VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MICROPROCESSORS-MICROCONTROLLERS

Report

Traffic light and crossing road system for pedestrian

Advisor(s): Le Trong Nhan

Student(s): Dinh Le Minh Quan 2152262

Nguyen Phuoc Thinh 2153838

Le Minh Quy 2153758

HO CHI MINH CITY, DECEMBER 2023

Member list & Workload

No.	Full name	Student ID	Contribution
1	Dinh Le Minh Quan	2152262	100%
2	Nguyen Phuoc Thinh	2153838	100%
3	Le Minh Quy	2153758	100%

Contents

1 Introduction

This project aims to improve the safety and convenience of pedestrians in Vietnam, where the current traffic light systems do not accommodate their needs. The existing pedestrian lights are integrated with the vehicle lights and have the opposite signal, which can be confusing and dangerous for people who want to cross the road.

Our solution is to design a separate system for pedestrians that allows them to activate the light system by pressing a button. The light system will then display a clear signal for pedestrians to cross the road safely, while alerting the pedestrians with the buzzer. When no one is crossing the road, the light system will turn off automatically to save energy and reduce traffic congestion.

2 Functionality

In this project, the STM32F103RB is used to simulate the 2-way traffic light system, having some main features:

- **Automatic mode:** The system operates as normal. The light colors are red, yellow, and green.
- **Manual mode:** A button is used to hold the current status of traffic light.
- **Tuning mode:** This mode is used to modify the light timing length
  - one button to select the light you want to change
  - one button to increase light timing length
  - one button to set new light timing length
- **Pedestrian scramble:** when the button for pedestrian is pressed, its light is turned on and operates reversely to the light of vehicles. After one cycle (red and green or green only), its light will be turn off.

3 Design

3.1 System Design

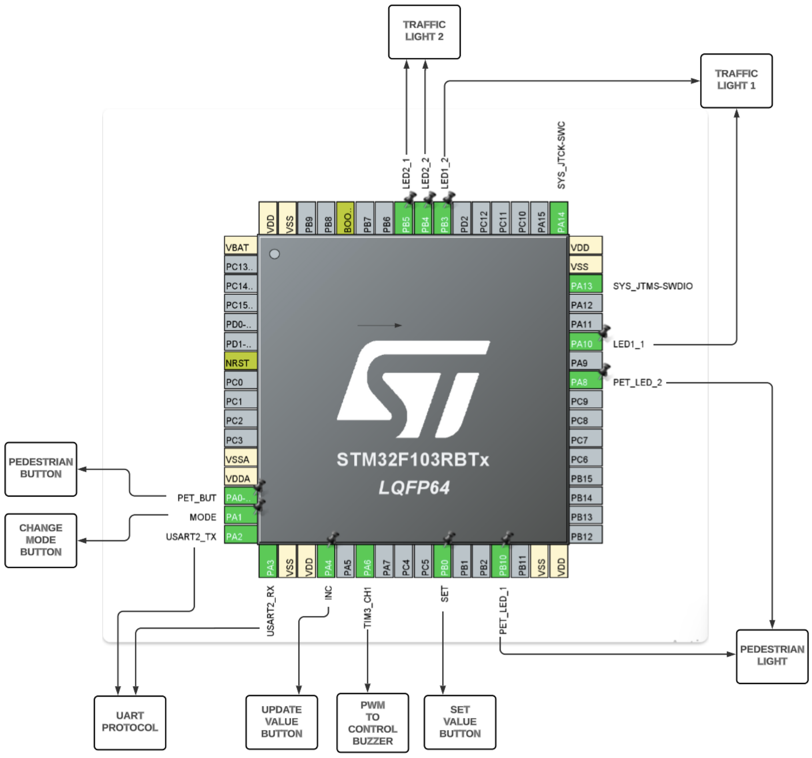

Figure 3.1: System design

Some of the key components in the system design:

- `STM32F103R1T-x LQFP64 microcontroller:` a 32-bit microcontroller with a variety of features, including a built-in real-time clock, timer, and analog-to-digital converter. It is a popular choice for embedded systems applications.
- `USART2 TX and RX pins` : These pins are used for serial communication with the UART display.
- `LED driver pins:` These pins are used to drive the TRAFFIC LIGHTs.
- `PET LED` : This pin is used to drive the PEDESTRIAN LIGHT.
- `PEDESTRIAN BUTTON` : This button is used to request a pedestrian walk signal (denote as **button 3**).

