialization of variables*

✦CONSTANTS DEFINITION

In the **definition.h** file declare and initialize the colour constants to represent the 3 lights in the traffic lights which are RED, AMBER, GREEN. But using YELLOW to represent AMBER light in this project. The aim to define WHITE here is used to recover the VGA Region in the main( ) function

```
#ifndef __DEFINITION_H_
#define __DEFINITION_H_

#define RED 0xf00      // represent the light RED
#define YELLOW 0xff0   // represent the light AMBER
#define GREEN 0x0f0    // represent the light GREEN
#define WHITE 0xfff    // define the colour of background, aim to recover the VGA region later

#endif
```

Figure 6: Constants Definition

Because the 12-bit colour_in signal has been assigned to each region in the Hardware implementation. The first 4 bits represent red, the middle 4 bits represent green, the last 4 bits represent blue. Then define RED as 0xf00 where 'f' means 1111 of the first 4 bits of the 12-bit colour_in signal. Defining as 0x0f0 means the middle 4 bits has been assigned to logic 1 to represent GREEN. Mixing up the RED and GREEN to get YELLOW therefore defines 0xff0. However, colour WHITE is achieved by mixing all three colours and the VGA region's default colour is WHITE.

✦Devices Declaration and initialization

All the devices used on the FPGA and VGA_REGION which are implemented in VGA hardware driver should be declared in the file gpio_init.h.

— XGpio LED_OUT: all the 16 LEDs (from L1 to V16) on the BASYS-3.
— XGpio SEG_HEX_OUT: to write a binary-coded decimal (BCD) to any digit.
— XGpio SEG_SEL_OUT: select corresponding digit.
— XGpio P_BTN_RIGHT: the right press button will be used as pedestrian button.
— XGpio REGION_{0:11}_COLOUR: write colour to the corresponding regions.

once these variables have been declared, related initialization needs to be achieved inside the Xstatus initGpio(void) of the file gpio_init.c. an example code shows below (See Figure7), all the device initialization share the similar format, the only difference is to find the unique device name and deviceID. The deviceID can be checked in xparameters.h.

```
status = XGpio_Initialize(&P_BTN_RIGHT, 5);
if (status != XST_SUCCESS)
{
    return XST_FAILURE;
}


status = XGpio_Initialize(&REGION_0_COLOUR, 7);
if (status != XST_SUCCESS)
{
    return XST_FAILURE;
}
```

Figure 7: Initialization of the Device and VGA_COLOUR

✦Variables in trafficLight.c file

InterruptCount is global variable modified by an interrupt service Routine (ISR). The reason I use volatile to declare is that the value of this variable could change unexpectedly. The hardware timer is set to give interrupts every 0.004s which means the interruptCount will increase 250 per second. InterruptCount here is used to refresh the number shown on 7-segment display, the refresh rate is too fast to let human beings notice. Other variables I use explained in the below (See Figure 8)

```
volatile int interruptCount=0; //initialise the interruptCount, giving the interrupt counter to 0 at first
u16 wait=FALSE; // flag to show if the pedestrian button has been pressed or not, giving it to False to represent not being pressed at first.
u16 time = 3;   // declare time, giving it to 3s represents duration of different states in Basic operations
u16 state = 0;  // declare state is used as variable in switch-case later, give state = 0 at first;
u16 blink = 0x0000; // declare the blink and initialise to 0x0000, aim to use bitwise OR to compare with GREEN
```

Figure 8 : Variables Defined in trafficLight.c file

●*int main() Functions*

❖The main function consists of two sections (See Figure 5): One is to check the initialization of the GPIOs and Interrupt by using the fixed format shows below.

```
    status = initGpio();// initialise the GPIOs
/***Check if the general-purpose IOs in the hardware are successful initialised. If not, the program is terminated.
    if (status != XST_SUCCESS) {
        print("GPIOs initialisation failed!\n\r");
        cleanup_platform();
        return 0;
    }
    status = setUpInterruptSystem(); // Setup the Interrupt System
/***Check if the interrupt system has been set up. If not, the program is terminated.******/
    if (status != XST_SUCCESS) {
        print("Interrupt system setup failed!\n\r");
        cleanup_platform();
        return 0;
    }
```

Figure 9 :Check GPIOS and Interrupt System

❖The other section is while(1), using while(1) to execute the codes inside this loop forever to ensure the traffic lights of T-junction roads can be always working and the 7-segment display never stop demonstrating time. From the design flow chart (See Figure 5), there are 3 parts inside while(1) loop:

— if (XGpio_DiscreteRead(&P_BTN_RIGHT, 1)) wait = TRUE;

When the pedestrian button (BTNR on the BASYS-3 board) is pressed, the variable ***wait*** will change from FALSE to TRUE. Using the boolean value of ***wait*** to decide if the indicator LED should be light up and determine the next state after one sequence cycle of traffic lights finish.

— Switch-case

Defines the different scenarios of the two traffic lights and pedestrian light will be analyzed further later.