- `CHANGE MODE BUTTON` : this button is used to change the current mode of the system (denote as **button 0**)
- `UPDATE VALUE BUTTON` : this button is used to increase the current value of the light to 1 digit (denote as **button 1**)
- `SET VALUE BUTTON` : this button is used to set the value after modifying (denote as **button 2**)
- `BUZZER` :This buzzer is used to alert pedestrians(using PWM from TIM3_CH1 channel) when the walk signal is active.

3.2 Finite State Machine

In the design of figure 3.1, we have 2 traffic lights. And the design below is the logic we apply for Traffic Light 1. Initially, we declare 3 variables : *g_val, r_val, y_val* to set the time that the traffic light need to count in each color and the *led_status* to indicate the current state of the traffic light system.

Figure 3.2: System' FSM

In the beginning, the system will go through 4 states as the color change for 2 traffic light . For example, `RED_GREEN` is the state where light 1 is RED and light 2 is GREEN and the system will go through the color states as scheduled. If the button 0 is pressed, the system will jump to manual section and we use button 1 and 2 to adjust and set the value, then we need to continue to press button 0 until exit manual section. Additionally, while the system is running, if we press button 1, traffic light system will hold at current state and only follow normal mode when we press button 1 again.

Let's dive into what we operate inside each state:

- `INIT`:
  - Set Timer1 to count g_val seconds
  - Set led_status to `RED_GREEN` state.
- `RED_GREEN`:
  - onRED1()
  - onGREEN2()
  - `Next state event`:
    - `timer1Flag = 1 :` When the counter done counting, the timer1Flag is set to 1 then we set led_status to `RED_YELLOW` state; Set timer1 to count y_val seconds
    - `isButtonPressed(0)`: to `RED_MAN` state
    - `isButtonPressed(1)`: to `RED_GREEN_NO_TIME` state
- `RED_YELLOW`
  - onRED1()
  - onYELLOW2()
  - `Next state event`:
    - `timer1Flag = 1 :` When the counter done counting, the timer1Flag is set to 1 then we set led_status to `GREEN_RED` state. Set timer1 to count g_val seconds
    - `isButtonPressed(0)`: to `RED_MAN` state
    - `isButtonPressed(1)`: to `RED_YELLOW_NO_TIME` state
- `GREEN_RED`
  - onGREEN1()
  - onRED2()
  - `Next state event`:
    - `timer1Flag = 1 :` When the counter done counting, the timer1Flag is set to 1 then we set led_status to `YELLOW_RED` state. Set timer1 to count y_val seconds
    - `isButtonPressed(0)`: to `RED_MAN` state

    * `isButtonPressed(1)`: to `GREEN_RED_NO_TIME` state

- `YELLOW_RED`
  - onYELLOW1()
  - onRED2()
  - `Next state event:`
    * `timer1Flag = 1 :` When the counter done counting, the timer1Flag is set to 1 then we set led_status to `RED_GREEN` state. Set timer1 to count g_val seconds
    * `isButtonPressed(0)`: to `RED_MAN` state
    * `isButtonPressed(1)`: to `YELLOW_RED_NO_TIME` state
- `RED_MAN`
  - toggle RED led of 2 traffic lights
  - press button 1 to increase value
  - press button 2 to set value
  - `Next state event:`
    * `isButtonPressed(0)`: to `YELLOW_MAN` state
- `YELLOW_MAN`
  - toggle YELLOW led of 2 traffic lights
  - press button 1 to increase value
  - press button 2 to set value
  - `Next state event:`
    * `isButtonPressed(0)`: to `GREEN_MAN` state
- `GREEN_MAN`
  - toggle YELLOW led of 2 traffic lights
  - press button 1 to increase value
  - press button 2 to set value
  - `Next state event:`

    * `isButtonPressed(0)`: to `RED_GREEN` state

- `RED_GREEN_NO_TIME`
    - onRED1()
    - onGREEN2()
    - `Next state event:`
        * `isButtonPressed(1)`: to `RED_GREEN` state. Set timer1 to count g_val seconds
- `RED_YELLOW_NO_TIME`
    - onRED1()
    - onYELLOW2()
    - `Next state event:`
        * `isButtonPressed(1)`: to `RED_YELLOW` state. Set timer1 to count y_val seconds
- `GREEN_RED_NO_TIME`
    - onGREEN1()
    - onRED2()
    - `Next state event:`
        * `isButtonPressed(1)`: to `GREEN_RED` state. Set timer1 to count g_val seconds
- `YELLOW_RED_NO_TIME`
    - onYELLOW1()
    - onRED2()
    - `Next state event:`
        * `isButtonPressed(1)`: to `YELLOW_RED` state. Set timer1 to count y_val seconds

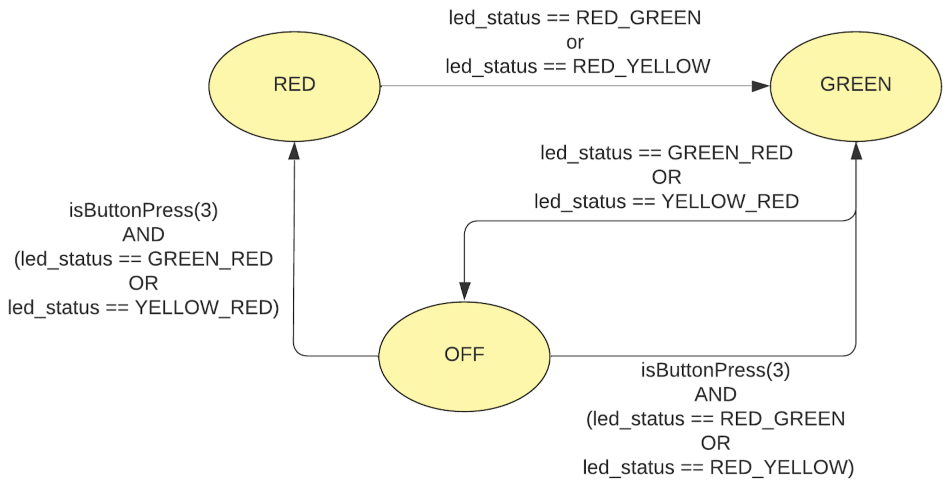

Figure 3.3: PEDESTRIAN FSM

The overall workflow is described as follow: If we push the button to turn on pedestrian light (in this case: button 3), it must be green at least once time, then turn off.

- `OFF`
  - Turn off the pedestrian light
  - `Next state event :`
    - `Button 3 is pressed and (led_status = GREEN_RED or led_status = YELLOW_RED)` : to `RED` state
    - `Button 3 is pressed and (led_status = RED_GREEN or led_status = RED_YELLOW)` : to `GREEN` state
- `RED`
  - Turn on `RED` light of the pedestrian light
  - `Next state event:`
    - `led_status = RED_GREEN or led_status = RED_YELLOW` : to `GREEN` state
- `GREEN`
  - Turn on `GREEN` light of the pedestrian light
  - Start the Buzzer
  - `Next state event`:
    - `led_status = GREEN_RED or led_status = YELLOW_RED` : to `OFF` state

3.3 Project Structure

```
.
├── Inc
│   ├── button.h
│   ├── buzzer.h
│   ├── fsm_manual.h
│   ├── fsm_traffic_light.h
│   ├── global.h
│   ├── main.h
│   ├── ped_fsm.h
│   ├── ped_manual.h
│   ├── physical.h
│   ├── scheduler.h
│   ├── software_timer.h
│   ├── stm32f1xx_hal_conf.h
│   ├── stm32f1xx_it.h
│   ├── traffic_no_time.h
│   └── uart.h
│
├── Src
│   ├── button.c
│   ├── buzzer.c
│   ├── fsm_manual.c
│   ├── fsm_traffic_light.c
│   ├── global.c
│   ├── main.c
│   ├── ped_fsm.c
│   ├── ped_manual.c
│   ├── physical.c
│   ├── scheduler.c
│   ├── software_timer.c
│   ├── stm32f1xx_hal_msp.c
│   ├── stm32f1xx_it.c
│   ├── syscalls.c
│   ├── sysmem.c
│   ├── system_stm32f1xx.c
│   ├── traffic_no_time.c
│   └── uart.c
│
```