—DisplayNumber(time);

Demonstrates the time on the 7-segment display. I place this line of code inside the while(1) in order to ensure the countdown timer never stops when running this project application.

Switch-Case

Using switch-case to represent different states is more clear than using if-statement. The structure of the switch-case follows switch( ) { different cases, default }. The sequence cycle of the Traffic lights which is defined in different cases follows the design flow shown in Figure 3. In each case, XGpio_DiscreteWrite( ) is used to assign colour of the VGA regions (Only region 0-2, region 4 and region 6-8 are used in this project). Moreover, how the LEDs demonstrate the traffic light sequence is separately defined in the different cases. If pedestrian button is triggered, ***wait*** = TRUE =1, LED V16 will be illuminated. An example code shows below in Figure 10 and specific implementation summary of each case shows in Figure 11.

```
switch(state){

    case 0:
        XGpio_DiscreteWrite(&REGION_0_COLOUR, 1, RED);
        XGpio_DiscreteWrite(&REGION_6_COLOUR, 1, RED);
        XGpio_DiscreteWrite(&REGION_4_COLOUR, 1, RED);

        XGpio_DiscreteWrite(&REGION_1_COLOUR, 1, WHITE);
        XGpio_DiscreteWrite(&REGION_2_COLOUR, 1, WHITE);
        XGpio_DiscreteWrite(&REGION_7_COLOUR, 1, WHITE);
        XGpio_DiscreteWrite(&REGION_8_COLOUR, 1, WHITE);

        XGpio_DiscreteWrite(&LED_OUT, 1, 0x9000 + wait);
        break;
```

Figure 10 : Switch_Case code example

Case Name	State Description (RED — R, YELLOW—Y, GREEN— G)	Working Regions	Non-working regions (white)	LEDs
0	Rd1 ***R*** & Rd2 ***R*** & PD ***R***	Region [0, 4, 6] RED	Region[1,2,7,8]	L1,P3 lighting + wait
1	Rd1 ***R*** & Rd1 ***Y*** & Rd2 ***R*** & PD ***R***	Region [0, 4, 6] RED Region_1 YELLOW	Region[2,7,8]	L1, P1,P3 lighting + wait
2	Rd1 ***G*** & Rd2 ***R*** & PD ***R***	Region [4, 6] RED Region_2 GREEN	Region[0,1,7,8]	N3, P3 lighting + wait
3	Rd1 ***Y*** & Rd2 ***R*** & PD ***R***	Region [4, 6] RED Region_1 YELLOW	Region[0,2,7,8]	P1, P3 lighting + wait
4	Rd1 ***R*** & Rd2 ***R*** & PD ***R***	Region [0, 4, 6] RED	Region[1,2,7,8]	L1,P3 lighting + wait
5	Rd2 ***R*** & Rd2 ***Y*** & Rd1 ***R*** & PD ***R***	Region [0, 4, 6] RED Region_7 YELLOW	Region[1,2,8]	L1,P3,U3 lighting + wait
6	Rd2 ***G*** & Rd1 ***R*** & PD ***R***	Region [0, 4] RED Region_8 GREEN	Region[1,2,6,7]	L1,W3 lighting + wait
7	Rd2 ***Y*** & Rd1 ***R*** & PD ***R***	Region [0, 4] RED Region_7 YELLOW	Region[1,2,6,8]	L1,U3 lighting + wait
8	Rd1 ***R*** & Rd2 ***R*** & PD ***R***	Region [0, 4, 6] RED	Region[1,2,7,8]	L1,P3 lighting + wait
9	Rd1 ***R*** & Rd2 ***R*** & PD ***G***	Region [0, 6] RED Region_4 GREEN	Region[1,2,7,8]	L1,P3 lighting

Figure 11 : Switch_Case Implementation Summary

ISR function

Interrupt Service Routine is a software process invoked by an interrupt request from a hardware timer. The ISR function is executed in parallel with main function and worked as an traffic lights controller to refresh the duration of each state, keep the traffic light sequence cycle to work properly and how the duration and state change when pedestrian button is being triggered. The specific design flow shows below (See Figure 12), and also the codes with explanation shows in Figure 13.