Figure 3.4: Project files structure

4 Implementation

4.1 Scheduler pattern

```
#include "main.h"
/* USER CODE BEGIN Includes */
#include "fsm_traffic_light.h"
#include "fsm_manual.h"
#include "ped_fsm.h"
#include "ped_manual.h"
#include "buzzer.h"
#include "traffic_no_time.h"
/* Private variables ---------------------------------*/
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
UART_HandleTypeDef huart2;
int main(void)
{
  HAL_Init();
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM2_Init();
  MX_USART2_UART_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim2);
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
  //TASK INIT
  SCH_Add_Task(timerRun, 0, 1);
  SCH_Add_Task(getKeyInput, 0, 1);
  SCH_Add_Task(fsm_traffic_light, 0, 1);
  SCH_Add_Task(fsm_manual_run, 0, 1);
  SCH_Add_Task(ped_fsm, 0, 1);
  SCH_Add_Task(sendUART, 0, 100);
  SCH_Add_Task(traffic_no_time, 0, 1);
  setTimer3(50);
/* USER CODE BEGIN WHILE */
  while (1)
  {
      SCH_Dispatch_Tasks();
  }
}
```

Program 4.1: main.c

Main Function

- `HAL_TIM_Base_Start_IT(&htim2)`: Starts Timer 2 in interrupt mode.

- `HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1)`: Starts Timer 3 in PWM mode on channel 1.
- `Task Initialization`: This section adds tasks to the scheduler for various functionalities:
  - `timerRun`: Runs at every timer interrupt.
  - `getKeyInput`: Reads user input from buttons.
  - `fsm_traffic_light`: Runs the traffic light FSM.
  - `fsm_manual_run`: Runs the manual mode FSM.
  - `ped_fsm`: Runs the pedestrian FSM.
  - `sendUART`: Sends data via UART periodically.
  - `traffic_no_time`: Handles traffic light behavior when system holds at a specific state.

Main loop

The main loop runs indefinitely, repeatedly calling **SCH_Dispatch_Tasks()**, which is likely responsible for executing scheduled tasks.

4.2 Traffic light FSM

Firstly, we design an FSM that follows the transition between three colors of two traffic lights. The colors are red, yellow and green. The FSM has 4 states base on behavior of 1 light, each representing a combination of the colors of the two lights. For instance, RED_GREEN means that the first light is red and the second light is green. The FSM transitions from one state to another according to a predefined logic that ensures the safety and efficiency of the traffic flow.

```
#include "fsm_traffic_light.h"

// Allowed signal

void fsm_traffic_light(){
  switch(led_status){
  case INIT:
    //TODO
    offALL();

    //INIT state
    led_status = RED_GREEN;
```

```
    timerRoad1 = r_val;
    timerRoad2 = g_val;
    setTimer1(g_val*100);
    setTimer2(100);
    break;
  case RED_GREEN:
    onRED1();
    onGREEN2();

    //decrement of counter of each road
    if (timer2_flag == 1){
      setTimer2(100);
      timerRoad1--;
      timerRoad2--;
      if (timerRoad2 <= 0) timerRoad2 = y_val;
    }

    //update state
    if (timer1_flag == 1){
      setTimer1(y_val*100);
      led_status = RED_YELLOW;
    }

    if (isButtonPressed(0)){
      offALL();
      led_status = RED_MAN;
      setTimer5(1);

      timerRoad1 = r_val;
      timerRoad2 = 2;
      //reset button flag
      resetButton();
    }

    if (isButtonPressed(1)){
      led_status = RED_GREEN_NO_TIME;
    }

    break;
  case RED_YELLOW:
    onRED1();
    onYELLOW2();

    //decrement of counter of each road
    if (timer2_flag == 1){
      setTimer2(100);
      timerRoad1--;
      if (timerRoad1 <= 0) timerRoad1 = g_val;
      timerRoad2--;
```

```
      if (timerRoad2 <= 0) timerRoad2 = r_val;
    }

    //update state
    if(timer1_flag == 1){
      setTimer1(g_val*100);
      led_status = GREEN_RED;
    }

    if (isButtonPressed(0)){
      offALL();
      led_status = RED_MAN;
      setTimer5(1);

      timerRoad1 = r_val;
      timerRoad2 = 2;

      //reset button flag
      resetButton();
    }
    if (isButtonPressed(1)){
      led_status = RED_YELLOW_NO_TIME;
    }

    break;
  case GREEN_RED:
    onGREEN1();
    onRED2();

    //decrement of counter of each road
    if (timer2_flag == 1){
      setTimer2(100);
      timerRoad1--;
      if (timerRoad1 <= 0) timerRoad1 = y_val;
      timerRoad2--;
    }

    //update state
    if(timer1_flag == 1){
      setTimer1(y_val*100);
      led_status = YELLOW_RED;
    }

    if (isButtonPressed(0)){
      offALL();
      led_status = RED_MAN;
      setTimer5(1);

      timerRoad1 = r_val;
```

```
        timerRoad2 = 2;

        //reset button flags
        resetButton();
      }

      if (isButtonPressed(1)){
        led_status = GREEN_RED_NO_TIME;
      }

      break;
    case YELLOW_RED:
      onYELLOW1();
      onRED2();

      //decrement of counter of each road
      if (timer2_flag == 1){
        setTimer2(100);
        timerRoad1--;
        if (timerRoad1 <= 0) timerRoad1 = r_val;
        timerRoad2--;
        if (timerRoad2 <= 0) timerRoad2 = g_val;
      }

      if (timer1_flag == 1){
        setTimer1(g_val*100);
        led_status = RED_GREEN;
      }
      if (isButtonPressed(0)){
        offALL();
        led_status = RED_MAN;
        setTimer5(1);

        timerRoad1 = r_val;
        timerRoad2 = 2;
        resetButton();
      }
      if (isButtonPressed(1)){
        led_status = YELLOW_RED_NO_TIME;
      }
      break;
    default:
      break;
  }
}
```