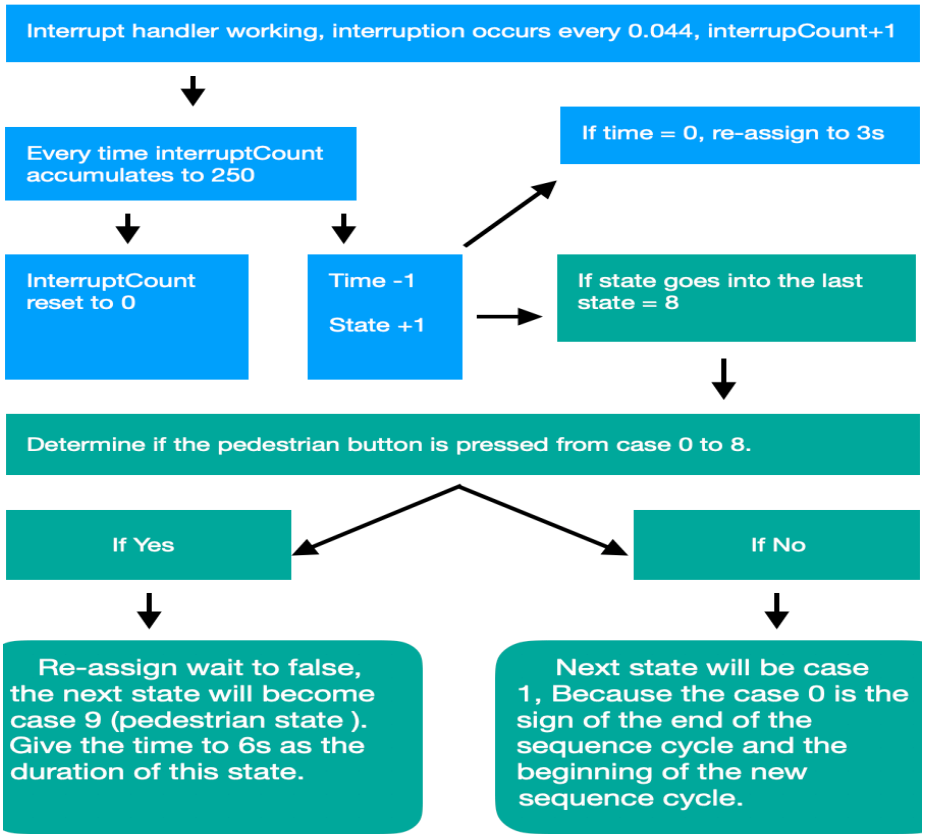

Figure 12 : ISR Design Flow Chart

```
void hwTimerISR(void *CallbackRef){
    displayDigit();
    interruptCount++; // 0.004 interruptCount++
    if(interruptCount == 250){
        //Every second, to reset the interruptCount and decrement time
        time--;
        interruptCount = 0;
        if(time == 0)//after 3 seconds
        {
            time = 3;//reassign the time to 3
            if(state == 8)// when the sequence cycle reach 8
            {
                if(wait == FALSE) //check if the pedestrian button being pressed
                    state = 0;    // if not, restart sequence cycle
                else
                {   // if the pedestrian button being pressed during state 0-8
                    wait = FALSE;
                    time = 6;// re-assign wait to False and give the duration of pedestrian state to 6s
                }
            }
            state++;//every time time=0, the state+1
        }
    }
```

Figure 13 : ISR Codes With Explanation

Pedestrian light blink

We are required to achieve the GREEN light blinks 5 times in the last remaining 2 seconds in the pedestrian state, 2 seconds can be regarded as 500 interruptions executed in total, using the knowledge of bitwise-OR to decide either WHITE or GREEN should be displayed. The **blink** is initialized as 0x000 in the beginning, when the interruptCount accumulates to 50, **blink** = ~ **blink**, **blink** is assigned by bitwise-NOT as 0xfff. At this moment the colour WHITE is assigned to region 4 because bitwise-OR (GREEN | **blink**) = (0x0f0 | 0xfff) = 0xfff. When the interruptCount accumulates to 100 which is the next 50 interruptions. **blink** = ~ **blink**, blink is assigned by bitwise-NOT as 0x000. At this moment the colour GREEN is assigned to region 4 because bitwise-OR (GREEN | **blink**) = (0x0f0 | 0x000) = 0x0f0. Repeat this process until the 2 seconds finish, the GREEN light and WHITE light have both occurred 5 times, the light blinking can be achieved. The code of this part shows below (See Figure 14).

```
case 9:
    if(time <= 2)
    {
        if(interruptCount % 50 == 0)
            blink = ~blink;
    }
    XGpio_DiscreteWrite(&REGION_0_COLOUR,1, RED);
    XGpio_DiscreteWrite(&REGION_6_COLOUR,1, RED);
    XGpio_DiscreteWrite(&REGION_4_COLOUR,1, GREEN | blink);

    XGpio_DiscreteWrite(&REGION_1_COLOUR, 1, WHITE);
    XGpio_DiscreteWrite(&REGION_2_COLOUR, 1, WHITE);
    XGpio_DiscreteWrite(&REGION_7_COLOUR, 1, WHITE);
    XGpio_DiscreteWrite(&REGION_8_COLOUR, 1, WHITE);

    XGpio_DiscreteWrite(&LED_OUT, 1, 0x9000);
    break;
```

Figure 14 : Codes To Achieve Blink

Part 4: Conclusion

This project successfully simulated the traffic lights which are followed UK standard, using the ides of controller and state declarations to make the code more readable and easy to follow. In addition using switch-case is better to debug and clear than if-statement especially there are many scenarios in one project need to describe.