Program 4.2: fsm_traffic_light.c

4.3 Light control manual FSM

This FSM is for controlling the Light manually. The Light has three colors: red, yellow and green. Each color has a timer that determines how long it stays on. The user can manipulate the timer using three buttons: change mode, increase and set. The change mode button (button 0) switches to the next color in the sequence. The increase button (button 1) adds one second to the current timer. The set button (button 2) confirms the current timer value and updates new value for traffic light system.

```
#include "fsm_manual.h"
void fsm_manual_run(){
  switch(led_status){
  case RED_MAN:
    if (timer5_flag == 1){
      setTimer5(25);
      if (toogleFlag == 0){
        onRED1();
        onRED2();
      }
      else {
        offALL();
      }
      toogleFlag = 1 - toogleFlag;
    }

    if (isButtonPressed(0) == 1){
      setTimer5(1);
      led_status = YELLOW_MAN;

      timerRoad1 = y_val;
      timerRoad2 = 3;

      resetButton();
    }

    if (isButtonPressed(1) == 1){
      timerRoad1++;
      if (timerRoad1 >= 100) timerRoad1=2;
    }

    if (isButtonPressed(2) == 1){
      r_val=timerRoad1;
    }
    break;

  case YELLOW_MAN:
    if (timer5_flag == 1){
      setTimer5(25);
```

```
      if (toogleFlag == 0){
        onYELLOW1();
        onYELLOW2();
      }
      else {
        offALL();
      }
      toogleFlag = 1 - toogleFlag;
    }

    if (isButtonPressed(0) == 1){
      setTimer5(1);
      led_status = GREEN_MAN;

      timerRoad1 = g_val;
      timerRoad2 = 4;

      resetButton();
    }

    if (isButtonPressed(1) == 1){
      timerRoad1++;
      if (timerRoad1 >= r_val) timerRoad1=1;
    }

    if (isButtonPressed(2) == 1){
      y_val=timerRoad1;
    }
    break;

  case GREEN_MAN:
    if (timer5_flag == 1){
      setTimer5(25);
      if (toogleFlag == 0){
        onGREEN1();
        onGREEN2();
      }
      else {
        offALL();
      }
      toogleFlag = 1 - toogleFlag;
    }

    if (isButtonPressed(0) == 1){
      led_status = RED_GREEN;
      g_val = r_val-y_val;
      timerRoad1 = r_val;
      timerRoad2 = g_val;
      setTimer1(g_val*100);
```

```
      setTimer2(100);
    }

    if (isButtonPressed(1) == 1){
      timerRoad1++;
      if (timerRoad1 >= r_val) timerRoad1=1;
    }

    if (isButtonPressed(2) == 1){
      g_val=timerRoad1;
      y_val=r_val-g_val;
    }
    break;

  default:
    break;
  }
}
```

Program 4.3: fsm_manual.c

4.4 Pedestrian manual

According to the led status, this FSM is the instruction to decide which color of the pedestrian light to display if we press button 3 to activate the pedestrian light.

```
#include "ped_manual.h"
void pedestrian_manual_fsm(){
  switch(led_status){
  case RED_GREEN:
    ped_status = GREEN;
    break;
  case RED_YELLOW:
    ped_status = GREEN;
    break;
  case GREEN_RED:
    ped_status = RED;
    break;
  case YELLOW_RED:
    ped_status = RED;
    break;
  default:
    break;
  }
}
```

Program 4.4: ped_manual.c

4.5 Pedestrian light FSM

This FSM is for the behavior of the pedestrian light. It has 3 state OFF, RED, and GREEN, especially there is a buzzer function to alert the pedestrian awareness of how much time left for the green light to cross the road.

```
#include "ped_fsm.h"

void ped_fsm(){
  if (isButtonPressed(3)) pedestrian_manual_fsm();
  switch(ped_status){
  case OFF:
    pedOff();
    __HAL_TIM_SetCompare (&htim3,TIM_CHANNEL_1,0);
    break;
  case RED:
    pedRed();
    if (led_status == RED_GREEN || led_status == RED_YELLOW)
   ped_status = GREEN;
    break;
  case GREEN:
    startBuzzer();
    pedGreen();
    if (led_status == GREEN_RED || led_status == YELLOW_RED)
   ped_status = OFF;
    break;
  }
}
```

Program 4.5: ped_fsm.c

4.6 Reset counter

This FSM is to hold our current state. If user presses button 1, state will be updated to previous state and system updates necessary timer of that state.

```
#include "traffic_no_time.h"

void traffic_no_time(){
  switch(led_status){
  case RED_GREEN_NO_TIME:
    onRED1();
    onGREEN2();
    if (isButtonPressed(1)){
      offALL();
      led_status = RED_GREEN;
      timerRoad1 = r_val;
```

```
        timerRoad2 = g_val;
        setTimer1(g_val*100);
        setTimer2(100);
      }
      break;
    case RED_YELLOW_NO_TIME:
      onRED1();
      onYELLOW2();
      if (isButtonPressed(1)){
        offALL();
        led_status = RED_YELLOW;
        timerRoad1 = y_val;
        timerRoad2 = y_val;
        setTimer1(y_val*100);
        setTimer2(100);
      }
      break;
    case GREEN_RED_NO_TIME:
      onGREEN1();
      onRED2();

      if (isButtonPressed(1)){
        offALL();
        led_status = GREEN_RED;
        timerRoad1 = g_val;
        timerRoad2 = r_val;
        setTimer1(g_val*100);
        setTimer2(100);
      }
      break;
    case YELLOW_RED_NO_TIME:
      onYELLOW1();
      onRED2();

      if (isButtonPressed(1)){
        offALL();
        led_status = YELLOW_RED;
        timerRoad1 = y_val;
        timerRoad2 = y_val;
        setTimer1(y_val*100);
        setTimer2(100);
      }
      break;
  }
}
```

Program 4.6: traffic_no_time.c

4.7 Buzzer

This function, called ‘startBuzzer‘, is responsible for continuously monitoring the remaining time for road 1 and uses the percentage of remaining time to control the duration and frequency of a buzzer sound. This effectively creates an audible countdown for the traffic light.

```
#include "buzzer.h"
#include <math.h>
void startBuzzer(){
  if(timer3_flag){
    double temp = (1-(double)timerRoad1/r_val)*10;
    char str[100];
    int res = (int)round(temp);
    HAL_UART_Transmit(&huart2, (void*)str, sprintf(str,"!RES
   =%d#\r\n",res), 100);
    if (sig) __HAL_TIM_SetCompare (&htim3,TIM_CHANNEL_1,res);
    else __HAL_TIM_SetCompare (&htim3,TIM_CHANNEL_1,0);
    sig = 1 - sig;
    double temp1 = ((double)timerRoad1/r_val)*10;
    int timeBuz = (int)round(temp1);
    setTimer3(timeBuz*10);
  }
}
```

Program 4.7: buzzer.c

4.8 UART

The function below allows us to transmit the value of the light to a PC terminal via UART communication protocol in STM32 microcontroller. The function takes the counter value as an argument and converts it to a string using sprintf. Then, it sends each character of the string through the UART interface using HAL_UART_Transmit function.

```
#include "uart.h"

void sendUART(){
  char str[100];
  HAL_UART_Transmit(&huart2, (void*)str, sprintf(str,"!
   TimerRoad1=%d#\r\n",timerRoad1), 100);
}
```

Program 4.8: uart.c

The STM32 microcontroller communicates with the computer terminal through the UART protocol, which requires a serial-to-USB converter. For this purpose, we use the TTL CP2102 module as a bridge, which connects the STM32 UART output pins to the computer USB_A port. This module allows us to send data from the microcontroller to the terminal software.